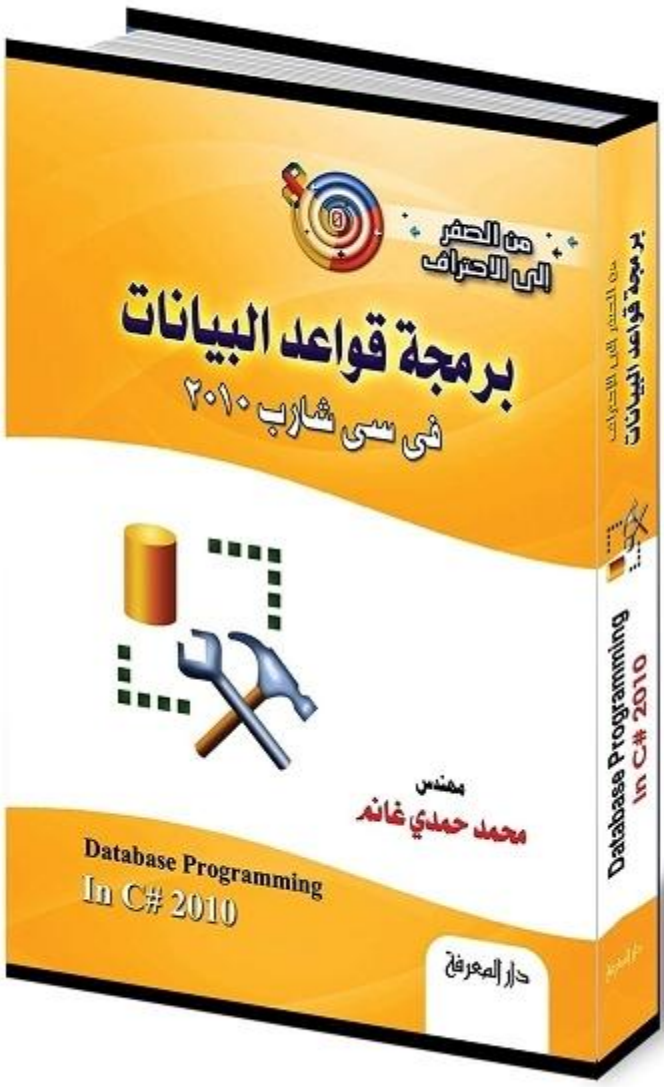

من الصفر
إلى الاحتراف
برمجة قواعد البيانات
فى سى شارب ٢٠١٠
مهندس
محمد حمدي غانم
Database Programming
In C# 2010
دار المعرفة
برمجة قواعد البيانات
من الصفر إلى الاحتراف
Database Programming
In C# 2010

من الصفر إلى الاحتراف:

ADO.NET

لمبرمجي سي شارب ٢٠١٠

بقلم:

م. محمد حمدي غانم

هذا الكتاب صدقة جارية على روح والدي:

أ. حمدي كامل الحديدي غانم

رحمه الله وغفر له وجعل مثواه الجنة

لهذا أرجو من كل من يستفيد به أن يتذكر أن أبي هو الـــذي ربـــاني وعلمنـــي ولولاه بعد توفيق الله ما خرج إلى الوجود هذا الكتاب وغيره من الكتب.

فادعوا له بالرحمة والمغفرة

ومن كان منكم في الحرمين الشريفين وكان قادرا على عمل عمرة له، فجزاه الله خيرا.

أدعو الله أن يكون هذا الكتاب وباقي كتبي من العلم الذي ينتفع به، وأن يجعل الله لأبي نصيبا من ثوابه، فيكون من عمله الذي لا ينقطع بموته.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته وقه من عذاب القبر وقه من عذاب النار،
وأدخله الجنة وأعلِ منزلته فيها
واحفظ والدتي وبارك في عمرها
اللهم ارحم والديّ كما ربياني صغيرا
آمين يا رب العالمين

للتواصل مع الكاتب:

- بريدي الالكتروني:

msvbnet@hotmail.com

- مدونتي:

http://mhmdhmdy.blogspot.com

- قناتي على يوتيوب (تحتوي على إلقاء أكثر من ٦٠ قصيدة بصوتي):

http://www.youtube.com/user/mhmdhmdy

- صفحتي الأدبية على فيسبوك:

https://www.facebook.com/Poet.Mhmd.Hmdy

- كتبي في مجال البرمجة بلغتي فيجوال بيزيك وسي شارب:

https://drive.google.com/drive/folders/1J21xi8Aw15BFSv-GUgVOElLuYM6zoNct

- صفحة فيجوال بيزيك وسي شارب على فيسبوك:

https://www.facebook.com/vbandcsharp

كتب مجانية للكاتب للتنزيل:

١ - كتاب: "خرافة داروين، حينما تتحول الصدفة إلى علم":

http://mhmdhmdy.blogspot.com/2013/11/blog-post_29.html

٢ - كتب أدبية (أشعار وقصص وروايات):

https://mhmdhmdy.blogspot.com/2018/10/blog-post_23.html

٣- المبرمج الصغير:

http://mhmdhmdy.blogspot.com.eg/2016/10/blog-post_9.html

٤- الرسم والتلوين والصور والمجسمات لمبرمجي فيجوال بيزيك دوت نت:

http://mhmdhmdy.blogspot.com.eg/2014/05/blog-post_26.html

٥- الرسم والتلوين والصور والمجسمات لمبرمجي سي شارب:

http://mhmdhmdy.blogspot.com.eg/2014/08/c.html

٦- من الصفر إلى الاحتراف: برمجة قواعد البيانات في فيجوال بيزيك دوت نت:

http://mhmdhmdy.blogspot.com.eg/2016/02/blog-post_28.html

كتب مطبوعة للكاتب:

١. **فيجيوال بيزيك وسي شارب:** طريقك المختصر للانتقال من إحدى اللغتين إلى الأخرى.
٢. من الصفر إلى الاحتراف: **فيجيوال بيزيك دوت نت ٢٠١٧.**
٣. من الصفر إلى الاحتراف: **سي شارب ٢٠١٧.**
٤. من الصفر إلى الاحتراف: **برمجة إطار العمل لمبرمجي فيجيوال بيزيك دوت نت وسي شارب.**
٥. من الصفر إلى الاحتراف: **برمجة نماذج الويندوز لمبرمجي فيجيوال بيزيك دوت نت وسي شارب.**
٦. المدخل العملي السريع إلى **فيجيوال بيزيك دوت نت ٢٠١٧.**
٧. المدخل العملي السريع إلى **سي شارب ٢٠١٧.**
٨. أساسيات WPF لمبرمجي **فيجيوال بيزيك دوت نت.**
٩. أساسيات WPF لمبرمجي **سي شارب.**

لقراءة مقدمة وفهرس كل كتاب:

https://drive.google.com/drive/folders/1J21xi8Aw15BFSv-GUgVOElLuYM6zoNct

لشراء هذه الكتب، يتم تحويل الثمن بحوالة بريدية داخل مصر، أو بويسترن يونيون من خارج مصر، ويتم إرسال الكتب بطرد بالبريد السريع.. لمزيد من التفاصيل أرسل رسالة بالكتب المطلوبة إلى:

msvbnet@hotmail.com

كتب أجهز لكتابتها في المرحلة القادمة بإذن الله:

- برمجة قواعد البيانات بـــ Entity Framework.
- إنشاء تقارير Report Viewer و Crystal Reports وعرضها وطباعتها.
- برمجة مواقع الويب بـــ ASP.NET MVC Core.
- المواضيع المتقدمة في برمجة إطار العمل.
- الوسائط المتعددة في WPF.
- برمجة مشاريع Windows Universal Applications.
- برمجة الاندرويد بـــ Xamarin.
- برمجة الشبكات بدوت نت.

سجلوا إعجابكم بصفحتي البرمجية لمتابعة صـــدور هـــذه الكتـــب بـــإذن الله، والاســـتفادة بالملاحظات البرمجية العملية التي أنشرها على الصفحة:

https://www.facebook.com/vbandcsharp

للتواصل مع باقي المبرمجين وتبادل الأسئلة والخبرات، يمكنكم الانضمام إلى هذه المجموعة:

https://www.facebook.com/groups/123809374886424/

محتويات الكتاب

ملحوظة:

تم نشر الفصول الأربعة الأولى في كتاب مستقل بعنوان:

إنشاء قواعد البيانات وكتابة استعلامات SQL

يمكنم تحميله من هنا:

https://drive.google.com/file/d/1H3FqC-jEXihVI5fx-7ZBFmVGJm2D7Oi3/edit?fbclid=IwAR31PkLyHT1QTN1xNoozTWsvkwQ5-uowwQzNarL4EhPULQsy4-CGBOl1Cv0

-٥-

تقنية ADO.NET

-٦-

كائن الاتصال Connection Object

فئة إعدادات نص الاتصال ConnectionStringSettings Class

واجهة الاتصال بقواعد البيانات IDbConnection Interface

فئة الاتصال DbConnection Class

فئة اتصال سيكيول SqlConnection Class

فئة خطأ سيكيول SqlError Class

-٧-

كائن الأمر Command Object

-٨-

قارئ البيانات DataReader

فئة قارئ البيانات DbDataReader Class

فئة قارئ بيانات سيكويل SqlDataReader Class

– ٩ –

مهيئ البيانات DataAdapter

واجهة مهيئ البيانات IDataAdapter Interface ١١٩

واجهة مهيئ بيانات قاعدة البيانات IDbDataAdapter Interface

فئة مهيئ البيانات DataAdapter Class

فئة مهيئ بيانات قاعدة البيانات DbDataAdapter Class

فئة مهيئ بيانات سيكويل SqlDataAdapter Class

التصارع على تحديث البيانات

معالج إعداد مهيئ البيانات Data Adapter Configuration Wizard

فئة باني أوامر قاعدة البيانات DbCommandBuilder Class

فئة باني أوامر سيكويل SqlCommandBuilder Class

واجهة مجموعة خرائط الجداول ITableMappingCollection

فئة مجموعة خرائط الجداول DataTableMappingCollection Class

واجهة خريطة الجدول ITableMapping Interface

فئة خريطة الجدول DataTableMapping Class

واجهة مجموعة خرائط العمود IColumnMappingCollection

فئة مجموعة خرائط العمود DataColumnMappingCollection

واجهة خريطة العمود IColumnMapping Interface

فئة خريطة العمود DataColumnMapping Class

-١٠-

مصانع المزودات Provider Factories

-١١-

مجموعة البيانات DataSet

-١٢-

الجداول والعلاقات والقيود

-١٣-

عروض البيانات Data Views

-١٤-

ربط البيانات Data Binding

فئة مجموعة ارتباطات الأداة ControlBindingsCollection Class

فئة الارتباط Binding Class

سجل معلومات عنصر الربط BindingMemberInfo Structure

فئة محتوى الربط BindingContext Class

فئة أساس مدير الربط BindingManagerBase Class

فئة مدير الخاصية PropertyManager Class

فئة مدير التسلسل CurrencyManager Class

ربط الأدوات في وقت التصميم

ربط مربعات القوائم Binding List Boxs

معالج تهيئة مصادر البيانات Data Source Configuration Wizard

متصفح مصادر البيانات

واجهة مزود مدير التسلسل ICurrencyManagerProvider Interface

واجهة إلغاء إضافة الجديد ICancelAddNew Interface

واجهة إطلاق أحداث التغير IRaiseItemChangedEvents Interface

فئة قائمة الربط عامة النوع BindingList<T> Class

واجهة مصدر القائمة IListSource Interface

فئة مصدر الربط BindingSource Class

فئة مساعد ربط القوائم ListBindingHelper Class

فئة موجه الربط BindingNavigator Class

ملحوظة:

تم نشر الفصول ١٥ و ١٦ و ١٧ والملحق ١ في كتاب مستقل بعناوان:

جدول عرض البيانات DataGridView

يمكنم تحميله من هنا:

https://drive.google.com/file/d/1tm27L0yGX1RA__vxXng0t5ny3XuJUFv_/view?fbclid=IwAR0IM7zdX9dqkWPXttQyQU6s-FPna2m0hshLq9itiog9SS6PISFdes7Gm_0

ملحق ٢

أنواع بيانات سيكويل المدارة

Managed SQL Data Types

ملحق: ٣

إعداد تطبيق قواعد البيانات على جهاز العميل

مقدمة

بسم الله، والحمد لله، والصلاة والسلام على رسول الله، وبعد:

يعلمك الكتاب كيف تتعامل مع قواعد البيانات من داخل مشاريع سي شارب باستخدام تقنية ADO.NET، لتستطيع الاتصال بقاعدة البيانات، وطلب السجلات منها، وكيف تقوم بحفظها مرة أخرى في قاعدة البيانات إذا دخلت عليها أية تعديلات.

ويعلمك الكتاب أيضا كيف تعرض البيانات للمستخدم من خلال تقنية الربط Binding، ويشرح بالتفصيل أهم الأدوات المخصصة لهذا الغرض، مثل موجه الربط BindingNavigator ومصدر الربط BindingSource.

***

ويشرح الكتاب بالتفصيل أكثر من ٥٠ مشروعا متنوعا تغطي محتوياته، لتتعلم من خلالها:

- كيف تحصل على البيانات من قواعد البيانات بمختلف الطرق، سواء باستخدام قارئ البيانات DataReader أو مهيئ البيانات DataAdapter أو مهيئ الجدول TableAdapter.
- كيف تحتفظ بالبيانات في الذاكرة، باستخدام مجموعة بيانات DataSet سواء كانت عادية أو محددة النوع Typed.
- كيف تنقل البيانات بين نوعين مختلفين من قواعد البيانات.
- كيف تحفظ البيانات الثنائية Binary Data في ملفات مستقلة على الخادم خارج قاعدة البيانات في SQL Server 2008.

- كيف تعرّف المعاملات Parameters والمعاملات الجدولية Table-Valued Parameters، وكيف تستخدمها لتمرير البيانات إلى الإجراءات المخزنة في Sql Server 2008.
- كيف تحمي قاعدة البيانات من القراصنة الذين يحاولون دس الاستعلامات SQL Injection.
- كيف تقرأ البيانات الثنائية والنصية الضخمة تتابعيا Sequentially على صورة أجزاء في SQL Server 2008.
- كيف تنشئ الإجراءات المخزنة في Access.
- كيف تحفظ البيانات في ملف XML وكيف تستعيدها منه مرة أخرى.
- كيف تستخدم مخطط XML لإنشاء مجموعات بيانات خاصّة Custom DataSet لا تعتمد على قاعدة بيانات.
- كيف تتعامل مع علاقة واحد بمتعدد One-To-Many Relation، وعلاقة متعدد بمتعدد Many-To-Many Relation، والعلاقة الذاتية Self Relation.
- كيف تستخدم مصانع المزودات Provider Factories لكتابة فئات عامة قادرة على التعامل مع أي نوع من قواعد البيانات، ما يختصر الكود الذي تكتبه، ويمهد لك الطريق لإنشاء مشاريع متعددة الطبقات N-Tier Applications.
- كيف تحل مشاكل تصارع أكثر من مستخدم على حفظ البيانات في نفس اللحظة باستخدام التطابق المتفائل Optimistic Concurrency.
- كيف تعرض البيانات في اللافتات ومربعات النص والقوائم والجداول، وكيف تربط كل هذه العناصر معا.

وغير هذا الكثير.

***

ويغطي هذا الكتاب بالتفصيل حوالي ١٣٥ واجهة وفئة وسجلا من مكتبة إطار العمل، مخصصة للتعامل مع تطبيقات قواعد البيانات، شارحا خصائص ووسائل وأحداث هذه المكونات بالتفصيل.. لهذا يعتبر الكتاب مرجعا مفصلا مبوبا، يمكن لقارئه الرجوع إليه عند

البحث عن تفاصيل أي فئة أو خاصية أو وسيلة أو حدث، في نفس الوقت الذي يجعله صالحا للقراءة ككتاب تعليمي عملي مرتب من الأسهل إلى الأصعب، ينقل إلى المبرمج في صفحات معدودات خبرة سنوات في برمجة تطبيقات قواعد البيانات، ويرشده إلى كيفية حل المشكلات غير المتوقعة التي تواجهه في هذا المجال، وكيف يحسن أداء برنامجه بتوفير أكبر قدر من الذاكرة، وكيف يحافظ على كفاءة خادم البيانات، بتقليل عدد الاتصالات ووقت كل اتصال بقدر الإمكان.

باختصار: هذا هو الكتاب الذي تبحث عنه.

والله ولي التوفيق

<u>لمن هذا الكتاب:</u>

رغم أن هذا الكتاب يفترض أن قارئه لا يمتلك أية معرفة مسبقة بقواعد البيانات والبرامج التي ينشئها بها، فإنه على الجانب الآخر، يشترط في قارئه أن يكون على دراية بلغة سي شارب #C، وأن يجيد المتطلبات التالية:

- أساسيات كتابة الكود بلغة سي شارب، كتعريف المتغيرات وكتابة جمل الشرط وحلقات التكرار Loops، وكتابة واستدعاء الدوال Functions.
- أساسيات ومفاهيم البرمجة الموجهة بالكائنات OOP، كالفئات Classes والواجهات Interfaces والوراثة Inheritance.
- أساسيات التعامل مع إطار العمل، وفئاته الرئيسية، خاصة المجموعات Collections والملفات Files وفئات معلومات الثقافة CultureInfo.
- أساسيات التعامل مع مشاريع الويندوز، والأدوات المختلفة كمربع النص TextBox ومربع الاختيار CheckBox والقوائم Lists.

فإذا لم تكن تجيد هذه الأساسيات، فننصح بقراءة القسم الأول من كتابنا "المدخل العملي السريع إلى سي شارب"، فهو يغطي هذه المواضيع باختصار من خلال إنشاء مشروع عملي كامل مشروح بالتفصيل.. أما النصف الثاني من الكتاب، فيشرح مشروع قواعد

بيانات كاملا مكتوبا بتقنية LinQ To SQL وهي غير مشروحة في الكتاب الـذي تقـرؤه الآن.. وهذا معناه أن كتاب المدخل العملي مكمل لهذا المرجع، فهـو مـن جهـة يشـرح مشروع قواعد بيانات واحدا كبيرا بينما يستعين المرجع الذي بين يديك بعشرات المشـاريع الصغيرة لشرح محتواه، كما أن هذا المرجع يشرح تقنية ADO.NET بينما يعطيك كتـاب المدخل العملي فكرة جيدة عـن اسـتخدام النمـوذج التصـوري Conceptual Model باستخدام تقنية LinQ To SQL.

الرموز المستخدمة في هذا الكتاب:

	سجل Structure.
	فئة Class.
	واجهة Interface.
	ثابت Constant.
	خاصية Property يمكنك قراءة أو تغيير قيمتها.
	خاصية للقراءة فقط Read Only Property.
	وسيلة Method.
	معامل Operator.
	حدث Event.
	هذا العنصر ثابت Static، يمكن استخدامه عبر اسم الفئة مباشرة.

ملحوظة:

تم نشر الفصول الأربعة الأولى في كتاب مستقل بعنوان:

إنشاء قواعد البيانات وكتابة استعلامات SQL

يمكنم تحميله من هنا:

https://drive.google.com/file/d/1H3FqC-jEXihVI5fx-7ZBFmVGJm2D7Oi3/edit?fbclid=IwAR31PkLyHT1QTN1xNoozTWsvkwQ5-uowwQzNarL4EhPULQsy4-CGBOl1Cv0

– ٥ –

تقنية ADO.NET

الخادم Server والعميل Client:

الخادم Server هو حاسوب توجد عليه قاعدة البيانات، ويعمل عليه SQL Server، لهذا فهو يقوم بقراءة البيانات المطلوبة وإرسالها إلى المستخدم أو استقبال البيانات الواردة من المستخدم وحفظها في قاعدة البيانات.

بينما العميل Client هو أيّ جهاز حاسوب آخر يوجد عليه برنامج قواعد البيانات الذي كتبته أنت، ويقوم بالاتصال بخادم سيكيول لطلب البيانات أو حفظها عليه.

ويمكن أن يكون هناك آلاف العملاء Clients، كلّ منهم يحاول الاتصال بقاعدة البيانات على الخادم في نفس الوقت، وطلب البيانات منها لمعالجتها على أجهزتهم، ثمّ إرسال أيّ تعديلات تمّ إجراؤها عليها إلى الخادم مرّة أخرى، ليتمّ حفظها في قاعدة البيانات.

ويقدم لنا نموذج الخادم والعميل الميزاتِ التالية:

- وجود قاعدة البيانات على الخادم يوفر لمستخدميها مساحة التخزين (بدلا من وضعها على أجهزة كل المستخدمين)، خاصّة حينما تكون قاعدة البيانات عملاقة.
- وجود قاعدة البيانات على الخادم يترك للجهة المسئولة عنها مهمّة تحديثها باستمرار، وهو أفضل من اضطرار كلّ مستخدم إلى شراء نسخة حديثة من قاعدة البيانات كلّ فترة.
- وجود قاعدة البيانات على الخادم يضمن مشاركتها بين مئات المستخدمين، مما يضمن مساهمتهم في إضافة البيانات وقدرة كل منهم على رؤية التعديلات التي أجراها الآخر، بينما لو كانت قاعدة البيانات على جهاز كل منهم بمفرده، فلن

يمكنهم العمل الجماعي عليها، وهذا لا يناسب نشاط الشركات التجارية والمؤسسات المالية.

- إجراء العمليّات على البيانات على جهاز العميل بعد الحصول عليها من الخادم، يكون أسرع بكثير من تنفيذ البرنامج على الخادم ثمّ إرسال الناتج إلى العميل، وذلك لأنّ هناك عددا ضخما من المستخدمين الذين يجرون آلاف العمليّات في نفس اللحظة.

وينظم خادم سيكويل عمليات الاتصال مع العملاء، حيث يخصص لكل اتصال عملية فرعية Thread للقراءة أو الكتابة في قاعدة البيانات.. ويتوقف عدد الاتصالات المتاحة في نفس اللحظة على حجم الذاكرة المؤقتة RAM الموجودة على الجهاز الخادم وقوة المشغل الدقيق الخاص به.. وفي حالة ازدياد الضغط على الخادم يقوم بتأجيل الاستجابة لبعض العملاء إلى حين الانتهاء من خدمة العملاء السابقين، مما قد يؤدي إلى بطء برنامجك وتضايق مستخدميه بسبب تعطله عن الاستجابة لفترات طويلة.. لهذا تقع على برنامجك مسئوليّة ضمان كفاءة عمليات الاتصال بقاعدة البيانات، بمراعاة ما يلي:

- ألا يطلب برنامجك بيانات لا ضرورة لها.. فإن كنت تحتاج مثلا إلى حقل أو حقلين من الجدول، فما الداعي لأن تقرأ كل الحقول؟
- التأكد من كتابة أقصر وأكفأ استعلامات SQL ليكون تنفيذها أسرع فلا ترهق الخادم.
- الاحتفاظ ببعض البيانات المجهزة Cashed على جهاز العميل بدلا من إعادة طلبها أكثر من مرة في فترات زمنية صغيرة، وذلك إذا كنت تضمن عدم تغير هذه البيانات بسرعة كبيرة.. وإذا كانت هذه البيانات ستظل ثابتة لجميع العملاء لفترة طويلة، فيمكن تجهيزها على الخادم وإرسالها إليهم مباشرة كلما طلبوها بدون إعادة تنفيذ الاستعلام، ولا يتم تحديث البيانات المجهزة إلا إذا حدث تغيير فيها في قاعدة البيانات.

مثل هذا التنظيم يضمن تخفيف عبء هائل من على خادم سيكويل وتقليل جمل SQL التي ينفذها، وبالتالي يوفر قدرة المشغل الدقيق Processor والذاكرة RAM الخاصة بالحاسوب الذي يعمل عليه خادم سيكويل، ليستطيع تنفيذ عمليات أخرى.

كما أن هناك بعض التحدّيات التي تواجه المبرمج وهو يكتب برنامجا يتعامل مع الخادم، مثل التعارض الذي يمكن أن ينتج عندما يحذف أحد المستخدمين بعض السجلات، بينما مستخدم آخر يحدّث قيمها!.. أو عندما يحاول أكثر من مستخدم تحديث نفس السجلات بطرق مختلفة في نفس الوقت.. وسنرى كيف نواجه مثل هذا الأمر لاحقا.

تقنية ADO.NET:

الأحرف ADO هي اختصار المصطلح "كائن البيانات الفعال" ActiveX Data Object، وهي تقنية برمجية ظهرت في فيجيوال ستديو ٦، تقدم جميع الفئات Classes اللازمة للاتصال بقاعدة البيانات وطلب البيانات منها وحفظها فيها.. وتفترض هذه التقنية أنّ العميل سيظل على اتصال بقاعدة البيانات طوال مدّة تعامله معها سواء على نفس الجهاز أو عبر الشبكة، حيث يحصل على البيانات من أيّ جدول يريده ويحدثها عبر نفس الاتصال.

لكن هذه الطريقة كانت عقيما، ففتح الاتصال طوال الوقت مع قاعدة البيانات عبر الشبكة غير عمليّ لأنه يعطل مستخدمين آخرين عن الاتصال بقاعدة البيانات عند تجاوز عدد المتصلين في نفس اللحظة الحد المسوح به، كما أنّ إجراء العمليّات عبر الشبكة أبطأ من إجرائها على جهاز المستخدم.. كل هذا جعل المبرمجين يستخدمون تقنية ADO بطريقة غريبة، فقد كانوا يتصلون بقاعدة البيانات وينسخون البيانات المطلوبة إلى أجهزتهم، ثمّ يغلقون الاتصال ويجرون العمليّات المطلوبة على البيانات على أجهزتهم.. وإذا كانت هناك تغييرات يجب حفظها في قاعدة البيانات، يتصلونَ بقاعدة البيانات مرّة أخرى ويحفظون البيانات ثمّ يغلقون الاتصال.

وقد أخذت ميكروسوفت هذا الأمر بعين الاعتبار، وطورت تقنية ADO مع ظهور فيجيوال ستديو دوت نت ٢٠٠٢، وصارت التقنية الجديدة تحمل الاسم ADO.NET، فصار

بالإمكان التعامل مع البيانات بعد إغلاق الاتصال فيما عرف باسم "التعامل المنفصل" Disconnected Mode، حيث يتصل المستخدم بقاعدة البيانات ويقوم بتحميل البيانات منها إلى الذاكرة ويغلق الاتصال، ليتعامل مع هذه البيانات على جهازه، وعندما يريد حفظ التغييرات، يفتح الاتصال مرّة أخرى لنقل البيانات إلى قاعدة البيانات.

وفيما يلي تلخيص للخطوات التي تقوم بها عبر تقنية ADO.NET للحصول على البيانات وتحديثها:

- في البداية عليك أن تقوم بالاتصال بقاعدة البيانات بواسطة كائن الاتصال Connection object.. هذا الكائن يتيح لك توضيح اسم قاعدة البيانات وكيفية الاتصال بها.
- بعد هذا عليك استخدام كائن الأمر Command Object، الذي يتولى تنفيذ جملة الاستعلام SQL Query عبر الاتصال المفتوح.
- بعد هذا يكون أمامك أحد اختيارين:

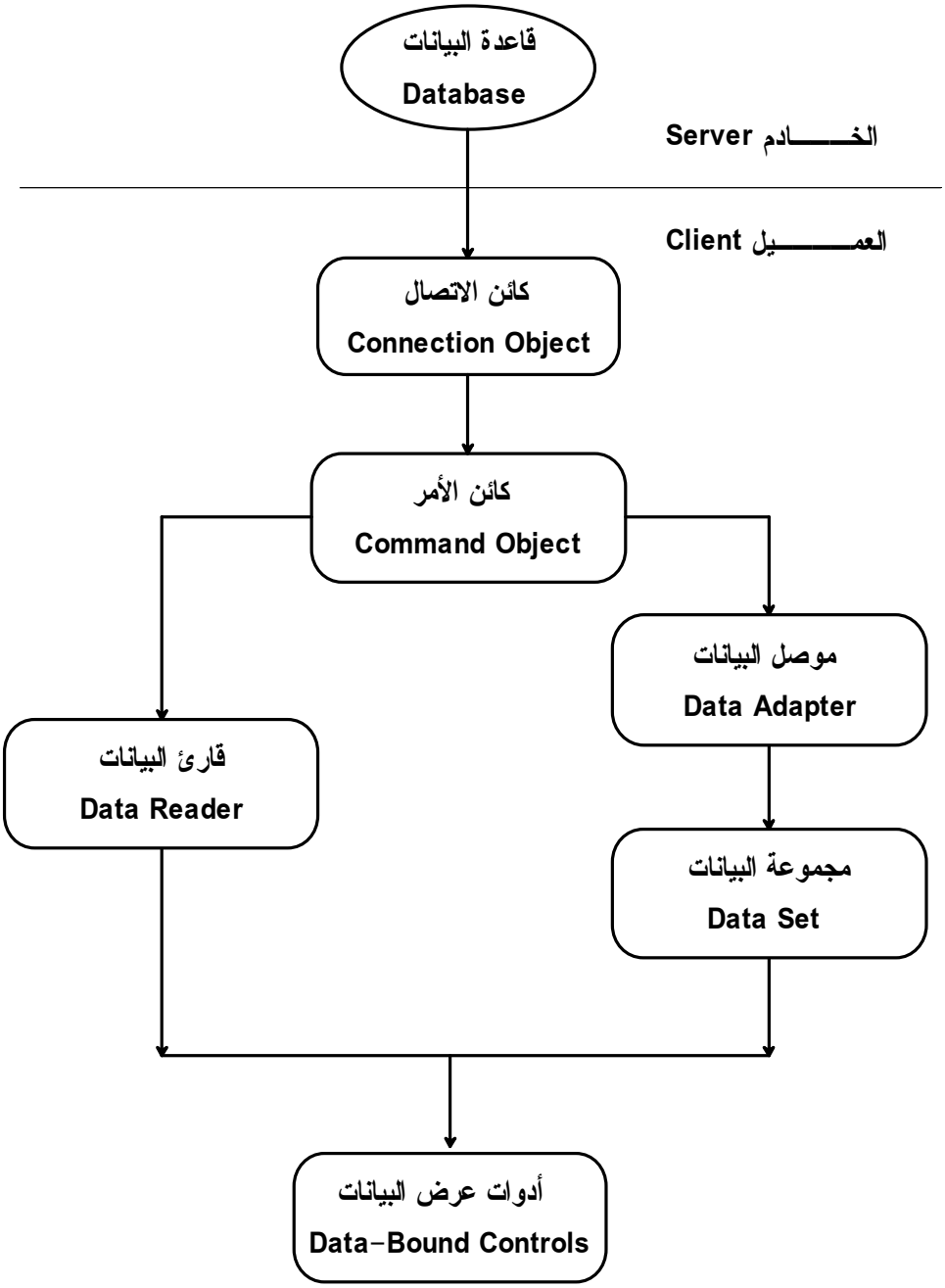
١. فإما أن تستخدم قارئ البيانات Data Reader لقراءة نتائج الاستعلام مباشرة دون حفظها على جهاز العميل.

٢. وإما أن تستخدم "مهيئ البيانات" Data Adapter لحفظ نتائج الاستعلام على جهاز العميل في مجموعة البيانات Data Set، التي يمكن القول إنّها صورة مصغّرة من قاعدة البيانات، تحتوي على الجداول والعلاقات Relations، والقيود Constraints المفروضة على قيم الحقول، بما في ذلك فرض التكامل المرجعي Referential Integrity بين الجداول.

- بعد هذا عليك إغلاق الاتصال، ومعالجة البيانات في برنامجك.. ويمكنك عرض البيانات للمستخدم لقراءتها أو تعديلها باستخدام أدوات ربط البيانات Data Bound Controls.
- وإذا كانت هناك أية تغييرات أجراها المستخدم على البيانات وتريد حفظها في قاعدة البيانات، فعليك استخدام كائن الاتصال مرة أخرى للاتصال بها، واستخدام كائن الأمر أو مهيئ البيانات لتنفيذ استعلام التحديث.

كما ترى: يعتمد هذا التنظيم على تقسّيم العمل إلى طبقات Layers مستقلّة في وظيفتها، وبهذا يسهل عليك التعديل في أي طبقة دون هدم الطبقات السابقة أو التالية لها، مما يوفر الوقت والجهد.

والشكل التالي يلخّص هذه التقنية:



وتوجد فئات ADO.NET في نطاقات الأسماء Namespaces التالية:

- System.Data
- System.Transactions
- Microsoft.SqlServer.Server

لغة XML:

الحروف XML هي اختصار للتعبير "لغة التوصيف القابلة للتمديد" Extensible Markup Language، وهي طريقة لتمثيل أيّ بيانات لها تركيب منظم، وذلك بتحويلها إلى نصّ يعبّر عنها.. لهذا تصلح لغة XML للتعبير عن أيّ نوع من أنواع البيانات، كالجداول والصور وغيرهما.

ورغم أنّ الملفّات النصيّة أكبر حجما من الملفّات الثنائيّة Binary Files، إلا إنّ الأولى صالحة للتعامل مع أيّ برنامج بل مع أيّ نظام تشغيل، دون الوقوع في مشاكل اختلاف طرق تمثيل البيانات الثنائية.. لهذا صارت لغة XML في السنوات الأخيرة أنسب وسيلة لنقل البيانات عبر الإنترنت، لأنها تتجاوز مشاكل عدم التوافق بين التطبيقات وأنظمة التشغيل المختلفة.. ولهذا تستخدم تقنية ADO.NET لغة XML في نقل البيانات بين الخادم والعميل.

ورغم أن إطار العمل يقدم دعما كاملا للغة XML ويتيح لك كتابتها وقراءتها، إلا أنك لن تحتاج إلى هذا عند التعامل مع قواعد البيانات، لأنّ تقنية ADO.NET تستخدم لغة XML كطبقة داخليّة، فهي تنتجها وتقرأ البيانات منها بطريقة آليّة.

ومن الإمكانيّات التي تتيحها لك لغة XML، قدرتك على استخدامها لإنشاء مجموعة بيانات DataSet بدون الاعتماد على أيّ قاعدة بيانات.. في هذه الحالة ستوضع البيانات في الذاكرة، ولو شئت الاحتفاظ بها، فيمكنك حفظها في ملفّ، ثمّ إعادة تحميلها مرّة أخرى حينما تريد.. وسنتعرف على هذا بتفصيل اكبر لاحقا.

مزودات قواعد البيانات Database Providers:

توفر تقنية ADO.NET عدة مزودات Providers للتعامل مع أنواع مختلفة من قواعد البيانات.. وهذه المزودات هي:

١- ODBC:

اسم هذا المزود هو اختصار للمصطلح "التواصل المفتوح مع قواعد البيانات":
Open **D**ata**b**ase **C**onnectivity

وقد طورت ميكروسوفت هذه التقنية ــ بالتعاون مع آخرين، عام ١٩٩٢، لتوفر طريقة عامة للتعامل مع قواعد البيانات بغض النظر عن لغة البرمجة المستخدمة ونظام التشغيل الذي تعمل عليه، وتطبيق قواعد البيانات المستخدم.

وتوجد فئات هذا المزود في النطاق:

System.Data.ODBC

٢- OLE DB:

اسم هذا المزود هو اختصار للمصطلح "قاعدة بيانات ربط وتضمين الكائنات":
Object **L**inking and **E**mbedding **D**ata**b**ase

وهو مزوّد Provider بنته ميكروسوفت باستخدام تقنية COM كتطوير وتحسين لتقنية ODBC، للتعامل بطريقة عامة مع أي نوع من أنواع قواعد البيانات، لهذا تستطيع استخدامه للتعامل مع آكسيس (فليس لقواعد بياناته مزود خاص بها)، وكذلك مع قواعد بيانات سيكويل سيرفر وأوراكل (رغم أن لكل منهما مزودا خاصا بهما)، ومع أي نوع آخر من أنواع قواعد البيانات، حتى ولو لم تكن تدعم استخدام لغة SQL مثل الجداول الشاملة Spreadsheets الخاصة بتطبيق إكسيل Excel.

وتوجد فئات هذا المزود في النطاق:

System.Data.OleDb

٣- SQL Server:

توفر دوت نت دعما خاصا لسيكويل سيرفر باعتباره أهم تطبيقات قواعد البيانات التي أنتجتها ميكروسوفت بعد انتشار استخدام الشبكات والإنترنت في عالم التجارة والأعمال.

وتوجد فئات هذا المزود في النطاقات التالية:

System.Data.SqlClient	فئات مزود سيكويل سيرفر.
System.Data.SQL	يقدم بعض الوظائف الخاصة بسيكويل سيرفر.
System.Data.SqlTypes	يحتوي على فئات تمثل أنواع البيانات Data Types الخاصة بسيكويل سيرفر، ليمكنك استخدامها بدلا من أنواع البيانات الموجودة في إطار العمل
Microsoft.SqlServer.Server	يحتوي على الفئات اللازمة لتشغيل سيكويل سيرفر في دوت نت.

٤- SQL Server Compact 3.5:

يتيح لك هذا المزود التعامل مع قواعد البيانات المنشأة بالنسخة الخفيفة من سكيويل سيرفر SQL Server Compact Edition، المخصصة لإنشاء قواعد بيانات للأجهزة الكفية المحمولة، التي تتعامل مع النسخة الخفيفة من الويندوز Windows CE والنسخة الخفيفة من إطار العمل .NET Compact Framework.

وتوجد فئات هذا المزود في النطاق:

System.Data.SqlServerCe

لكن استخدام هذا النطاق يتطلب منك أولا إضافة مرجع إليه في برنامجك، علما بأنه يوجد في الملف:

system.data.sqlserverce.dll

٥- Oracle:

قدمت ميكروسوفت منذ إصدار دوت نت ٢٠٠٣ دعما للتعامل مع قواعد بيانات أوراكل، فهي تمتاز بالقوة والشهرة والانتشار.

وتوجد فئات هذا المزود في النطاق: System.Data.OracleClient

لكن استخدام هذا النطاق يتطلب منك أولا إضافة مرجع إليه في برنامجك، علما بأنه يوجد في الملف:

System.Data.OracleClient.dll

وعليك أن تلاحظ أن جميع هذه المزودات توفر نفس أدوات الاتصال بقاعدة البيانات (كائن الاتصال Connection، كائن الأمر Command، مهيئ البيانات DataAdapter، مجموعة البيانات DataSet، قارئ البيانات DataReader... إلخ)، لكنّ كلا منها يبدأ باختصار يوضح نوع المزود، مثل:

SqlCeConnection SqlConnection OdbcConnection OleDbConnection OracleConnection	كائنات الاتصال
SqlCeCommand SqlCommand OdbcCommand OleDbCommand OracleCommand	كائنات الأمر
SqlCeDataAdapter SqlDataAdapter OdbcDataAdapter OleDbDataAdapter OracleDataAdapter	مهيئات البيانات
SqlDataSet OdbcDataSet OleDbDataSet OracleDataSet	مجموعات البيانات

قارئات البيانات	SqlCeDataReader SqlDataReader OdbcDataReader OleDbDataReader OracleDataReader

ونظرا لأنه لا توجد فروق تذكر بين أنواع الكائنات الخاصة بأحد المزودات والكائنات الخاصة بنوع آخر، فسنقتصر في هذا الكتاب على شرح مزود سيكويل سيرفر، لأن استخدامك لباقي أنواع المزودات لن يختلف في شيء، سوى في تغيير نطاق الاسم والبادئة التي تسبق اسم كل كائن من كائنات التعامل مع قاعدة البيانات!

-٦-

كائن الاتصال
Connection Object

يتيح لك هذه الكائن الاتصال بقاعدة بيانات تعمل على الخادم.. وكما ذكرنا سابقا، سنركز هنا على دراسة الفئة sqlConnection، لهذا لا تنسَ إضافة جملة الاستخدام التالية أعلى صفحة الكود:

using System.Data.SqlClient;

نص الاتصال Connection String:

للاتصال بسيكويل سيرفر، يجب أن ترسل إليه نصا يحتوى على البيانات اللازمة، مثل اسم قاعدة البيانات، واسم المستخدم وكلمة السر.. ويتكون نص الاتصال من مجموعة من القيم، يفصل بين كل منها العلامة ; وذلك على الصيغة:

Property1 = Value1; Property2 = Value2; ……
PropertyN = ValueN

على سبيل المثال، النص التالي هو نص الاتصال بقاعدة بيانات الكتب على سيكويل سيرفر:

Data Source = .\SQLEXPRESS;
AttachDbFilename = C:\Books.mdf;
Integrated Security = true;
Connect Timeout = 30;

ملحوظة:
عند بناء نص الاتصال بمزود ODBC، عليك وضع القيم بين قوسين متعرجين { }، على الصيغة: Property1 = {Value1}; Property2 = {Value2}; ……. PropertyN = {ValueN} بينما في باقي المزودات يمكنك استخدام علامات التنصيص بدلا من الأقواس المتعرجة.

وتختلف بعض الخصائص المرسلة عبر نص الاتصال، تبعا لنوع مـــزود قاعـــدة البيانـــات المستخدم.. وقد كانت كتابة نص الاتصال تمثل مشكلة قبل ظهور الإصدار الثاني من إطار العمل مع دوت نت ٢٠٠٥، حيث وفر إطار العمل فئة خاصة تسمى "باني نص الاتصـــال" DbConnectionStringBuilder، ومنها تم اشتقاق فئة لبناء نص اتصال كل مزود مـــن مزودات قاعدة البيانات، وهي:

الفئة	وظيفتها
SqlConnectionStringBuilder	باني نص اتصال سيكويل سيرفر.
OleDbConnectionStringBuilder	باني نص اتصال OLEDB.
OdbcConnectionStringBuilder	باني نص اتصال ODBC.
OracleConnectionStringBuilder	باني نص اتصال أوراكل.

فئة باني نص الاتصال
DbConnectionStringBuilder Class

هذه الفئة موجودة في النطاق System.Data.Common، وهي تمثل واجهة القاموس IDictionary، مما يعني أنها مجموعة Collection كل عنصر من عناصرها يكون في صورة مفتاح Key وقيمة Value.. وبهذا التصميم تستطيع تكوين نص الاتصال، بإضافة عناصر إلى هذا القاموس، كل عنصر منها يتكون من اسم خاصية الاتصال وقيمتها، حيث ستقوم الفئة DbConnectionStringBuilder بتكوين صيغة نص الاتصال بناء على هذه العناصر.

وتوجد هذه الفئة في نطاق الاسم System.Data.Common، لهذا لا تنسَ إضافة الجملة التالية أعلى صفحة الكود قبل استخدامها:

```
using System.Data.Common;
```

ولحدث الإنشاء Constructor الخاص بهذه الفئة صيغتان:

١- الأولى لا تستقبل أي معاملات.. مثال:

```
var CsB = new DbConnectionStringBuilder( );
```

٢- والثانية تستقبل معاملا منطقيا Boolean، إذا جعلت قيمته true فسيتم وضع القيم المكتوبة في نص الاتصال بين قوسين مضلعين { } لاستخدامه مع مزود ODBC.

وبالإضافة إلى خصائص القاموس الشهيرة، تمتلك هذه الفئة الخاصيتين التاليتين:

نص الاتصال ConnectionString:

تقرأ أو تغير نص الاتصال الذي تتعامل معه هذه الفئة.. وتستطيع الحصول على نص الاتصال أيضا باستخدام الوسيلة ToString.. لاحظ أن باني نص الاتصال يرتب المفاتيح في النص العائد حسب أولويتها، وليس على حسب ترتيب إضافتك لها.

نص اتصال قابل للتصفح BrowsableConnectionString:

إذا جعلت قيمة هذه الخاصية true، فسيتم عرض نص الاتصال في نافذة الخصائص عندما تستخدم الأداة PropertyGrid لعرض خصائص باني نص الاتصال.

وبالإضافة إلى وسائل القاموس الشهيرة، تمتلك هذه الفئة الوسائل التالية:

إضافة مفتاح وقيمة AppendKeyValuePair:

تتيح لك إضافة خاصية وقيمتها إلى نص اتصال موجود في باني نص StringBuilder، حيث ستقوم بتكوين الصيغة الصحيحة للخاصية والقيمة، ثم إضافتها في نهاية باني النص.

وتستقبل هذه الوسيلة ثلاثة معاملات: باني النص StringBuilder، ونصا يمثل اسم الخاصية، ونصا يمثل قيمتها.. مثال:

```
var SB = new System.Text.StringBuilder(
          "Data Source = .\\SQLEXPRESS;");
DbConnectionStringBuilder.AppendKeyValuePair(SB,
          "AttachDbFilename", "C:\\Books.mdf");
DbConnectionStringBuilder.AppendKeyValuePair(SB,
          "Integrated Security", "true");
MessageBox.Show(SB.ToString( ));
```

ستعرض الرسالة النص:

```
Data Source = .\SQLEXPRESS; AttachDbFilename
= C:\Books.mdf; Integrated Security=true
```

وتوجد صيغة أخرى لهذه الوسيلة، تزيد على الصيغة الأولى بمعامل رابع، إذا جعلت قيمته true، فسيتم وضع القيم بين قوسين متعرجين {} لاستخدام نص الاتصال مع مزود ODBC.

مساوٍ لـ EquivalentTo:

أرسل إلى هذه الوسيلة نسخة من الفئة DbConnectionStringBuilder لمقارنتها بالنسخة الحالية من الفئة DbConnectionStringBuilder.. وتتم المقارنة بالتأكد من أن كل مفتاح في القاموس الأول له ما يناظره في القاموس الثاني (بغض النظر عن الترتيب)، وأن القيمتين المحفوظتين في كليهما متساويتان.. لاحظ أن مقارنة المفاتيح لا تراعي حالة الأحرف، بينما مقارنة القيم تراعي حالة الأحرف.. وفي حالة نجاح المقارنة يعتبر نصا الاتصال الموجودين في القاموسين متساويين، وتعيد هذه الوسيلة true.. وستجد مثالا على هذه الوسيلة في الزر EquivalentTo في المشروع ConStrBuilder.

هل يحتوي على ShouldSerialize:

تعيد true إذا كان المفتاح الذي أرسلته إليها كمعامل موجودا ضمن نص الاتصال.. هذا معناه أن هذه الوسيلة مكافئة تماما للوسيلة ContainsKey.. وستجد مثالا على هذه الوسيلة في الزر ShouldSerialize في المشروع ConStrBuilder.

محاولة معرفة القيمة TryGetValue:

تحاول قراءة قيمة أحد المفاتيح الموجودة في نص الاتصال، فإن كان المفتاح موجودا أعادت true، وإن لم يكن موجودا أعادت false دون أن تسبب خطأ في البرنامج.. لهذا يعتبر استخدامها أفضل من استخدام المفهرس Indexer لقراءة قيمة المفتاح، فهو يسبب خطأ إن لم يكن المفتاح موجودا، مما يستلزم استخدام الوسيلة ContainsKey أولا على سبيل الاحتياط.. مثلا:

```
if (CsB.ContainsKey("AttachDbFilename"))
   MessageBox.Show(CsB["AttachDbFilename"].
            ToString( )); // C:\Books.mdf
```

وللوسيلة TryGetValue معاملان: الأول معامل نصي يستقبل اسم المفتاح، والثـــاني معامل إخراج out من النوع Object، يعيد إليك قيمة المفتاح إن وجد.. والكود التالي هو إعادة كتابة للمثال السابق باستخدام هذه الوسيلة:

```
object Value = null;
if (CsB.TryGetValue("AttachDbFilename", out Value))
   MessageBox.Show(CsB["AttachDbFilename"].
            ToString ( )); // C:\Books.mdf
```

فئة باني نص اتصال سيكيول
SqlConnectionStringBuilder Class

هذه الفئة ترث الفئة DbConnectionStringBuilder، ويمكنك استخدامها لبناء نص الاتصال بسيكويل سيرفر، فهي تمتلك المزيد من الخصائص التي تحمل أسماء المعلومات اللازمة للاتصال بسيكويل سيرفر، مما يجعل تكوين نص الاتصال في منتهى السهولة والوضوح.

ولحدث إنشاء هذه الفئة صيغتان:

١- الأولى بدون معاملات.

٢- والثانية تستقبل نص اتصال لإضافته إليها مبدئيا، حيث يمكنك إضافة أي تفاصيل أخرى إليه بعد ذلك.

وبجوار ما ترثه من الفئة الأم، تمتلك هذه الفئة الخصائص التالية:

توصيل ملف قاعدة البيانات AttachDBFilename:

تناظر المفتاح AttachDBFilename أو initial file name في نص الاتصال.. ويمكنك أن تضع في هذه الخاصية مسار واسم الملف الأساسي لقاعدة البيانات التي تريد الاتصال بها، وفي هذه الحالة سيتم توصيل هذه القاعدة بالخادم، والاتصال بها.

وعليك أن تتأكد أن ملف قاعدة البيانات ليس للقراءة فقط Read Only، لأن توصيل قاعدة البيانات يحتاج إلى إنشاء ملف سجل الأداة Log، واسمه يوضع في ملف قاعدة البيانات، ولو كانت للقراءة فقط فسيحدث خطأ ولن ينجح الاتصال.

أيضا، قد يحدث خطأ إذا كان ملف سجل الأداء Log موجودا في مجلد قاعدة البيانات وأنت تحاول توصيلها، مع وجود المفتاح Database في نص الاتصال.. في هذه الحالة عليك حذف ملف سجل الأداء وإعادة الاتصال، حيث سيتم إنشاء سجل أداء جديد لقاعدة البيانات.

الفهرس الأساسي InitialCatalog:

تقرأ أو تغير اسم قاعدة البيانات التي تريد الاتصال بها على الخادم.. وتختلف هذه الخاصية عن الخاصية السابقة في أنها تتعامل مع قاعدة بيانات متصلة بالخادم فعلا في هذه اللحظة، لهذا يتم ذكر اسم قاعدة البيانات فقط بدون المسار والامتداد (مثل Books).. بينما في الخاصية AttachDBFilename يتم ذكر مسار ملف قاعدة البيانات لتوصيلها بالخادم ثم الاتصال بها.

وتناظر هذه الخاصية المفتاح database أو Initial Catalog في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا.

مصدر البيانات DataSource:

تقرأ أو تغير عنوان خادم سيكويل.. قد يكون هذا العنوان للخادم المحلي SQLEXPRESS\.، أو لخادم بعيد Remote Server له عنوان بروتوكول الإنترنت IP Address الخاص به مثل (10.0.0.127).

وتناظر هذه الخاصية المفتاح Data Source أو server أو address أو addr أو network address في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا.

بديل فشل الاتصال FailoverPartner:

تقرأ أو تغير عنوان خادم سيكويل البديل، الذي سيتم استخدامه إذا فشل الاتصال بالخادم الرئيسي الموضح في الخاصية السابقة.

وتناظر هذه الخاصية المفتاح Failover Partner في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا.

حماية متكاملة IntegratedSecurity:

تناظر المفتاح trusted_connection أو Integrated Security في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false، مما يعني أن عليك إمداد الاتصال باسم المستخدم وكلمة المرور.. أما لو جعلت قيمتها true، فسيتم استخدام حساب المستخدم على الويندوز للاتصال.. هذا مفيد عند الاتصال بالخادم المحلي، أو عند استخدام البرنامج داخل شركة تستخدم شبكة داخلية LAN، ففي هذه الحالة يقوم مدير نظام سيكويل سيرفر System Administrator بتعريف حسابات الويندوز الخاصة بأجهزة المستخدمين المتصلة بالشبكة والمسموح لها بالاتصال بالخادم، وبهذا يكفي مجرد تسجيل الدخول على الويندوز لضمان سرية الاتصال بالخادم.

معرف المستخدم UserID:

تقرأ أو تغير اسم المستخدم الذي يتصل بالخادم، وذلك في حالة عدم استخدام الحماية المتكاملة Integrated Security.

وتناظر هذه الخاصية المفتاح User ID أو user أو uid في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا.

كلمة السر Password:

تقرأ أو تغير كلمة المرور اللازمة للاتصال بالخادم، وذلك في حالة عدم استخدام الحماية المتكاملة Integrated Security.

وتناظر هذه الخاصية المفتاح Password أو pwd في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا.

معرف الجهاز WorkstationID:

تقرأ أو تغير اسم الجهاز الذي يتصل بالخادم، وهي تناظر المفتاح Workstation ID أو wsid في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا.

إبقاء معلومات السرية PersistSecurityInfo:

تناظر المفتاح Persist Security Info أو persistsecurityinfo في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false، مما يعني أن عليك إرسال اسم المستخدم وكلمة السر كلما أردت فتح الاتصال.. أما لو جعلت قيمتها true، فيمكنك إرسال هذه المعلومات عند فتح الاتصال لأول مرة فقط، وسيتم الاحتفاظ بها لاستخدامها في فتح الاتصال بعد هذا.

نفاد وقت الاتصال ConnectTimeout:

تمثل وقت الانتظار الذي ستعتبر محاولة الاتصال بالخادم فاشلة بعد مروره دون استجابة من الخادم، وهي تناظر المفتاح Connect Timeout أو connection timeout أو timeout في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها ١٥ ثانية.

اسم التطبيق ApplicationName:

تناظر المفتاح app أو Application Name في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها .NET SqlClient Data Provider، لكن بإمكانك أن تضع فيها اسم برنامجك.

اللغة الحالية CurrentLanguage:

تقرأ أو تغير اسم سجل اللغة في سيكويل سيرفر، وهي تناظر المفتاح language أو Current Language في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا "".

التشفير Encrypt:

تناظر المفتاح Encrypt في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false مما يعني أن خادم سيكويل لن يشفر البيانات المرسلة بينه وبين العميل.. ولو جعلت قيمتها true، فسيتم استخدام نوع من التشفير يسمى SSL Encryption الذي يعني "تشفير طبقة مقابس الاتصال الآمنة" Secure Sockets Layer Encryption، وهو يستخدم مفتاحين: مفتاحا عاما Public Key يستخدم لتشفير البيانات، ومفتاحا خاصا Private Key يستخدم لفك تشفيرها.

إجازة خادم موثوق به TrustServerCertificate:

تناظر المفتاح TrustServerCertificate في نص الاتصال، وإذا جعلت قيمتها true، فسيتم تجاهل عملية إجازة الخادم Certification، اكتفاءً بحماية البيانات باستخدام تشفير SSL.

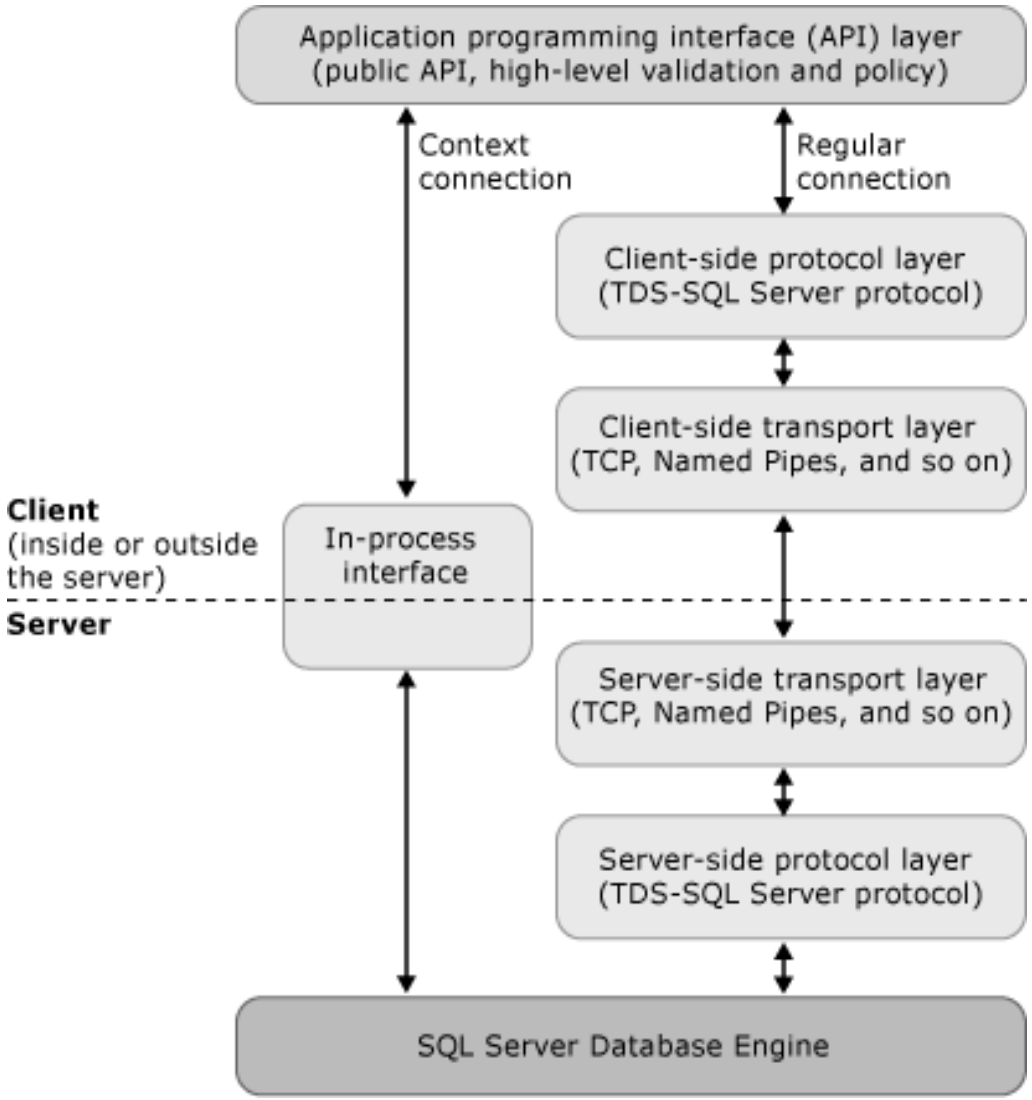
اتصال بالمحتوى ContextConnection:

تناظر المفتاح Context Connection في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false.

وينصح بجعل قيمة هذه الخاصية true عند الاتصال بخادم محلي Local Server توجد عليه الإجراءات المخزنة ودوال T-SQL التي تريد تنفيذها، لأن هذا يجعلك تستخدم نفس موارد الاتصال السابق بالخادم المحلي، مما يوفر عليك إعادة إدخال اسم المستخدم وكلمة السر، ويتيح لك التفاعل مع التعاملات Transactions التي لم يتم حفظها بعد، كما يتيح لك استخدام الجداول المؤقتة التي تم إنشاؤها على الاتصال المحلي.. وتؤدي هذه الطريقة إلى أداء أفضل للبرنامج، لأنها تتجاهل بروتوكولات الشبكة ومراحل نقل البيانات عبرها، وتتعامل مباشرة مع الخادم المحلي (لأنه يوجد على نفس الجهاز)، مما يجعل الاتصال أسرع وأكفأ.

وينصح بجعل قيمة هذه الخاصية false لاستخدام الاتصال العادي Regular Connection، وذلك عند الاتصال بخادم بعيد (غير محلي) Remote Server.

والشكل التالي يلخص الفارق بين هذين النوعين من الاتصال.. لاحظ أن الاتصال بالمحتوى يتجاهل العديد من طبقات الاتصال عبر الشبكة Network Layers، ويستخدم واجهة الاتصال المباشر In-Process Interface.



في القائمة Enlist:

تناظر المفتاح Enlist في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها true، مما يعني أن الاتصال الحالي سيوضع في قائمة الاتصالات المستخدمة لمحتوى التعاملات الحالي Current Transaction Context.. هذا يسمح بجعل أكثر من اتصال تتشارك في تعامل Transaction واحد، بحيث لو فشلت أي عملية على أي اتصال منها يتم إلغاء العمليات التي تمت على باقي الاتصالات.. لن نتعمق في موضوع التعاملات Transactions في هذا الكتاب، وسنتعرف عليه مع باقي المواضيع المتقدمة في برمجة قواعد البيانات في الكتاب القادم بإذن الله.

معالجة غير متزامنة AsynchronousProcessing:

تناظر المفتاح async أو Asynchronous Processing في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false، ولو جعلت قيمتها true فسيعني هذا السماح للخادم بإجراء عمليات معالجة غير متزامنة.. هذا معناه أن برنامجك سيواصل العمل مباشرة بعد إرسال الاستعلام إلى الخادم، تاركا الخادم يواصل تنفيذ الاستعلام.. هذا يوفر عليك كتابة الكثير من الكود لإنشاء عمليات فرعية Threads أو عمليات غير متزامنة في برنامجك لضمان مواصلة الاستجابة للمستخدم أثناء معالجة سيكويل سيرفر للاستعلامات السابقة.

مساهمة Pooling:

تناظر المفتاح Pooling في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها true، مما يعني أن هذا الاتصال سينضم إلى رصيد الاتصالات المساهمة Connection Pool التي تظل مفتوحة دائما لاستخدامه فور الحاجة إليها.. أما لو جعلت قيمتها false فسيعني هذا أن هذا الاتصال سيتم فتحه وإغلاقه مباشرة بعد انتهاء استخدامه، وعند الاحتياج إليه مجددا يتم فتحه من جديد.. وهكذا.

أقصى حجم للمساهمة MaxPoolSize:

تناظر المفتاح Max Pool Size في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها ١٠٠، مما يعني الاحتفاظ في رصيد الاتصالات المساهمة المتاحة للاستخدام، بمئة اتصال ــ كحد أقصى ــ مفتوحة بين الخادم والعميل.

أقل حجم للمساهمة MinPoolSize:

تقرأ أو تغير أصغر عدد من الاتصالات يجب أن يظل مفتوحا بين الخادم والعميل في رصيد الاتصالات المساهمة، وهي تناظر المفتاح Min Pool Size في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها صفرا.

وقت انتظار توازن الحمل LoadBalanceTimeout:

تقرأ أو تغير الوقت بالثانية، الذي يتم انتظاره أثناء وجود الاتصال في رصيد الاتصالات المساهمة Connection Pool، قبل أن يتم إغلاق الاتصال، وذلك لضمان عدم ترك الاتصالات المفتوحة خاملة بدون استخدام لفترات طويلة.. وفي الوضع الافتراضي تكون قيمتها صفرا، مما يعني ترك الاتصال مفتوحا دائما بدون قيود.. وتناظر هذه الخاصية المفتاح connection lifetime أو Load Balance Timeout في نص الاتصال.

إعادة الاتصال إلى وضعه الأصلي ConnectionReset:

تناظر المفتاح Connection Reset في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها true، وهذا معناه أن الاتصال سيعود إلى وضعه الأصلي عند طلب استخدامه من رصيد الاتصالات المساهمة Connection Pool.

مجموعات النتائج الفعالة المتعددةMultipleActiveResultSets:

تناظر المفتاح MultipleActiveResultSets في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false، وهذا معناه استخدام "مجموعة النتائج العادية" Default Result Set، وفيها يتم إرسال نتائج الاستعلام من خادم سيكويل إلى جهاز العميل، حيث يتم حفظها في مخزن وسيط Buffer في الذاكرة، وعندما يحتاج برنامجك إلى عرضها للمستخدم، يتم المرور عبرها سجلا بسجل.. ولا يستطيع العميل استخدام الاتصال المفتوح مع الخادم في تحديث البيانات قبل أن ينتهي من التعامل مع كل البيانات التي أرسلها الخادم أولا، أو قبل أن يرسل إلى الخادم طلبا لإلغاء إرسال باقي النتائج.. وتعتبر هذه الطريقة أكثر كفاءة في استغلال الاتصال، لأن الخادم يرسل أكبر كم ممكن من النتائج عبر حزم البيانات Packets المرسلة عبر الشبكة Network.

ولو جعلت قيمة هذه الخاصية true، فسيتم استخدام "مجموعات النتائج الفعالة المتعددة" Multiple Active Result Sets أو اختصارا MARS، وهي متاحة فقط مع سيكويل سيرفر ٢٠٠٥ وما يليه من إصدارات، وفيها يسمح للعميل باستخدام أكثر من قارئ بيانات SqlDataReader في نفس الوقت.

مكتبة الشبكة NetworkLibrary:

ضع في هذه الخاصية اسم مكتبة الربط DLL التي تريد من الخادم استخدامها للاتصال عبر الشبكة، وذلك بشرط توفر هذه المكتبة على الخادم.

وتناظر هذه الخاصية المفتاح Network Library أو network أو net في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها نصا فارغا.. والجدول التالي يوضح القيم المحتملة لهذه الخاصية:

اسم مكتبة الربط	نوع الاتصال
dbnmpntw	Named Pipes
dbmsrpcn	Multiprotocol

AppleTalk	dbmsadsn
VIA	dbmsgnet
Shared Memory	dbmslpcn
IPX/SPX	dbmsspxn
TCP/IP	dbmssocn

وفي حالة التعامل مع خادم محلي وترك قيمة هذه الخاصية فارغة، يتم استخدام المكتبة dbmslpcn (Shared Memory).

حجم حزمة البيانات PacketSize:

عند إرسال البيانات عبر الشبكة أو الإنترنت، يتم تقسيمها إلى حزم Packets.. وتحدد هذه الخاصية حجم كل حزمة من هذه الحزم بالوحدة الثنائية Byte، وفي الوضع الافتراضي تكون قيمتها ٨٠٠٠ وحدة Byte.

وتناظر هذه الخاصية المفتاح Packet Size في نص الاتصال.

النسخ المطابق Replication:

تناظر المفتاح Replication في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false، وإذا جعلتها true فسيتم تمكين عملية النسخ المطابق Replication عبر هذا الاتصال، وهي تقنية آلية تتيح لك نسخ قاعدة بيانات بين أكثر من خادم، وإبقاء البيانات متزامنة بين الخادمين، بحيث يتم تحديث إحدى قاعدتي البيانات إذا حدث تغيير في الأخرى.. هذا مفيد عندما يكون هناك خادم أصلي وخادم احتياطي للتعامل معه إذا حدثت مشكلة في الخادم الأصلي أو تم إيقافه للصيانة مثلا.

ربط التعاملات TransactionBinding:

تناظر المفتاح Transaction Binding في نص الاتصال.. ويمكنك أن تضع فيها إحدى القيمتين التاليتين:

Implicit Unbind	فك ارتباط ضمني: وهي القيمة الافتراضية، وفيها يؤدي إغلاق الاتصال إلى فصله عن التعاملات الجارية Current Transactions.
Explicit Unbind	فك ارتباط صريح: يجب عليك فك الارتباط بين الاتصال والتعاملات الجارية بطريقة صريحة قبل إغلاق الاتصال، وإلا حدث خطأ.

إصدار نظام الأنواع TypeSystemVersion:

تناظر المفتاح Type System Version في نص الاتصال، وهي تتيح لك تحديد إصدار سيكويل سيرفر الذي تريد أن تستخدم أنواع البيانات الخاصة به.. على سبيل المثال، لو كان الخادم يستخدم سيكويل سيرفر ٢٠٠٨، وجعلت قيمة هذه الخاصيةSQL Server 2000، فيمكنك استخدام أنواع البيانات Data Types الخاصة بسيكويل سيرفر ٢٠٠٠، حيث سيقوم سيكويل سيرفر ٢٠٠٨ بإجراء التحويلات المناسبة للتعامل معها.. والجدول التالي يلخص لك القيم الممكنة لهذه الخاصية:

SQL Server 2000	يتم استخدام الأنواع الخاصية بسيكويل سيرفر ٢٠٠٠.. هذا يعني إجراء بعض التحويلات عند التعامل مع إصدارات أحدث، مثل: - تحويل XML إلى NTEXT. - تحويل UDT إلى VARBINARY. - تحويل VARCHAR(MAX) إلى TEXT. - تحويل NVARCHAR(MAX) إلى NEXT. - تحويل ARBINARY(MAX) إلى IMAGE.
SQL Server 2005	يتم استخدام الأنواع الخاصية بسيكويل سيرفر ٢٠٠٥.

SQL Server 2008	يتم استخدام الأنواع الخاصية بسيكويل سيرفر ٢٠٠٨.
Latest	يتم استخدام أحدث إصدار من سيكويل سيرفر يمكن للخادم والعميل التعامل معه.

نسخة المستخدم UserInstance:

تناظر المفتاح User Instance في نص الاتصال.. وفي الوضع الافتراضي تكون قيمتها false، ولو جعلتها true فسيتم توجيه الاتصال من نسخة خادم سيكويل الافتراضية، إلى نسخة أخرى مخصصة للعميل، لكن هذا قد يسبب أخطاء في الاتصال إذا كنت تستخدم السمة FILESTREAM لحفظ بيانات بعض الأعمدة في ملفات خارجية.

ولا تمتلك هذه الفئة أية وسائل Methods غير ما ترثه من الفئة الأم.

والمثال التالي يريك كيف تستخدم هذه الفئة لتكوين نص اتصال بقاعدة الكتب على الخادم المحلي باستخدام الحماية المتكاملة:

```
var CnStrBldr = new SqlConnectionStringBuilder( );
CnStrBldr.DataSource = ".\\SQLEXPRESS";
CnStrBldr.InitialCatalog = "Books";
CnStrBldr.IntegratedSecurity = true;
var CnStr = CnStrBldr.ConnectionString;
MessageBox.Show(CnStr);
```

حفظ نص الاتصال في إعدادات البرنامج Settings:

عند كتابة برنامج يتعامل مع قواعد البرنامج، يلجأ المبرمج في معظم الأحول إلى إنشاء قاعدة البيانات على جهازه، أو ينسخ جزءا من قاعدة البيانات من الخادم إلى جهازه، ليجعلها تعمل على الخادم المحلي، ومن ثم يتصل بها من برنامجه.. هذا يجعل كتابة واختبار الكود وتصحيحه أسرع من التعامل مع خادم حقيقي عبر شبكة الإنترنت، كما أنه يضمن عدم تخريب قاعدة البيانات الرئيسية عند إضافة أو حذف السجلات للاختبار.

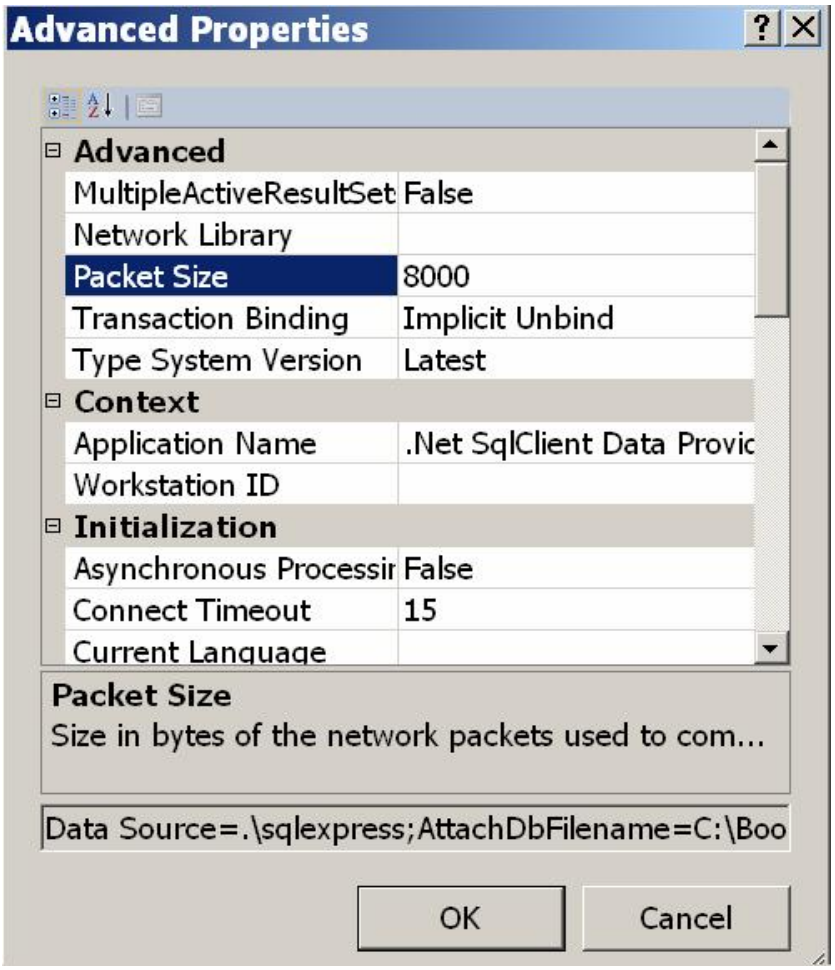
وبعد الانتهاء من البرنامج، يتم رفع قاعدة البيانات إلى الخادم (إن لم تكن موجودة عليه)، وتصحيح نصوص الاتصال لتشير إلى الخادم الحقيقي بدلا من الخادم المحلي، لكي يبدأ البرنامج عمله في صورته النهائية.

ونظرا لأن البرامج العملية قد تتعامل مع أكثر من قاعدة بيانات، كما أن عنوان قاعدة البيانات قد يتغير في أي لحظة لو تم نقلها من خادم إلى آخر على الإنترنت، يصير من غير العملي كتابة نص الاتصال في الكود، لأن البحث عنه وتغييره في كل المواضع أمر مرهق وعرضة للخطأ.. لهذا يفضل كتابة نصوص الاتصال في ملف خارج البرنامج، مع استخدام إحدى طرق التشفير لحماية التفاصيل المكتوبة به (كاسم المستخدم وكلمة السر).. ويمكن فعل هذا يدويا، أو باستخدام إحدى الطرق الجاهزة التي تمنحها دوت نت، كالإعدادات Settings التي تعرفنا عليها بالتفصيل الممل في مرجع "برمجة نماذج الويندوز"، وقلنا هناك إن دوت نت تقدم رعاية خاصة لنصوص الاتصال، فقد خصصت لها مقطعا خاصا في ملف الإعدادات، ومنحتنا فئة تتعامل معه، هي الفئة ConnectionStringsSection.. ولقد أرجأنا شرح هذا الموضوع إلى حين التعرف على قواعد، وها نحن أولاء ☺ .

لإضافة نص اتصال إلى الإعدادات بطريقة مرئية اتبع الخطوات التالية:

- افتح متصفح المشاريع Solution Explorer، وانقر مرتين بالفأرة فوق العنصر Properties.. سيؤدي هذا إلى فتح نافذة خصائص المشروع.
- اضغط العنصر Settings من الهامش الأيسر لفتح صفحة مصمم الإعدادات كما تعلمنا من قبل.

- في العمود Name اكتب اسم خاصية الإعداد، ولتكن BooksConStr.
- في العمود Type اضغط زر الإسدال، ومن القائمة المنسدلة اختر العنصر الخاص (Connection String).. هذا سيغير نطاق خاصية الإعداد Scope ليصير على مستوى التطبيق Application.
- اضغط الزر الموجود في خانة القيمة Value.. سيعرض لك هذا مربع حوار خصائص الاتصال الذي استخدمناه من قبل لإنشاء اتصال من متصفح الخوادم Server Explorer.. هذا يتيح لك تكوين نص الاتصال بطريقة مرئية سهلة.. حدد مزود البيانات وقاعدة البيانات واختيارات الحماية.. ولو أردت استخدام المزيد من مفاتيح نص الاتصال، فاضغط الزر Advanced.. سيعرض لك هذا النافذة التالية:



هذه النافذة تعرض خصائص نص الاتصال، وهي نفس الخصائص التي شرحناها في الفئة SqlConnectionStringBuilder.. ويمكنك تغيير القيم الافتراضية لهذه الخصائص، وكل خاصية ستغيرها ستظهر في نص الاتصال الذي سيتم تكوينه.

- اضغط OK لإغلاق النافذتين.. سيظهر نص الاتصال الذي تم تكوينه في الخانة Value.
- وإذا كنت تحتاج إلى هذا، يمكنك إضافة نصوص اتصال أخرى إلى الإعدادات، بالكتابة في الصفوف التالية، وبهذا تجمع في مكان واحد، كل نصوص الاتصال اللازمة للتعامل مع كل قواعد البيانات والخوادم التي تحتاجها في برنامجك، مما يسهل عليك تعديلها في أي لحظة.
- اضغط زر الحفظ من شريط الأدوات لحفظ هذه التغييرات في ملف إعدادات المشروع.

الآن، تم توليد خاصية اسمها BooksConStr في فئة الإعدادات Settings في النطاق Properties، وتستطيع قراءة قيمتها في أي وقت بمنتهى البساطة.. مثلا:

MessageBox.Show(Properties.Settings.Default.BooksConStr);

لاحظ أنك لا تستطيع تغيير قيمة الخاصية BooksConStr لأنها معرفة للقراءة فقط، لكن ما زال بوسعك فتح مصمم الإعدادات في أي لحظة لتغيير قيمة نص الاتصال، أو فتح الملف app.config من متصفح المشاريع، وتحرير قيمة الخاصية BooksConStr التي ستجدها تحت المقطع <connectionStrings>، دون الحاجة إلى تغيير أي كود في برنامجك.

فئة مقطع نصوص الاتصال
ConnectionStringsSection Class

هذه الفئة موجودة في النطاق System.Configuration، وهي ترث الفئة ConfigurationSection التي تعرفنا عليها في كتاب برمجة الويندوز.
ولا جديد في هذه الفئة، سوى امتلاكها للخاصية التالية:

نصوص الاتصال ConnectionStrings:

تعيد مجموعة من النوع ConnectionStringSettingsCollection، التي ترث الفئة ConfigurationElementCollection، وكل عنصر من عناصر هذه المجموعة هو من نوع فئة إعدادات نص الاتصال ConnectionStringSettings Class، التي سنتعرف عليها بعد قليل.
وتتيح لك هذه المجموعة قراءة كل نصوص الاتصال الموجودة في ملف الإعدادات.

ويمكنك استخدام إحدى وسائل فتح التهيئة OpenxxConfiguration الخاصة بمدير التهيئة ConfigurationManager للحصول على كائن تهيئة Configuration Object يتعامل مع النوع المراد من الإعدادات، ثم استخدام الخاصية ConnectionStrings لكائن التهيئة للحصول على نسخة من الفئة ConnectionStringsSection كالتالي:

```
var Cnfg = ConfigurationManager.
                        OpenMachineConfiguration( );
var CnStrSett = Cnfg.ConnectionStrings;
```

فئة إعدادات نص الاتصال
ConnectionStringSettings Class

هذه الفئة ترث فئة عنصر التهيئة ConfigurationElement Class، التي تعرفنا عليها في كتاب برمجة نماذج الويندوز.

ولحدث إنشاء هذه الفئة ثلاث صيغ:

١- الأولى بدون معاملات.

٢- والثانية تستقبل معاملين: اسم خاصية الإعداد التي ستحفظ نص الاتصال، ونص الاتصال نفسه.

٣- والثالثة تزيد على الصيغة السابقة بمعامل ثالث، يستقبل اسم مزود البيانات Provider الذي سيستخدم نص الاتصال.

وإضافة إلى ما ترثه من الفئة الأم، تمتلك هذه الفئة الخصائص التالية:

الاسم Name:

تقرأ أو تغير اسم خاصية الإعداد التي ستحفظ نص الاتصال.

نص الاتصال ConnectionString:

تقرأ أو تغير نص الاتصال المحفوظ في خاصية الإعداد.

اسم المزود ProviderName:

تقرأ أو تغير اسم مزود البيانات الذي سيستخدم نص الاتصال.

ويمكنك الحصول على مجموعة إعدادات نصوص الاتصال الخاصة بالتطبيق، باستخدام الخاصية المشتركة Static Property التالية:

```
var CnStrSett = ConfigurationManager.ConnectionStrings;
```

والمثال التالي يعرض لك كل نصوص الاتصال الموجودة في ملف إعداد التطبيق:

```
var CnStrSett = ConfigurationManager.ConnectionStrings;
foreach (ConnectionStringSettings CnStr in CnStrSett)
{
   MessageBox.Show(CnStr.Name);
   MessageBox.Show(CnStr.ProviderName);
   MessageBox.Show(CnStr.ConnectionString);
}
```

ملحوظة:
يجب عليك حماية نص الاتصال بتشفيره، وذلك لأن ملف الإعدادات يتم توزيعه مع البرنامج، مما يجعل المستخدمين قادرين على قراءته وأخذ كلمات المرور منه. ويمكنك تشفير مقطع نصوص الاتصال <ConnectionStrings> في ملف الإعدادات، بنفس الطريقة التي شرحناها في كتاب برمجة الويندوز، واستخدمناها في البرنامج AddAppSettings المرفق بذلك الكتاب.

واجهة الاتصال بقواعد البيانات
IDbConnection Interface

هذه الواجهة تمثل الواجهة IDisposable، وهي تتيح لك إنشاء كائن اتصال خاص بك، وذلك بكتابة فئة تمثلها Implements the interface.. هذا يسهل عليك كتابة مزود جديد للتعامل مع نوع معين من قواعد البيانات غير متاح في إطار العمل.
وتمتلك الواجهة IDbConnection الخصائص التالية:

نص الاتصال ConnectionString:

تقرأ أو تغير نص الاتصال الذي يحتوي على المعلومات اللازمة للاتصال بقاعدة البيانات.. ولا يمكنك تغيير قيمة هذه الخاصية إلا عندما يكون الاتصال مع قاعدة البيانات مغلقا.

وقت الانتظار ConnectionTimeout:

تقبل عددا صحيحا، يمثل الوقت بالثانية، الذي سيتم انتظاره أثناء محاولة الاتصال بقاعدة البيانات، فإذا مر هذا الوقت دون أن يستجيب الخادم، يتم إلغاء العملية وينطلق خطأ في برنامجك.. والقيمة الافتراضية لهذه الخاصية هي ١٥ ثانية، لكن إذا أردت أن تظل منتظرا إتمام الاتصال إلى ما لانهاية، فضع صفرا في هذه الخاصية!.. لكن هذا قد يؤدي إلى توقف برنامجك عن العمل إذا فشلت عملية الاتصال بالخادم، لهذا لو استخدمت هذه القيمة فيجب أن تعطي للمستخدم طريقة لإلغاء محاولة الاتصال بنفسه، كأن تضع على النموذج زر إلغاء، مع جعل عملية الاتصال في عملية فرعية مستقلة Thread لكي لا يتوقف البرنامج عن الاستجابة.
لاحظ أن وضع قيمة كبيرة في هذه الخاصية سيؤدي إلى تعطيل البرنامج لفترة أطول، ووضع قيمة صغيرة فيها سيؤدي إلى فشل محاولات الاتصال بسرعة.. وأفضل قيمة لهذه الخاصية هي ما تراه مناسبا لظروف برنامجك.. فلو كنت تتوقع ضغطا كبيرا

على الخادم يجعل استجابته لمحاولات الاتصال بطيئة أو متأخرة، فضع قيمة أكبر في هذه الخاصية (مثل ٦٠ أو ٩٠ مثلا).

قاعدة البيانات Database:

تعيد اسم قاعدة البيانات التي يتم الاتصال بها.

الحالة State:

تعيد إحدى قيم المرقم ConnectionState التي تعبر عن حالة الاتصال في هذه اللحظة، وهي:

Closed	تم إغلاق الاتصال.
Open	الاتصال مفتوح.
Connecting	يتم إجراء الاتصال.
Executing	يتم تنفيذ أحد الأوامر على قاعدة البيانات عبر الاتصال الحالي.
Fetching	يتم إحضار بيانات من قاعدة البيانات عبر الاتصال الحالي.
Broken	تم فتح الاتصال، لكن حدث عطل أدى إلى إغلاقه.. ويمكنك إعادة محاولة فتح هذا الاتصال.

كما تمتلك هذه الواجهة الوسائل التالية:

فتح Open:

تفتح الاتصال بقاعدة البيانات، تبعا للبيانات الموجودة في الخاصية ConnectionString.

تغيير قاعدة البيانات ChangeDatabase:

أرسل إلى هذه الوسيلة معاملا نصيا، يمثل اسم قاعدة بيانات جديدة موجودة على نفس الخادم، للتعامل معها بدلا من قاعدة البيانات الحالية الموضحة في الخاصية

Database.. لاحظ أن استخدام هذه الوسيلة متاح فقط أثناء فتح الاتصال بالخادم، وإلا حدث خطأ يخبرك أن الاتصال مغلق!.. الحكمة من هذا، هو استغلال نفس الاتصال المفتوح مع الخادم للتعامل مع أكثر من قاعدة بيانات، لتوفير وقت إغلاق الاتصال وإعادة فتح اتصال جديد.

إنشاء أمر CreateCommand:

تنشئ هذه الوسيلة كائن أمر Command Object جديد لتنفيذه عبر كائن الاتصال الحالي.. ولا يشترط أن يكون الاتصال مفتوحا عند استدعاء هذه الوسيلة، فكل ما تفعله هو إنشاء كائن أمر مناسب، ووضع مرجع لكائن الاتصال الحالي في الخاصية Connection الخاصة بكائن الأمر.. لكن عند تنفيذ الأمر يجب أن يكون الاتصال مفتوحا فعلا، وإلا حدث خطأ.

والقيمة العائدة من هذه الوسيلة من نوع واجهة "أمر قاعدة البيانات" IDbCommand التي سنتعرف عليها بالتفصيل لاحقا.

بدء التعاملات BeginTransaction:

تنشئ هذه الوسيلة كائن تعاملات Transaction Object لتستطيع من خلاله إجراء عدد من العمليات على قاعدة البيانات، مع قدرتك على التراجع عنها بعد ذلك.. والقيمة العائدة من هذه الوسيلة من نوع واجهة "تعاملات قاعدة البيانات" IDbTransaction التي سنتعرف عليها بالتفصيل لاحقا.

وتوجد صيغة ثانية لهذه الوسيلة، لها معامل واحد يستقبل إحدى قيم المرقم "مستوى العزل" IsolationLevel.. وسنتعرف على هذا المرقم بالتفصيل لاحقا.

إغلاق Close:

تقوم بالتراجع Rollback عن أي تعاملات Transactions لم يتم إحالتها إلى قاعدة البيانات Committed.. ثم تغلق الاتصال.

لاحظ أنه في حالة تفعيل خاصية المساهمة Pooling، فإن خادم سيكويل يحافظ على عدد محدد من الاتصالات المفتوحة بينه وبين برنامجك، وذلك لتوفير وقت إغلاق وإعادة فتح الاتصالات بينهما.. وفي هذه الحالة لا تقوم الوسيلة Close بإغلاق الاتصال، بل تترك الاتصال مفتوحا، وتضيفه إلى رصيد الاتصالات المساهمة Connection Pool، ليمكن استخدامه مباشرة عند الاحتياج إليه.

فئة الاتصال DbConnection Class

هذه الفئة أساسية مجردة Abstract Base Class، وهي تمثل الواجهة IDbConnection، كما أنها ترث فئة المكون Component Class، لكنك لن تستطيع إضافتها إلى صينية مكونات النموذج Component Tray لأنك لا تستطيع إنشاء نسخة جديدة منها، لكن الفئات المشتقة منها مثل SQLConnection يمكن إضافتها إلى صينية المكونات.. لفعل هذا افتح صندوق الأدوات Toolbox، وأسدل الشريط Data، واضغطه بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط Choose Items، وفي النافذة التي ستظهر، ضع علامة الاختيار بجوار مجموعة الأدوات التي تبدأ بالحروف SQL ومن ضمنها SQLConnection، ثم اضغط الزر OK.. الآن ستجد هذه الأدوات تحت الشريط Data في صندوق الأدوات.. انقر الأداة SQLConnection مرتين بالفأرة لإضافة نسخة منها إلى صينية المكونات.

وبالإضافة إلى ما تمثله من خصائص الواجهة IDbConnection، تملك هذه الفئة الخاصيتين التاليتين:

مصدر البيانات DataSource:

تعيد اسم خادم سيكويل الذي سيتم الاتصال به.

إصدار الخادم ServerVersion:

تعيد نصا يمثل إصدار سيكويل سيرفر الذي يتصل به العميل.. ويجب أن يكون الاتصال مفتوحا في تلك اللحظة وإلا حدث خطأ.

وبالإضافة إلى ما تمثله من وسائل الواجهة IDbConnection، تمتلك هذه الفئة الوسيلتين التاليتين:

إضافة إلى قائمة التعاملات EnlistTransaction:

أرسل إلى هذه الوسيلة كائن التعاملات Transaction Object الذي تريد ضم تعاملات الاتصال الحالي إليه، لتكوين تعاملات منتشرة Distributed Transaction، وهي تعاملات تنفذ عمليات على أكثر من مصدر وأكثر من اتصال، ولا ينجح تنفيذها إلا إذا نجحت كل أجزائها.. وسنتعرف على كائن التعاملات لاحقا.

معرفة المخطط GetSchema:

تعيد كائن جدول DataTable Object، يحتوي على بيانات المخطط الخاص بالخادم.

ولهذه الوسيلة صيغة ثانية، تستقبل معاملا نصيا يمثل اسم المخطط الذي تريد استعادته.

كما توجد صيغة ثالثة، تزيد على الصيغة السابقة بمعامل ثان، يستقبل مصفوفة نصية String Array، تمثل القيود Restrictions التي تريد الحصول على مخططها.

كما تمتلك هذه الفئة الحدث التالي:

تغير الحالة StateChange:

ينطلق عند تغير حالة الاتصال (عند إغلاقه أو فتحه).. والمعامل الثاني e لهذا الحدث من النوع StateChangeEventArgs، وهو يمتلك الخاصيتين التاليتين:

	OriginalState	تعيد إحدى قيم المرقم ConnectionState التي تمثل حالة الاتصال قبل حدوث التغيير.
	CurrentState	تعيد إحدى قيم المرقم ConnectionState التي تمثل حالة الاتصال الحالية (بعد حدوث التغيير).

والفئات التالية ترث الفئة DbConnection:

١. SqlConnection.

٢. OdbcConnection.

٣. OleDbConnection.

٤. OracleConnection.

مما يعني أنها جميعا تمتلك خصائص ووسائل هذه الفئة.. وسنتعرف الآن على واحدة من هذه الفئات، وهي الفئة SqlConnection.

فئة اتصال سيكيول SqlConnection Class

هذه الفئة ترث الفئة DbConnection، مما يعني أنها ضمنيا ترث الفئة Component وتمثل الواجهة IDbConnection.
وبالإضافة إلى الخصائص التي ترثها من الفئة الأم، تمتلك هذه الفئة الخصائص التالية:

إطلاق حدث الخطأ FireInfoMessageEventOnUserErrors:

إذا جعلت قيمة هذه الخاصية true، فسيتم إطلاق الحدث InfoMessage فور حدوث خطأ في الاتصال، ودون انتظار انتهاء تنفيذ الإجراء الذي أنشأ الاتصال.. أما إذا تركت قيمتها الافتراضية false، فسينطلق استثناء Exception في البرنامج عند حدوث خطأ في الاتصال، ولم ينطلق الحدث InfoMessage إلا بعد انتهاء تنفيذ الإجراء الذي أنشأ الاتصال.

حجم حزم البيانات PacketSize:

تعيد حجم حزم البيانات (بالوحدة الثنائية Byte) المستخدمة في نقل البيانات.

تفعيل الإحصائيات StatisticsEnabled:

إذا جعلت قيمة هذه الخاصية true، فسيتم جمع إحصائيات عن عملية الاتصال.. لاحظ أن هذا مفيد في بعض الحالات، لكنه قد يؤدي إلى إبطاء الاتصال، لهذا لا تستخدمه إلا للضرورة.. والقيمة الافتراضية لهذه الخاصية هي false.

معرف الجهاز WorkstationId:

تعيد اسم جهاز العميل المتصل بالخادم.

وبالإضافة إلى ما تمثله من وسائل الواجهة IDbConnection، تمتلك هذه الفئة الوسائل التالية:

تغيير كلمة السر ChangePassword:

أرسل إلى هذه الوسيلة نص الاتصال، وكلمة السر الجديدة التي تريد استخدامها مع المستخدم المحدد في نص الاتصال بدلا من كلمة السر القديمة.. هذا معناه أن نص الاتصال يجب أن يحتوي على اسم المستخدم UserID وكلمة السر القديمة Password، لهذا لو أرسلت نص اتصال فيه خيار الحماية المتكاملة IntegratedSecurity فسيحدث خطأ.

وتقوم هذه الوسيلة بفتح اتصال خاص بها لتغيير كلمة السر، وإغلاقه فور الانتهاء من هذا، دون التعامل مع رصيد الاتصالات المساهمة Connection Pool.

وتفيدك هذه الوسيلة إذا كانت كلمة السر الخاصة بالمستخدم قد انتهت صلاحيتها Expired ويجب تغييرها.. ويمكنك معرفة هذا عند استخدام الوسيلة Open لفتح الاتصال، حيث سيحدث خطأ في البرنامج من النوع SqlException، وعليك أن تفحص قيمة الخاصية Number الخاصة بهذا الاستثناء، فإن وجدت قيمتها ١٨٤٨٧ أو ١٨٤٨٨ فهذا معناه انتهاء صلاحية كلمة السر ووجوب تغييرها.

والمثال التالي يحاول الاتصال بالخادم، فإن فشل الاتصال بسبب انتهاء صلاحية كلمة السر، فإنه يغير كلمة السر القديمة:

```
var Csb = new SqlConnectionStringBuilder( );
Csb.DataSource = ".\\SQLEXPRESS";
Csb.InitialCatalog = "Books";
Csb.UserID = "User1";
Csb.Password = "2009";
SqlConnection Cn = new SqlConnection(Csb.ToString( ));
try {
   Cn.Open( );
}
catch (SqlException ex) {
   if (ex.Number == 18487 || ex.Number == 18488)
      SqlConnection.ChangePassword(Csb.ToString( ),
          "2010");
}
```

إلغاء المساهمة ClearPool:

أرسل إلى هذه الوسيلة كائن الاتصال SqlConnection Object لإلغاء الاتصال الذي يمثله من رصيد الاتصالات المساهمة Connection Pool.. لاحظ أن هذا الاتصال قد يكون مستخدما في تلك اللحظة لأداء بعض الاستعلامات، لهذا يتم إنهاؤه في الحال، وسيظل مستخدما إلى حين إغلاقه باستخدام الوسيلة Close، وعندها لن يعود إلى رصيد الاتصالات المساهمة، بل سيغلق في الحال.. مثال:

```
SqlConnection.ClearPool(Cn);
```

إلغاء كل أرصدة المساهمة ClearAllPools:

تغلق جميع الاتصالات الموجودة في رصيد الاتصالات المساهمة Connection Pool، وإذا كان بعضها مستخدما، لا يتم إنهاؤه إلى أن يتم استدعاء الوسيلة Close الخاصة به.. مثال:

```
SqlConnection.ClearAllPools( );
```

إضافة إلى قائمة التعاملات المنتشرة EnlistDistributedTransaction:

مماثلة للوسيلة EnlistTransaction.

الحصول على الإحصائيات RetrieveStatistics:

تعيد مجموعة تمثل واجهة القاموس IDictionary، تحتوي على أزواج من المفاتيح Keys والقيم Values، تمثل إحصائيات الاتصال حتى هذه اللحظة.. ويمكنك استدعاء هذه الوسيلة أكثر من مرة على فترات، للحصول على أحدث قيم للإحصائيات.. لاحظ أنك لن تحصل على أي إحصائيات إلا إذا جعلت للخاصية StatisticsEnabled القيمة true أولا.

تصفير الإحصائيات ResetStatistics:

تعيد جميع قيم الإحصائيات إلى الصفر.

كما تمتلك هذه الفئة الحدث التالي:

رسالة المعلومات InfoMessage:

ينطلق عندما يرسل الخادم رسالة تحذير أو خطأ.. والمعامل الثاني e لهذا الحدث من النوع SqlInfoMessageEventArgs، وهو يمتلك الخصائص التالية:

	Errors	تعيد مجموعة من النوع SqlErrorCollection، التي تمثل الواجهة ICollection، وكل عنصر من عناصرها من نوع الفئة SqlError، التي تحتوي على أحد الأخطاء أو التحذيرات التي أرسلها خادم سيكويل.. وسنتعرف على الفئة SqlError بعد قليل.
	Message	تعيد نصا يشرح أول خطأ موجود في مجموعة الأخطاء Errors.. هذا مفيد إن كان هناك خطا واحد فقط.
	Source	تعيد نصا يحدد اسم الكائن الذي تسبب في أول خطأ موجود في مجموعة الأخطاء Errors.. هذا مفيد إن كان هناك خطا واحد فقط.

وقد استخدمنا فئة اتصال سيكويل في التطبيق AuthorBooks_Reader للاتصال بقاعدة بيانات الكتب على الخادم المحلي.. لاحظ أننا فتحنا الاتصال في حدث تحميل النموذج Load ولم نغلقه إلا في حدث إغلاق النموذج FormClosing، مما أتاح لنا استخدام نفس الاتصال لتنفيذ جميع الاستعلامات التي يقوم بها المستخدم.. ورغم أن كائن الاتصال معرف كمتغير موضعي Local Variable في حدث تحميل النموذج، إلا أننا وضعنا مرجعا له في الخاصية Connection الخاصة بكائن الأمر Command Object المعرف على مستوى النموذج، مما جعل كائن الاتصال حيا طالما كان كائن الأمر حيا.. هذا هو السبب

في أننا استخدمنا الخاصية Connection الخاصة بكائن الأمر لإغلاق الاتصال في حدث إغلاق النموذج كالتالي:

Cmd.Connection.Close( );

لاحظ أن ترك كائن الاتصال مفتوحا طوال الوقت عمليّ فقط في حالتنا هذه، لأننا نتعامل هنا مع خادم محلي، وهناك نسخة واحدة فقط من البرنامج تتعامل معه.. لكن في البرامج العملية التي تتعامل مع خادم حقيقي، قد يؤدي ترك الاتصال مفتوحا إلى تقليل كفاءة البرنامج، خاصة إذا كان هناك عدد كبير من المستخدمين يتعاملون مع برنامجك في نفس اللحظة.

افترض مثلا أنك تصمم برنامجا لشركة يعمل بها ٢٥٠ موظفا، وأن خادم سيكويل مجهز لاستقبال ١٠٠ اتصال فقط في نفس اللحظة.. في هذه الحالة لو جعلت برنامجك يفتح نفس الاتصال بصورة دائمة طوال تشغيل المستخدم له، فإن أول ١٠٠ موظف يفتحون البرنامج على أجهزتهم سيمنعون خادم سيكويل من الاستجابة لمئة وخمسين موظفا آخرين إلى أن يغلق بعض المستخدمين البرنامج!

لقد أحببت أن أريك كيف يمكن أن يؤدي التصميم الخاطئ لبرنامجك إلى نتائج كارثية، ويعطل العمل ولا تجني من ورائه سوى السخط ☺ !

ورغم أن هذا مثال افتراضي لتقريب الفكرة، حيث إن سيكيول سيرفر يستطيع فعليا خدمة بضع مئات من العملاء وربما أكثر في نفس الوقت، إلا أن هذا العدد مهما بدا كبيرا لك فهو محدود، ويمكن تجاوزه عمليا في المؤسسات الضخمة كالحكومة الالكترونية مثلا (هل تتخيل كم عدد الموظفين في الوزارات المختلفة الذين يتعاملون مع قاعدة البيانات في نفس اللحظة؟)، أو في مواقع الإنترنت التي تحفظ قواعد بياناتها على سيكويل سيرفر، وأنت تعرف أن بعض هذه المواقع يصل زوارها إلى عدة ملايين يوميا، مما يؤدي إلى بطء استجابة الخادم، واعتذاره للكثير من العملاء عن وجود ضغط كبير بجبرهم على الانتظار (لا ريب أنك واجهت هذه المشكلة مع خادم بريد هوتميل في بعض الأوقات)!

ولحل هذه المشكلة في برنامجنا، عليك أن تنقل الكود من حدث تحميل النموذج وحدث إغلاقه إلى حدث ضغط الزر، ليتم فتح الاتصال وإغلاقه فقط عند الحاجة.. ولا تقلق من

كثرة فتح برنامجك للاتصال وإغلاقه، فخاصية مساهمة الاتصالات Connection Pooling التي يدعمها خادم سيكويل تغنيك عن أي عناء لحل هذه المشكلة، حيث يتم ترك بعض الاتصالات مفتوحة لضمان سرعة الاستجابة للاستعلامات المتكررة، دون إعاقة بعض المستخدمين عن الاتصال بالخادم.. ولحسن الحظ فإن تقنية المساهمة Pooling تكون فعالة في الوضع الافتراضي، ما لم تطلب أنت إيقافها صراحة عبر نص الاتصال كما عرفنا سابقا.

وستجد كود الاتصال المحسّن في المشروع المسمى AuthorBooks_Reader2.

فئة خطأ سيكيول SqlError Class

تحتوي هذه الفئة على معلومات عن رسالة الخطأ (أو التحذير) التي أرسلها خادم سيكويل.. وتمتلك هذه الفئة الخصائص التالية:

الرتبة Class:

تعيد رقما من ٠ إلى ٢٥٥ يمثل درجة خطورة الخطأ.. وقيمتها الافتراضية ٠.

الخادم Server:

تعيد نصا يمثل اسم نسخة سيكويل سيرفر التي أرسلت الخطأ.

المصدر Source:

تعيد اسم المزود Provider الذي تسبب في الخطأ.

الإجراء Procedure:

تعيد اسم الإجراء المخزن الذي تسبب في الخطأ.

رقم السطر LineNumber:

تعيد رقم السطر الذي تسبب في الخطأ في الإجراء المخزن.

الرسالة Message:

تعيد نصا يصف رسالة الخطأ.. .. ويمكنك أيضا استخدام الوسيلة ToString الخاصة بهذا الكائن لعرض نص هذه الرسالة.

الرقم Number:

تعيد رقم الخطأ.

الحالة State:

تعيد رقما من ٠ إلى ٢٥٥ يمثل الكود الرقمي للخطأ.

ملحوظة:
بالإضافة إلى ما ترثه فئة استثناء سيكويل SqlException من خصائص ووسائل من الفئة الأم SystemException والفئة الأم Exception، فإنها تمتلك نفس خصائص الفئة SqlError، مما يمنحك القدرة على الحصول على نفس المعلومات، سواء استخدمت الحدث InfoMessage أم استخدمت الطريقة التقليدية لمعالجة الأخطاء Exception Handling.

-٧-

كـــائـــــــن الأمـــــــــــر

Command Object

يتعامل كائن الأمر مع استعلام SQL أو إجراء مخزن Stored Procedure، مع امتلاكـــه الوسائل اللازمة لتنفيذهما عبر اتصال مفتوح، واستلام النتيجة من الخادم.
وسنتعرف في هذا الفصل على كيفية التعامل مع كائن الأمر.

واجهة أمر قاعدة البيانات IDbCommand Interface

هذه الواجهة تمثل الواجهة IDisposable، وهي توجـد فـي النطـاق System.Data..
وتمتلك هذه الفئة الخصائص التالية:

الاتصال Connection:

تستقبل هذه الخاصية أي كـائن اتصـال يمثـل الواجهـة IDbConnection، ليـتم استخدامه في تنفيذ الأمر.. ويشترط فتح الاتصال أولا قبل محاولة تنفيـذ الأمـر، وإلا حدث خطأ.

نوع الأمر CommandType:

تحدد نوع الأمر المراد تنفيذه، وهي تأخذ إحدى قيم المرقم CommandType التالية:

Text	الأمر يتكون من جملة SQL.. وهي القيمة الافتراضية.
StoredProcedure	الأمر يتكون من اسم إجراء مخزن يراد تنفيذه.
TableDirect	الأمر يتكون من اسم جدول يراد تحميل كل بياناته كاملة من قاعدة البيانات مباشرة.. هذه القيمة غير مقبولة مع قواعد بيانات سيكويل سيرفر، لكن يمكن استخدامها مع قواعد بيانات آكسيس.

نص الأمر CommandText:

جملة SQL التي تريد تنفيذها، أو اسم الإجراء المخزن الذي تريد تنفيذه، أو اسم الجدول المراد تحميله، وذلك تبعا لقيمة الخاصية CommandType.

لاحظ أنك تستطيع كتابة أكثر من جملة SQL في هذه الخاصية مع الفصل بينها بالفاصلة المنقوطة ; ، وفي هذه الحالة سيتم تنفيذها جميعا، والحصول على أكثر من مجموعة من النتائج Resultsets، وهو نفس ما يمكن أن يحدث عند استدعاء إجراء مخزن يقوم بتنفيذ أكثر من جملة SELECT.. وسنعرف كيف يمكن التعامل مع هذه النتائج لاحقا.

وقت انتظار الأمر CommandTimeout:

تحدد وقت الانتظار بالثانية، الذي سيتم بعده اعتبار محاولة تنفيذ الأمر فاشلة.. والقيمة الافتراضية لهذه الخاصية هي ٣٠ ثانية، وعليك أن تزيدها إذا كانت الاستعلام معقدا يتم على جداول كثيرة ويجري عليها الكثير من العمليات، أو كان هناك ضغط كبير على الخادم يجعل استجابته بطيئة.

المعاملات Parameters:

تعيد أي كائن يمثل واجهة "مجموعة معاملات البيانات" IDataParameterCollection، التي ترث واجهة القائمة IList.. وتتيح لك

مجموعة المعاملات إضافة المعاملات المطلوب التعويض بها في جملة الاستعلام أو الإجراء المخزن.. وسنتعرف على المعاملات بالتفصيل لاحقا.

التعامل Transaction:

تستقبل أي كائن من نوع واجهة تعاملات قاعدة البيانات IDbTransaction، ليتم تنفيذ الأمر الحالي في نطاقه.

مصدر الصفوف المحدّثة UpdatedRowSource:

تحدد كيف سيتم تحديث صفف البيانات DataRow الموجود في أحد جداول مجموعة البيانات DataSet.. لاحظ أن هذه الخاصية مفيدة فقط عندما يكون كائن الأمر الحالي هو أمر التحديث Update Command الخاص بمهيئ البيانات DbDataAdapter الذي يملأ مجموعة البيانات ويحدثها.. ويتم تنفيذ أمر التحديث على الصفوف التي تغيرت في مجموعة البيانات واحدا تلو الآخر، وقد يحتوي هذا الأمر على معاملات إخراج Output Parameters، أو ترافقه جملة SELECT تعيد السجل بعد تحديثه في قاعدة البيانات.. الحكمة في هذا أن هناك بعض القيم التي تولدها قاعدة البيانات بنفسها (مثل عمود الترقيم التلقائي) ولا يمكنك معرفة قيمة هذه الأعمدة إلا بالحصول على السجل مرة أخرى بعد تحديثه (أو إضافته).. وتتحكم هذه الخاصية في كيفية الاستفادة من القيم العائدة من أمر التحديث، وهي تأخذ إحدى قيم المرقم UpdateRowSource التالية:

None	يتم تجاهل أي معاملات أو صفوف عائدة من أمر التحديث.
OutputParameters	توضع قيم معاملات الإخراج Output Parameters في خانات سجل مجموعة البيانات DataSet.
FirstReturnedRecord	توضع قيم أول سجل عائد من أمر التحديث، في سجل مجموعة البيانات.

Both	توضع قيم معاملات الإخراج وأول سجل عائد من أمر التحديث، في سجل مجموعة البيانات.

كما تمتلك هذه الواجهة الوسائل التالية:

تجهيز Prepare:

تنشئ نسخة محسنة مجهزة من الأمر وتحفظها على الخادم، ليكون تنفيذها أسرع.. ولا تستقبل هذه الوسيلة معاملات، وليس لها قيمة عائدة، ولا يكون لها أي تأثير إن كانت للخاصية CommandType القيمة TableDirect.

لاحظ أن استدعاء هذه الوسيلة صار عديم القيمة تقريبا، لأن إصدارات سيكويل سيرفر ٢٠٠٠ و ٢٠٠٥ و ٢٠٠٨ تقوم بتجهيز البيانات تلقائيا عند اللزوم، لتحسين الأداء.

إنشاء معامل CreateParameter:

تعيد كائنا من النوع IDbDataParameter، لتخصصه للتعامل مع أحد المعاملات الموجودة في الاستعلام.

تنفيذ بدون استعلام ExecuteNonQuery:

تنفذ الأمر دون أن تعيد أية سجلات، ولكن تعيد عددا صحيحا Integer يمثل عدد السجلات التي تأثرت بتنفيذ الأمر.. ويمكنك استخدام هذه الوسيلة لتنفيذ أوامر التحديث Update والحذف Delete والإدراج Insert.

وستجد مثالا على هذه الوسيلة في الزر CREATE PROC في المشروع AccessStoredProcedure الذي عرفنا من قبل أنه ينشئ إجراء مخزنا في قاعدة بيانات الكتب المنشأة بتطبيق Access.. في هذا الزر نستخدم كائن أمر من النوع OleDbCommand للتعامل مع قاعدة بيانات Access، ونستخدم الوسيلة ExecuteNonQuery لتنفيذ استعلام SQL الذي ينشئ الإجراء المخزن في قاعدة البيانات، لأنه لا يعيد إلينا أي ناتج.

لاحظ أن كود الاتصال بقاعدة البيانات وتنفيذ الأوامر يتكرر كثيرا، وهو يبدو طويلا مع وجود تعديلات طفيفة.. لهذا سيكون من الأذكى لو جمعنا الكود المتشابه في إجراء وأرسلنا إليه المعاملات التي تناسب الاستعلام الذي ننفذه.. ولقد فعلنا هذا في المشروع DbTasks، حيث عرفنا فيه فئة اسمها MyDbConnector، وأضفنا إليها خاصية اسمها ConnectionStr تستقبل نص الاتصال، وحدث إنشاء Constuctor يستقبل نص الاتصال أيضا ويضعه في هذه الخاصية على سبيل الاختصار.. وقد عرفنا في هذه الفئة عددا من الوسائل التي تتيح لنا التعامل مع قاعدة البيانات، ومن بينها دالة اسمها ExcuteCommand.. هذه الدالة تستقبل معاملين:

- نص الأمر CommandText الذي تريد تنفيذه.. ويمكنك أن ترسل إلى هذا المعامل استعلام SQL أو اسم إجراء مخزن، حيث سنقوم باستنتاج نوع الأمر بحيلة صغيرة، فلو كان النص يبدأ بأي من أوامر SQL مثل " insert" أو " update"... إلخ، فمعنى هذا أنه نوع الأمر CommandType.Text.. لاحظ أننا وضعنا مسافة بعد اسم الكلمة، تلافيا لاحتمال أن تكون هذه الكلمة جزءا من اسم إجراء مخزن (مثل UpdateAuthors)، فنحن واثقون أن اسم الإجراء المخزن لا يحتوي على مسافات، بينما الاستعلامات تحتوي على مسافات.. غير هذا يكون نوع الأمر CommandType.StroresProcedure.
- مصفوفة نصية ثنائية البعد (,)String، يتكون كل عنصر فيها من اسم المعامل وقيمته.. وبهذا يمكنك تمرير المعاملات مباشرة إلى الدالة، حيث سنضيف هذه المعاملات إلى مجموعة معاملات الأمر Command.Parameters.. وإذا لم يكن للأمر معاملات، فأرسل إلى المعامل الثاني للدالة null.

وتستخدم هذه الدالة الوسيلة Command.ExecuteNonQuery لتنفيذ الأمر، وتعيد true إذا لم يحدث خطأ، وتعيد false إذا حدث خطأ.

وستجد مثالا على استخدام هذه الدالة في نفس المشروع، حيث وضعنا مربعي نص على النموذج لاستقبال اسم المؤلف ونبذة عنه، وعند ضغط الزر يتم تنفيذ استعلام

لإضافته إلى جدول الكتب.. انظر كيف سيكون الكود في منتهى البساطة والاختصـــار باستخدام الفئة MyDbConnector:

```
var DbBooks = new MyDbConnector(
    Properties.Settings.Default.BooksConStr);
var SQL = @"INSERT INTO Authors
                (Author, CountryID, About)
                VALUES (@Author, 21, @About)";
var Params = new[,] { { "@Author", TxtAuthor.Text },
                      { "@About", TxtAbout.Text } };
if (DbBooks.ExcuteCommand(SQL, Params))
{
   TxtAuthor.Clear( );
   TxtAbout.Clear( );
}
```

ويمكنك استخدام الفئة MyDbConnector للتعامل مع أي قاعدة بيانات، وتنفيـــذ أي أمر عليها باستخدام الإجراء ExcuteCommand.. أليس هذا شيئا مريحا؟

تنفيذ قارئ ExecuteReader:

تنفذ الأمر، وتعيد كائنا من النوع IDataReader، حيث يمكنـــك اســـتخدامه لقـــراءة النتيجة سجلا تلو سجلّ.. وسنتعرف على قارئ البيانات لاحقا.

وستجد مثالا على هذه الوسيلة في المشروع AuthorBooks_Reader.

ولهذه الوسيلة صيغة ثانية، تستقبل معاملا من نوع المـــرقم CommandBehavior، الذي يحدد سلوك قارئ البيانات، كالتالي:

Default	السلوك العادي، حيث يمكن أن يؤدي تنفيذ الأمر إلـــى الحصـــول على أكثر من مجموعة من مجموعات النتائج Result Sets (كما يحدث في حالة تنفيذ أكثر من جملة SQL مـــن داخـــل إجـــراء مخزن).. هذا مكافئ لاستدعاء الوسيلة ExecuteReader بـــدون معاملات.

SingleResult	يؤدي تنفيذ الاستعلام إلى الحصول على مجموعة نتائج واحدة فقط.
SchemaOnly	يؤدي تنفيذ الاستعلام إلى الحصول على معلومات الأعمدة فقط بدون أي سجلات.. هذا يعني الحصول على جدول فارغ به أسماء الأعمدة فقط.. هذا مماثل لاستخدام مزود سيكويل للخيار: SET FMTONLY ON
KeyInfo	يؤدي تنفيذ الاستعلام إلى الحصول على معلومات الأعمدة والمفتاح الأساسي Primary Key.
SingleRow	يؤدي تنفيذ الاستعلام إلى الحصول على سجل واحد فقط، ولو كان الاستعلام يحصل على أكثر من مجموعة من النتائج، فسيكون بكل مجموعة منها سجلا واحدا فقط.. استخدم هذا الاختيار عندما تحتاج إلى أو سجل فقط، فهذا يؤدي إلى تحسين أداء وسرعة البرنامج.
Sequential Access	قراءة تتابعية.. هذا مفيد للقراءة من الأعمدة التي تحوي قدرا ضخما من البيانات، فبدلا من طلبها كلها من الخادم، يتم طلب أجزاء من البيانات فقط تبعا لاحتياجك.. لاحظ الآتي: - يجب عليك قراءة قيم الحقول بنفس ترتيبها في الاستعلام، لأنك لو قرأت أي حقل، فلن تستطيع قراءة الحقل السابق له مرة أخرى، فنحن هنا نقرأ البيانات تتابعيا، أي بالترتيب. - استخدم الوسيلة GetValue لقراءة القيمة الموجودة في أي حقل كاملة. - استخدام الوسيلة GetBytes الخاصة بقارئ البيانات لقراءة أجزاء من الحقل الذي يحتوي على بيانات ثنائية ضخمة، مثل image و varbinary(MAX). - استخدام الوسيلة GetChars الخاصة بقارئ البيانات لقراءة

	أجزاء من الحقل الذي يحتوي على نصوص ضخمة، مثل text و ntext و varchar(MAX) و nvarchar(MAX). والمشروع ReadLargeData يقرأ الصور الخاصة بشعار كل ناشر من الجدول Publishers، ويحفظها في ملف على الجهاز.. ونظرا لأن الصورة قد تكون ضخمة، فقد استخدمنا هذا الخيار لنقرأ البيانات تتابعيا.. ويريك هذا المشروع أن هذه الطريقة تصلح للقراءة من الحقل Logo الذي نوعه image، وتصلح أيضا للقراءة من الحقل Logo2 الذي نوعه varbinary(MAX).
Close Connection	يتم إغلاق الاتصال Connection مع قاعدة البيانات أليا، بمجرد إغلاق قارئ البيانات.

ويمكنك استخدام أكثر من قيمة من هذه القيم، بربطها معا باستخدام المعامل |.

وقد أضفنا وسيلة اسمها GetReader إلى الفئة MyDbConnector في المشروع DbTasks، مهمتها تنفيذ استعلام باستخدام الوسيلة ExecuteReader وإعادة قارئ البيانات.. لاحظ أنك لو أغلقت الاتصال في نهاية هذه الوسيلة فسيحدث خطأ عند محاولتك استخدام قارئ البيانات الذي أعادته إليك، لأن الاتصال الذي يستخدمه قد تم إغلاقه.. لهذا عليك عدم إغلاق الاتصال، وإرسال القيمة CommandBehavior.CloseConnection إلى معامل الوسيلة ExecuteReader لجعل قارئ البيانات يغلق الاتصال بنفسه عندما يتم إغلاقه.. وقد جعلنا للوسيلة GetReader معاملا اختياريا اسمه Sequential إذا جعلته true فستحصل على قارئ بيانات تتابعي لاستخدامه في قراءة البيانات الضخمة على أجزاء، والقيمة الافتراضية لهذا المعامل هي false لتحصل على قارئ بيانات عادي.

وستجد مثالا لاستخدام هذه الوسيلة في نفس المشروع في زر "الكتب".. هذا الزر يعرض كتب المؤلف الذي كتبت اسمه في مربع النص العلوي.

تنفيذ قيمة ExecuteScalar:

تنفذ الأمر، وتعيد كائنا Object يحتوي على قيمة الخانة الموجودة في الصف الأوّل من العمود الأوّل في الجدول الناتج، وتتجاهل باقي الخانات.

ويمكنك استخدام هذه الوسيلة لتنفيذ دوال التجميع Aggregate Functions.

والمشروع AvgPrice يريك مثالا على استخدام هذه الوسيلة لمعرفة متوسط أسعار الكتب.

وقد أضفنا وسيلة اسمها GetValue إلى الفئة MyDbConnector في المشروع DbTasks، مهمتها تنفيذ استعلام باستخدام الوسيلة ExecuteScalar وإعادة القيمة الناتجة، وستجد مثالا لاستخدامها في نفس المشروع في الزر "معرفة عدد المؤلفين".

إلغاء Cancel:

تحاول إلغاء تنفيذ الأمر.. ولا يحدث خطأ إن لم يكن الأمر قيد التنفيذ حاليا، أو إن فشلت في إيقاف تنفيذه.

فئة أمر قاعدة البيانات DbCommand Class

هذه الفئة أساسية مجردة Abstract Base Class، وهي ترث فئة المكون Component Class، كما أنها تمثل الواجهة IDbCommand وبالتالي تمتلك جميع وسائلها وخصائصها.

وبالإضافة إلى ما تمثله من خصائص الواجهة IDbCommand، تمتلك هذه الفئة الخاصية التالية:

مرئية في وقت التصميم DesignTimeVisible:

إذا جعلت قيمة هذه الخاصية true (وهي القيمة الافتراضية)، فسيظهر كائن الأمر في وقت التصميم في واجهة الأدوات التي تستخدمه.

ولا تمتلك هذه الفئة أي وسائل جديدة غير ما تمثله من وسائل الواجهة IDbCommand.

لاحظ أن الفئات التالية ترث الفئة DbCommand:

١. OdbcCommand Class.
٢. OleDbCommand Class.
٣. SqlCommand Class.
٤. OracleCommand Class.

وسنكتفي هنا بالتعرف على الفئة SqlCommand.

فئة أمر سيكويل SqlCommand Class

هذه الفئة ترث الفئة DbCommand، وهي مخصصة للتعامل مع الأوامر التي يتم تنفيذها على خادم سيكويل سيرفر.

ولحدث إنشاء هذه الفئة أربع صيغ مختلفة:

١. الصيغة الأولى بدون معاملات.

٢. والصيغة الثانية لها معامل واحد، يستقبل نص الاستعلام الذي سيوضع في الخاصية CommandText.

٣. والصيغة الثالثة تزيد على الصيغة السابقة بمعامل ثان من النوع SqlConnection، يستقبل كائن الاتصال الذي سيتم تنفيذ الأمر من خلاله.

٤. والصيغة الأخيرة تزيد على الصيغة السابقة بمعامل ثالث من النوع SqlTransaction، يستقبل كائن التعامل الذي سيتم تنفيذ الأمر في نطاقه.

وإضافة إلى ما ترثه من الفئة الأم، تمتلك هذه الفئة الخاصيتين التاليتين:

التنبيه Notification:

تحدد كائن طلب التنبيه SqlNotificationRequest الذي سيستخدمه كائن الأمر في تلقي التنبيهات من الخادم عند تنفيذ الاستعلام.. وسنتعرف على الفئة SqlNotificationRequest لاحقا.

ضم تلقائي إلى قائمة التنبيهات NotificationAutoEnlist:

إذا جعلت قيمة هذه الخاصية true، فسيستقبل كائن الأمر تنبيهات تلقائية من كائن معلومات التبعية SqlDependency.. وتستخدم هذه الخاصية مع صفحات المواقع في ASP.NET لتتيح عرض الصفحة المجهزة Cashed Page إلى أن يأتي تنبيه بحدوث تغير في بعض بياناتها فيتم إنعاشها.. وسنتعرف على الفئة SqlDependency لاحقا.

وإضافة إلى ما ترثه من الفئة الأم، تمتلك هذه الفئة الوسائل التالية:

نسخ Clone:

تعيد كائن SqlCommand جديدا مماثلا في كل شيء للكائن الحالي.

تصفير زمن انتظار الأمر ResetCommandTimeout:

تعيد قيمة الخاصية CommandTimeout إلى قيمتها الافتراضية ٣٠ ثانية.

تنفيذ قارئ بيانات "XML" ExecuteXmlReader:

تنفذ الأمر، وتعيد كائنا من النوع XMLReader، الذي يحتوي على البيانات بتنسيق XML.. لاحظ أن عليك استخدام هذه الوسيلة في الحالات التالية:

- عند استخدام الفقرة FOR XML في جملة الاستعلام.. هذه الفقرة تعيد ناتج الاستعلام في صورة وثيقة XML.
- عند قراءة عمود نوع بياناته XML.
- عند قراءة عمود نوع بياناته ntext أو nvarchar لكنه يحتوي على بيانات بتنسيق XML.

ولن نتطرق إلى فئات XML في هذا الكتاب، وسنفرد لها كتابا مستقلا إن قدر الله.

وتدعم الفئة SqlCommand استخدام العمليات غير المتزامنة Asynchronous Operations لتنفيذ الأمر، وذلك من خلال أزواج الوسائل التالية:

BeginExecuteXmlReader **EndExecuteXmlReader**
BeginExecuteNonQuery **EndExecuteNonQuery**
BeginExecuteReader **EndExecuteReader**

حيث تقوم الوسائل التي تبدأ بالكلمة Begin بتنفيذ الأمر في عملية غير متزامنة، ويمكنك أن ترسل إليها مندوبا عن إجراء يتم استدعاؤه بعد انتهاء العملية لتقرأ فيه البيانات الناتجة..

وتعيد هذه الوسائل كائنا يمثل الواجهة IAsyncResult ليمكنك استخدامه في متابعة العملية، كما يمكن إرساله إلى الوسائل التي تبدأ بالكلمة End لإنهاء العملية غير المتزامنة والحصول على نتائجها.

وتمتاز العمليات غير المتزامنة بأنها لا توقف تنفيذ البرنامج إلى حين انتهاء إتمام العملية، بل ينتقل التنفيذ إلى السطر التالي مباشرة، بينما ترسل النتائج فور توفرها إلى الدالة الخاصة بالحصول على النتائج Callback Function.. وتقع العمليات غير المتزامنة خارج نطاق هذا الكتاب، وسنتعرف عليها بالتفصيل بإذن الله في كتاب المواضيع المتقدمة في برمجة إطار العمل.

كما تمتلك الفئة SqlCommand الحدث التالي:

اكتملت الجملة StatementCompleted:

ينطلق عند اكتمال تنفيذ جملة الاستعلام الخاصة بكائن الأمر.. والمعامل الثاني e لهذا الحدث من النوع StatementCompletedEventArgs، وهو يمتلك خاصية واحدة هي "عدد السجلات" RecordCount، التي تعيد عدد السجلات التي تأثرت بتنفيذ جملة الاستعلام.

تمرير القيم إلى جمل الاستعلام:

افترض أنك تريد الحصول على كتب "توفيق الحكيم" من قاعدة البيانات.. في هذه الحالة يمكنك وضع جملة SQL التالية في الخاصية CommandText لكائن الأمر (وليكن اسمه Cmd):

```
Cmd.CommandText = @"SELECT Books.Book
        FROM Authors, Books
        WHERE Authors.ID = AuthorID
        AND Authors.Author = 'توفيق الحكيم'";
```

ولكن، هل تظن أنك ستكتب الجملة السابقة في أي تطبيق عملي فعلا؟.. هل ستقتصر وظيفة برنامجك على عرض كتب مؤلف واحد فقط، أم أنك ستسمح للمستخدم باختيار المؤلف الذي يرده ليعرض له البرنامج كتب هذا المؤلف؟

المنطقي والعملي، هو أن تضع على النموذج مربع نص (وليكن اسمه TxtAuthor) ليكتب فيه المستخدم اسم المؤلف، ومن ثم تعرض له كتبه.. في مثل هذه الحالة، عليك تعديل نص الاستعلام السابق ليصير كالتالي:

```
Cmd.CommandText = @"SELECT Books.Book
        FROM Authors, Books
        WHERE Authors.ID = AuthorID
        AND Authors.Author = '" + TxtAuthor.Text +"'";
```

حيث استخدمنا طريقة تشبيك النصوص في الكود، لإضافة النص الموجود في مربع النص إلى جملة الاستعلام، وبهذا حصلنا على جملة استعلام مرنة، تستطيع البحث عن اسم أي مؤلف يريده المستخدم.

لكن هذه الطريقة تحتوي على ثغرة قاتلة، تسمح للأشقياء بتدمير قاعدة بياناتك وربما نظام التشغيل الذي يوجد عليه خادم سيكويل لو أرادوا!

كيف؟.. هذا هو موضوع الفقرة التالية.

دس الإستعلامات SQL Injection:

في المثال السابق، سمحنا للمستخدم بكتابة اسم المؤلف في مربع نص، ثم أدرجنا محتوى النص داخل جملة الاستعلام كجزء من شرط الفقرة WHERE.. وقد تفتق ذهن بعض العباقرة عن فكرة شريرة، وهي كتابة بعض جمل الاستعلام في مربع النص بدلا من اسم المؤلف، وبهذا يستطيعون حقن استعلامات مدسوسة خاصة بهم داخل جملة الاستعلام الخاصة بك، فيقوم برنامجك بتنفيذ ما يريدون على قاعدة البيانات، وهو أمر يشبه سلوك الفيروسات التي تحقن مادتها الوراثية في نواة الخلية وتتركها تنفذها لإنتاج فيروسات جديدة!

وخطورة هذه الطريقة، هي أنها تتيح للمخترقين الاستعلام عن بعض حسابات المديرين والمستخدمين، ومعرفة تركيب الجداول، بل وتنفيذ بعض أوامر غلاف الويندوز Shell من خلال خادم سكيويل، مما قد يضر بالجهاز الذي يعمل عليه الخادم!

ولكن كيف تتم عملية الحقن Injection؟

١- أول شيء، يتوقع القرصان Hacker ضرورة وجود النص بين علامتي تنصيص، ولا بد أنك وضعت علامة تنصيص بادئة قبل النص الذي سيأتي من مربع النص، لهذا يجب على المخترق أن يكتب أي كلمة، ثم يتبعها بالعلامة ' لإغلاق علامتي التنصيص، وبهذا يضمن عدم حدوث خطأ في صيغة جملة SQL.

٢- بعد هذا يضع القرصان فاصلة منقوطة ; ليستطيع كتابة أمر SQL جديد خاص به، وهنا تكون لديه الحرية في كتابة الأمر الذي يريده!

٣- نظرا لأن القرصان يتوقع منك إضافة تكملة لجملة SQL بعد النص الذي كتبه في مربع النص، فإنه يضع في نهاية الكود المدسوس الرمز -- ليجعل أي نص تال له مجرد تعليق، وبهذا يلغي أي تكملة خاصة بك لجملة الاستعلام، ويضمن سلامة صيغة جملة الاستعلام!

والآن، دعنا نرى ماذا سيحدث لو كتب القرصان في مربع النص الجملة التالية:

```
Ahamd'; drop table Books--
```

في هذه الحالة ستصبح جملة الاستعلام بعد إضافة هذه الجملة كالتالي:

SELECT Books.Book
FROM Authors, Books
WHERE Authors.ID = AuthorID
AND Authors.Author = 'Ahamd'; drop table Books--'

كما ترى: صار لدينا استعلامان صحيحان وتعليق:

- الاستعلام الأول لا قيمة له، وهو يبحث عن كتب مؤلف اسمه Ahmad.
- والاستعلام الثاني أمر حذف يطلب حذف جدول الكتب كاملا من قاعدة البيانات.. لاحظ أن القرصان لا يعرف أسماء الجداول، ولكنّ توقع اسم جدول الكتب لن يكون عسيرا، ولن ييأس القرصان من تجربة عشرات الأسماء المحتملة، ما دام عزمه قد قرّ على تدمير برنامجك!
- وفي النهاية يوجد تعليق صغير، هو العلامة ' الخاصة بك، والتي استطاع القرصان تهميشها بحيلة صغيرة بارعة!

يبدو الأمر مفزعا، أليس كذلك؟

إن هذه الثغرة تتيح للقراصنة تدمير قاعدة البيانات، ودخول حسابات المستخدمين، دون الحاجة إلى كتابة اسم المستخدم أو كلمة المرور، والكثير من الكوارث التي تكفي لتطير النوم من عيون المبرمجين ☺.

فكيف إذن يمكن إغلاق هذه الثغرة القاتلة؟

في الحقيقة هناك عدة نصائح هامة في هذا الصدد:

١- التقليل من استخدام مربعات النص، والاستعاضة عنها بأدوات تتيح اختيار القيم، مثل القوائم Lists إن كان هذا ممكنا.

٢- استخدام الخاصية MaxLength الخاصة بمربع النص لتحديد طول النص المسموح بكتابته في مربع النص.. هذا سيحد من قدرة القرصان على كتابة أوامر مدسوسة.

٣- إجراء بعض الفحوصات الصغيرة على قيمة مربع النص، للتأكد من خلو النص الذي كتبه المستخدم من العلامات المريبة مثل ; ' -- */ /* .. هذا سيشل

حركة القرصان تماما.. والأفضل أن تمنع كتابة هذه الحروف في مربع النص من المنبع باستخدام الحدث KeyPress.

٤- عليك أيضا أن تمنع الكلمات الدالة على أوامر SQL في مربع النص، خاصة DROP و DELETE و UPDATE و INSERT.

٥- لا تقبل أيا من الكلمات التالية في مربع نص يدخل فيه المستخدم اسم ملف:

AUX, CLOCK$, CON, CONFIG$, NUL, PRN
COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8
LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8

٦- من المهم أيضا أن تحدد صلاحيات مستخدمي قاعدة البيانات، وألا تعطي الصلاحيات الخطيرة (كحذف الجداول أو إنشائها) إلا للمديرين، وعليك أن تصنع نسخة خاصة من البرنامج لهؤلاء المديرين بحيث لا يتم تداولها إلا بينهم.. أما المستخدمون العاديون، فعليك أن تصنع لهم نسخة أخرى من البرنامج، وأن تتصل هذه النسخة بالخادم من خلال حساب مستخدم محدود الصلاحيات، وبهذا لو نجح أي قرصان في تجاوز خطوط دفاعك عبر هذه النسخة، لا يجد الكثير مما يستطيع فعله!

٧- كن حذرا من الكلمات التي تبدأ بـ xp_، لأنها البادئة التي يتم بها تسمية الإجراءات المخزنة الإضافية لمخطط قاعدة البيانات Catalog-extended stored procedures، مثل الإجراء xp_cmdshell.

٨- استخدام الإجراءات المخزنة Stored Procedures في تنفيذ الاستعلامات، لأنها تكون محفوظة في قاعدة البيانات على الخادم، ومن ثم يقوم سيكويل سيرفر بفحص قيم المعاملات المرسلة إلى الإجراء المخزن، والتأكد من أنها من النوع الصحيح وبالطول المحدد.

٩- ارفض كتابة الكلمات التالية في مربع النص، لأنها تسمح بحقن الأكواد المدسوسة في الإجراءات المخزنة:

EXECUTE, EXEC, sp_executesql

١٠- استخدام المعاملات Parameters لتمرير القيم إلى جمل الاستعلام، بدلا من استخدام طريقة تشبيك النصوص Concatination، لأن المعاملات تضمن التحقق من نوع المعامل والطول المسموح به لقيمته.. لكن نصيحة: لا تتخل عن فحص النصوص التي يكتبها المستخدم في مربعات النصوص، حتى لو استخدمت المعاملات، فقد تسمح بعض المعاملات النصية الطويلة بعبور بعض الكود المدسوس.

وفي المشروع AuthorBooks_Reader استخدمنا الإجراء SqlInjection للتأكد من أن النص الذي كتبه المستخدم في مربع النص لا يحتوي على أية رموز أو كلمات مريبة، كما استخدمنا معاملا لتمرير النص إلى الاستعلام لمزيد من الحماية.

كما استخدمنا الدالة SqlInjection في المشروع DbTasks للتأكد من أن قيم المعاملات لا تحتوي على استعلامات مدسوسة.

وسنتعرف فيما يلي على المعاملات وكيفية استخدامها.

المعاملات Parameters:

المعامل هو علامة موضعية Placeholder توضع في جملة SQL لتشير إلى أن هناك قيمة سيتم التعويض بها بدلا منها.. ويمكنك تعريف أي عدد تريده من المعاملات في جملة الاستعلام.

وتختلف صيغة هذه العلامة تبعا لنوع مزود البيانات، فمزود سيكويل يستخدم الرمز @ لتمييز المعامل، يتبعه اسم متغير بدون أن يفصل بينهما مسافات (مثل @UserName).. لهذا نستطيع كتابة الاستعلام عن كتب أحد المؤلفين كالتالي:

```
Cmd.CommandText = @"SELECT Books.Book
      FROM Authors, Books
      WHERE Authors.ID = AuthorID
      AND Authors.Author = @Author";
```

هذا يبدو كأننا عرفنا متغيرا اسمه @Author واستخدمناه في جملة الاستعلام، ليتم التعويض عنه عند تنفيذها.

أما في مزود OLEDB و ODBC فيتم استخدام علامة الاستفهام الإنجليزية ? للإشارة إلى وجود معامل، دون منح هذا المعامل أي اسم:

```
Cmd.CommandText = @"SELECT Books.Book
      FROM Authors, Books
       WHERE Authors.ID = AuthorID
       AND Authors.Author = ?";
```

لاحظ أن قواعد بيانات آكسيس صارت تقبل تسمية المعاملات (وما زالت تقبل العلامة ? أيضا)، لكنها لا تميز المعاملات المسماة باستخدام أي علامة خاصة.. يمكنك مثلا أن تكتب الاستعلام السابق كما يلي:

```
Cmd.CommandText = @"SELECT Books.Book
      FROM Authors, Books
       WHERE Authors.ID = AuthorID
       AND Authors.Author = AuthorValue";
```

في هذه الحالة ستعتبر آكسيس أن AuthorValue هو اسم معامل.. بل يمكنك أيضا أن تستخدم الاسم @Author في الاستعلام، وستقبله آكسيس كاسم معامل، وهذا يساعدك على استخدام نفس استعلامات سيكويل سيرفر مع آكسيس!

لكن هذا المرونة من آكسيس تسبب مشكلة غريبة في بعض الأحيان، فلو أخطأت مثلا في كتابة اسم الحقل AuthorID في الاستعلام السابق، وكتبته مثلا AutherID، فستعتبره آكسيس اسم معامل، وبدلا من أن تحصل في برنامجك على رسالة تخبرك أن هذا العمود ليس موجودا في الجدول، ستحصل على رسالة تخبرك بأن قيم بعض المعاملات مفقودة.. هذا هو السبب الذي سيجعلك ترى آلاف الأسئلة من المبرمجين عن سبب ظهور هذه الرسالة الغريبة في برنامجهم:

No value given for one or more required parameters"

رغم أنهم لا يستخدمون استعلامات فيها معاملات، أو أنهم مرروا قيم المعاملات الصحيحة فعلا!.. فكل ما هناك، أنهم أخطأوا في كتابة اسم أحد الحقول، فتم اعتباره معاملا!

ولكن، كيف يمكن التعويض عن قيم المعاملات؟

لفعل هذا، عليك استخدام مجموعة المعاملات الخاصة بكائن الأمر DbCommand.Parameters لتعريف كائنات المعاملات ووضع القيم فيها، حيث سيقوم كائن الأمر بتمريرها إلى جملة الاستعلام عند تنفيذ الأمر.. وعليك أن تنتبه جيدا إلى أن وضع رمز المعامل في جملة الاستعلام لا ينشئ معاملات في مجموعة المعاملات تلقائيا، فهذا الرمز يحدد فقط موضع التعويض عن المعامل، بينما تظل أنت مسئولا عن تعريف معامل في مجموعة المعاملات لتحديد نوع القيمة التي يقبلها المعامل، وتمرير القيمة من خلاله إلى كائن الأمر.

ويشترط في حالة سيكويل سيرفر أن يكون لكل من المعامل الموجود في نص الاستعلام والمعامل الموجود في مجموعة المعاملات نفس الاسم.. فإذا عرفت في نص الاستعلام معاملا اسمه @Author، فيجب أن يكون اسمه في مجموعة المعاملات أيضا @Author.

أما معاملات OLEDB و ODBC فكلها ممثلة بالرمز ? مما يجعلك مضطرا إلى تعريفها في مجموعة المعاملات DbCommand.Parameters بنفس ترتيب ظهورها في نص الاستعلام.. وحتى لو كنت تستخدم معاملات مسماة في آكسيس، فما زلت مضطرا إلى المحافظة على الترتيب الصحيح لهذه المعاملات، كما أن اسم كل معامل ما زال غير مهم،

لهذا تستطيع تعريف معامل في مجموعة المعاملات اسمه X ليعوض عن معامل في نـــص الاستعلام اسمه Author@.. فكل ما يهم حقا هو الترتيب وليس الاسم.. يمكنك اعتبـــار أن آكسيس يمحو اسم المعامل ويضع بدلا منه علامة استفهام ? على سبيل التسهيل عليك.

هذا هو ما جعل من الممكن أن نستخدم نفس جملة الاستعلام، ونفس كود تعريف المعاملات في حدث ضغط زر "الكتب" في المشروع Factories للحصول على كتب أحد المؤلفين من قاعدة الكتب في آكسيس أو قاعدة الكتب في سيكويل سيرفر.

والآن، دعنا نتعرف على مجموعة المعاملات والفئات المستخدمة معها.

فئة مجموعة معاملات قاعدة البيانات
DbParameterCollection Class

هذه المجموعة تمثل الواجهة IDataParameterCollection، التي ترث واجهة القائمة IList.. وكل عنصر يضاف إلى المجموعة DbParameterCollection هو من نوع الفئة DbParameter التي سنتعرف عليها لاحقا.

ولا تحتوي هذه الفئة على أية خصائص أو وسائل جديدة غير ما تمثله من عناصر الواجهة.

لاحظ أن الفئة DbParameterCollection أساسية مجردة تجب وراثتها، لهذا ترثها كل من الفئات التالية:

١. OdbcParameterCollection Class
٢. OleDbParameterCollection Class
٣. SqlParameterCollection Class
٤. OracleParameterCollection Class

ويعنينا هنا أن نتعرف على فئة مجموعة معاملات سيكويل SqlParameterCollection Class.

فئة مجموعة معاملات سيكويل
SqlParameterCollection Class

هذه الفئة ترث الفئة DbParameterCollection، وهي لا تختلف عنها كثيرا إلا في أن عناصرها من نوع الفئة SqlParameter ، التي سنتعرف عليها بعد قليل.
كما أن للوسيلة Add الخاصة بهذه الفئة العديد من الصيغ التي من المفيد أن نتعرف عليها:

إضافة Add:

تضيف معاملا إلى مجموعة المعاملات، ولها الصيغ التالية:

١- الصيغة الأولى تستقبل معاملا واحدا من النوع SqlParameter.
٢- الصيغة الثانية تستقبل نصا يمثل اسم المعامل وإحدى قيم المرقم SqlDbType التي توضح نوع المعامل.
٣- الصيغة الثالثة تزيد على الصيغة السابقة بمعامل ثالث من النوع Integer، يستقبل حجم البيانات التي ستوضع في المعامل.
٤- الصيغة الرابعة تزيد على الصيغة السابقة بمعامل، يستقبل اسم عمود في مجموعة البيانات DataSet ليربط المعامل به.
٥- وهناك صيغة خامسة لكنها لم يعد من المنصوح استخدامها، تستقبل اسم المعامل، وكائنا Object يحمل قيمته، وذلك بسبب التعارض بينها وبين الصيغة الثانية، لهذا تم إضافة وسيلة جديدة اسمها AddWithValue كبديل لهذه الصيغة.

إضافة بالقيمة AddWithValue:

تضيف معاملا إلى المجموعة، وهي تستقبل اسم المعامل، وكائنا Object يحمل قيمته.. وتعيد هذه الوسيلة مرجعا إلى المعامل الذي تم إنشاؤه.

لاحظ أنك تستطيع أن ترسل إلى المعامل الثاني لهذه الوسيلة قارئ بيانات DataReader أو جدول بيانات DataTable، لتتم قراءة كل الصفوف الموجودة فيهما ووضعها في المعامل.. هذا مفيد إذا كنت تتعامل مع إجراء مخزن يستقبل معاملا جدولا Table-Valued Parameter، وتريد أن ترسل إليه جدولا كاملا.. وستجد مثالا على هذا في المشروع TableValuedParameters.. في هذا المشروع نقرأ جدول المؤلفين في قارئ بيانات من النوع OledbDataReader، ثم نرسله كمعامل ثان إلى الوسيلة AddWithValue لاستخدامه كمعامل للإجراء المخزن InsertAuthors، وبهذا نستطيع إضافة المؤلفين من قاعدة بيانات آكسيس إلى قاعدة بيانات سيكويل سيرفر.

لاحظ أن قارئ البيانات يجب أن يظل مفتوحا هو والاتصال الذي يستخدمه، لأن الوسيلة AddWithValue لا تنسخ السجلات من قارئ البيانات فعليا، ولا يتم نسخ هذه السجلات إلا عند استدعاء الوسيلة ExecuteNonQuery الخاصة بكائن الأمر الذي ينفذ الإجراء المخزن ويرسل إله المعامل الجدول.

واجهة معامل البيانات IDataParameter Interface

هذه الواجهة معرفة في النطاق System.Data، وهي تمتلك الخصائص التالية:

اسم المعامل ParameterName:

تحدد اسم المعامل، مع ملاحظة أنه يبدأ بالرمز @، مثل @UserName.

هل هو منعدم IsNullable:

إذا جعلت قيمة هذه الخاصية true، فسيقبل المعامل القيمة DBNull.. والقيمة الافتراضية هي false.

الاتجاه Direction:

تحدد اتجاه المعامل، وهي تأخذ إحدى قيم المرقم ParameterDirection التالية:

Input	معامل إدخال، لإرسال قيمة إلى الاستعلام أو الإجراء المخزن.
Output	معامل إخراج، لقراءة قيمة متغير إخراج من الإجراء المخزن.. هذا النوع غر متاح مع قواعد بيانات آكسيس!
InputOutput	معامل إدخال وإخراج معا.
ReturnValue	قيمة عائدة من إجراء مخزن.

القيمة Value:

تقرأ أو تغير قيمة المعامل، وهي من النوع Object لتقبل أي نوع من القيم، لكن هذا لا يعني أن كل القيم مسموح بها، لأن الخاصية DbType تحدد نوع القيم المسموح بها.

ويجب وضع القيمة في هذه الخاصية قبل تنفيذ الاستعلام من خلال كائن الأمـــر، وإذا كان المعامل للإخراج، فإن القيمة العائدة من الخادم توضع في هذه الخاصية بعد تنفيذ الأمر أو بعد إغلاق قارئ البيانات DataReader إن كنت تستخدمه.

ولوضع القيمة DBNull في هذه الخاصية، استخدم الفئة DBNull كالتالي:

P.Value = DBNull.Value;

حيث إن Value هي خاصية ثابتـــة للقـــراءة فقـــط Static ReadOnly Property معرفة في الفئة DBNull لتعيد نسخة جديدة من هذه الفئة تمثل القيمة DBNull.

النوع DbType:

تحدد نوع المعامل، وهي تأخذ إحدى قيم المرقم DbType التالية:

الثابت	معناه	مجال القيم أو طول المتغير
SByte	وحدة ثنائية بإشارة.	من -١٢٨ إلى ١٢٧
Byte	وحدة ثنائية موجبة.	من ٠ إلى ٢٥٥
Binary	بيانات ثنائية	من ١ بايت إلى ٨٠٠٠ بايت
Boolean	قيمة منطقية	true أو false
Int16	عدد قصير	من -٣٢٧٦٨ إلى ٣٢٧٦٧
UInt16	عدد قصير موجب	من ٠ إلى ٦٥٥٣٥
Int32	عدد صحيح	من -٢١٤٧٤٨٣٦٤٨ إلى ٢١٤٧٤٨٣٦٤٧
UInt32	عدد صحيح موجب	من ٠ إلى ٤٢٩٤٩٦٧٢٩٥

الثابت	معناه	مجال القيم أو طول المتغير
Int64	عدد طويل	من -٩٢٢٣٣٧٢٠٣٦٨٥٤٧٧٥٨٠٨ إلى ٩٢٢٣٣٧٢٠٣٦٨٥٤٧٧٥٨٠٧
UInt64	عدد طويل موجب	من ٠ إلى ١٨٤٤٦٧٤٤٠٧٣٧٠٩٥٥١٦١٥
Single	عدد مفرد	من ١,٥ × (١٠ ^ -٤٥) إلى ٣,٤ × (١٠^٣٨) بدقة ٧ خانات عشرية.
Double	عدد مزدوج	من ٥,٠ × (١٠^-٣٢٤) إلى ١,٧ × (١٠^٣٠٨) بدقة ١٥ خانة عشرية.
Currency	عملة	من -٢ ^ ٦٣ إلى (٢ ^ ٦٣) -١ بدقة ٤ خانات عشرية.
Decimal	عدد عشري	من ١,٠ × (١٠^-٢٨) إلى ٧,٩ × (١٠^٢٨) بدقة تصل إلى ٢٨ خانة عشرية.
VarNumeric	رقم متغير	قيمة رقمية متغيرة الطول، تقبل أيا من الأنواع السابقة.
Guid	معـــرف عـــام متفـــرد	
Time	وقت	وقت بدون تاريخ
Date	تاريخ	تاريخ بدون وقت
DateTime	تاريخ ووقت	تقبل تواريخ بين ١٧٥٣/١/١ و ٩٩٩٩/١٢/٣١
DateTime2	تاريخ ووقت	تقبل تواريخ بين ١/١/١ و ٩٩٩٩/١٢/٣١

الثابت	معناه	مجال القيم أو طول المتغير
DateTimeOffset	إزاحة الوقت والتاريخ	
String	نص	نص متغير الطول، مكون من حروف موسعة Unicode.
StringFixedLength	نص ثابت الطول	نص ثابت الطول، مكون من حروف موسعة Unicode.
AnsiString	نص قياسي	نص متغير الطول، مكون من حروف قياسية بترميز ASCII.
AnsiStringFixedLength	نص قياسي ثابت الطول	نص ثابت الطول، مكون من حروف قياسية بترميز ASCII.
Xml	XML	صفحة XML أو جزء منها.
Object	كائن	أي نوع غير موجود في الأنواع السابقة.

عمود المصدر SourceColumn:

تحدد اسم العمود في مجموعة البيانات DataSet الذي سيتم ربطه بالمعامل.. هذا مفيد عند حفظ التغيرات من مجموعة البيانات إلى قاعدة البيانات باستخدام أمر التحديث Update Command.. هذه هي الخطوات التي تحدث:

- يعرّف أمر التحديث Update Comman معاملا لكل عمود في مجموعة البيانات، ويضع اسم العمود في الخاصية SourceColumn لكل معامل.
- عند تنفيذ عملية التحديث، يقوم مهيئ البيانات DataAdapter بالمرور على صفوف مجموعة البيانات واحدا تلو الآخر وتنفيذ أمر التحديث على كل منها على حدة.
- لتحديث أي صف، يضع مهيئ البيانات في كل معامل من معاملات أمر التحديث، قيمة الخانة الموجودة في العمود المحدد في الخاصية

DbParameter.SourceColumn، وبهذا يمكن تنفيذ استعلام التحديث لكل صف بصورة صحيحة.

وستفهم هذه الأمور بصورة أوضح عندما نتعرف على مهيئ البيانات DataAdapter ومجموعة البيانات DataSet.

إصدار المصدر SourceVersion:

تحدد نوع القيمة التي ستوضع في المعامل.. هذا مفيد عند استخدام المعامل لتحديث مجموعة البيانات، لأن كل خانة في مجموعة البيانات تحتفظ بعدة أنواع من القيم، مثل القيمة الأصلية (القادمة من قاعدة البيانات)، والقيمة الحالية (القيمة الجديدة التي أدخلها المستخدم).. وتأخذ هذه الخاصية إحدى قيم المرقم DataRowVersion.. وسنتعرف على هذا الأمر بتفصيل أكبر لاحقا عند التعرف على مجموعة البيانات.

واجهة معامل بيانات قاعدة البيانات
IDbDataParameter Interface

هذه الواجهة ترث الواجهة IDataParameter، وهي تمتلك بعض الخصائص الإضافية التي تضيف مزيدا من التحكم في القيم التي يقبلها المعامل.. وهذه الخصائص هي:

الحجم Size:

تحدد أقصى حجم مسموح به للمعامل بالوحدة الثنائية Byte.. والقيمة الافتراضية لهذه الخاصية تستنتج من نوع المعامل، فإن كان عددا صحيحا على سبيل المثال، تكون قيمتها ٤، وإن كان المعامل يحتوي على مصفوفة فإن هذه الخاصية تأخذ طول المصفوفة.. لاحظ أنك لو صغرت قيمة هذه الخاصية عن حجم البيانات، فسيتم أخذ جزء من هذه البيانات فقط وإسقاط الجزء الزائد.

الدقة Precision:

تحدد أكبر عدد مسموح به من الخانات الرقمية في القيمة التي يقبلها المعامل.. والقيمة الافتراضية هي ٠، وهي تعني عدم فرض قيود على عدد الخانات، وترك ذلك لمزود قاعدة البيانات.

المقياس Scale:

تحدد أكبر عدد مسموح به من الخانات العشرية في القيمة التي يقبلها المعامل.. ولو زاد عدد الخانات عن هذا الرقم يتم تقريبه.. والقيمة الافتراضية هي ٠ .

فئة معامل قاعدة البيانات DbParameter Class

هذه الفئة معرفة في النطاق System.Data.Common، وهي فئة أساسية مجردة تجب وراثتها، تمثل الواجهة IDbDataParameter مما يعني أنها تمثل أيضا الواجهة IDataParameter، وبالتالي فهي تملك كل خصائصهما، ولا تزيد عليها إلا خاصية واحدة جديدة وهي:

تمثيل قيمة منعدمة في عمود المصدر SourceColumnNullMapping:

إذا جعلت قيمة هذه الخاصية true، فسيعني هذا أن هذا المعامل يستخدم لإخبارك إن كان عمود المصدر SourceColumn في مجموعة البيانات فارغا أم لا، حيث تكون قيمة المعامل ٠ إن كان العمود فارغا، وتكون قيمته ١ إن كان العمود يحتوي على أي قيمة.. هذا مفيد عند تعريف استعلامات التحديث، لأن مقارنة أي خانتين قيمتهما Null تكون نتيجته false رغم أن الخانتين متساويتين فعلا!.. لهذا يجب أن نتأكد قبل إجراء المقارنة إن كانت الخانتين فارغتين أم لا.. وسنرى كيف نستخدم هذه الطريقة عند التعرف على مهيئ البيانات في فصل لاحق.

كما تمتلك هذه الفئة وسيلة واحدة، وهي:

تصفير النوع ResetDbType:

تعيد الخاصية DbType إلى قيمتها الافتراضية.

والفئات التالية ترث هذه الفئة:

١. OdbcParameter Class
٢. OleDbParameter Class
٣. SqlParameter Class
٤. OracleParameter Class

وسنقتصر هنا على التعرف على الفئة SqlParameter.

فئة معامل سيكويل SqlParameter Class

هذه الفئة ترث الفئة DbParameter، وهي تمثل معامل استعلام موجه إلى سيكويل سيرفر.

ولحدث إنشاء هذه الفئة سبع صيغ، الصيغة الأولى بدون معاملات، والصيغ الأخرى تتيح لك إمداد المعامل ببعض قيم خصائصه، مثل اسم المعامل ونوعه واتجاهه.. إلخ.. على سبيل المثال، تستقبل الصيغة السابعة ١٣ معاملا هي بالترتيب:

- اسم المعامل.
- إحدى قيم المرقم SqlDbType توضح نوع المعامل.
- عدد صحيح يوضح طول المعامل إذا كان نصا أو وحدات ثنائية Bytes.. بالنسبة للمعاملات العددية استخدم القيمة صفر.. وإذا كان المعامل من الأنواع القصوى (MAX)، فاستخدم القيمة -١.
- إحدى قيم المرقم ParameterDirection التي توضح اتجاه المعامل.
- وحدة ثنائية Byte توضح عدد الخانات العشرية للمعامل الرقمي (الخاصية Precision).
- وحدة ثنائية Byte توضح عدد خانات التقريب العشري للمعامل الرقمي (الخاصية Scale).
- اسم عمود المصدر في مجموعة البيانات، الذي سيأخذ منه المعامل قيمته.
- إحدى قيم المرقم DataRowVersion، توضح إصدار العمود في مجموعة البيانات.. وسنتعرف على هذا المرقم في فصل لاحق.
- قيمة منطقية، إذا جعلتها true، فستكون وظيفة معامل البيانات أن يتأكد أن العمود ليس فارغا Null في مجموعة البيانات، حيث تكون قيمة المعامل ٠ إذا كان العمود فارغا، وتكون قيمته ١ إذا لم يكن فارغا.
- كائن Object يحمل القيمة التي تريد تمريرها إلى المعامل.. وعليك أن تستخدم القيمة null، إذا كان المعامل سيقرأ القيمة من مجموعة البيانات.

- نص ترسل قيمته إلى الخاصية XmlSchemaCollectionDatabase التي سنتعرف عليها لاحقا.
- نص ترسل قيمته إلى الخاصية XmlSchemaCollectionOwningSchema التي سنتعرف عليها لاحقا.
- نص ترسل قيمته إلى الخاصية XmlSchemaCollectionName التي سنتعرف عليها لاحقا.

وهناك صيغة تستقبل فقط أول أربعة معاملات من الصيغة السابقة، وفي هذه الحالة يكون اتجاه المعامل للإدخال، ويقرأ القيمة الحالية للسجل Current Value.. وهناك صيغة أخرى تستقبل كل معاملات الصيغة السابقة ما عدا آخر ثلاثة معاملات.

والكود التالي يعرف معاملا يقرأ قيمته من الحقل Book في مجموعة البيانات:

```
var PrmBook = new SqlParameter("@Book",
    SqlDbType.NVarChar, 0, "Book");
```

والكود التالي يعرف معاملا يأخذ قيمته من النسخة الأصلية للحقل ID في مجموعة البيانات:

```
var PrmID = new SqlParameter("@Original_ID",
    SqlDbType.Int, 0, ParameterDirection.Input, false, 0,
    0, "ID", DataRowVersion.Original, null);
```

ويمكنك استخدام هذين المعاملين لتعريف أمر التحديث Update كالتالي:

```
SqlCommand UpdateCmd = new SqlCommand( );
UpdateCmd.CommandText = @"UPDATE Books
            SET Book = @Book
            WHERE ID = @Original_ID";
UpdateCmd.Connection = SqlConnection1;
UpdateCmd.Parameters.AddRange(
          new SqlParameter[] {PrmBook, PrmID});
```

ويمكنك جعل هذا الأمر أمر التحديث الخاص بمهيئ البيانات كالتالي:

```
DaAuthorBooks.UpdateCommand = UpdateCmd;
```

وستجد هذا الكود في حدث تحميل النموذج Load في المشروع ViewAndEditBooks.. ويمكنك نقل التغييرات من مجموعة البيانات إلى قاعدة البيانات

بضغط زر الحفظ، حيث يقوم مهيئ البيانات DaAuthorBooks بتنفيذ الوسيلة Update، التي تستخدم أمر التحديث والمعاملات التي عرفناها.
وإضافة إلى ما ترثه من الفئة الأم، تمتلك الفئة SqlParameter الخصائص التالية:

المعرف المحلي LocaleId:

تحدد معرف الثقافة المحلية للغة التي تريد استخدامها في التعامل مع قيمة المعامل.. على سبيل المثال: معرف اللغة العربية بالنظام السداسي عشري هو: 0x0001 . (راجع كتاب: "برمجة إطار العمل"، للمزيد من التفاصيل عن الثقافات العالمية ومعرفاتها).

معلومات المقارنة CompareInfo:

تحدد كيف سيقوم هذا المعامل بمقارنة النصوص، وهي تأخذ إحدى قيم المرقم SqlCompareOptions التي تعرفنا عليها في الفصل السادس.

الإزاحة Offset:

تحدد موضع بداية القراءة من الخاصية Value.. وتقاس الإزاحة بعدد الوحدات الثنائية Bytes عند التعامل مع بيانات ثنائية، وبعدد الحروف Characters عند التعامل مع نصوص.. والقيمة الافتراضية هي ٠، أي أن القراءة تتم من بداية البيانات.
وتحدد الخاصية Size عدد الحروف أو الوحدات الثنائية التي ستتم قراءتها بدءا من الموضع Offset.. هذا مفيد إذا وضعت في الخاصية Value كما كبيرا من البيانات، وأردت تجزئتها دون الحاجة إلى متغيرات وسيطة وحيل برمجية ملتوية.. في هذه الحالة ستعطي للخاصية Offset مبدئيا القيمة صفر وللخاصية Size الطول الذي تريده (وليكن ١٠٠).. بعد هذا تستخدم حلقة تكرار Loop تنفذ فيها الأمر على قاعدة البيانات (حيث سيقرأ الأمر الجزء المحدد فقط في المعامل بدءا من الموضع Offset

وبالطول Size)، ومن ثم تزيد قيمة الخاصية Offset بمقدار ١٠٠ لقراءة جزء تال.. ويستمر الدوران وتنفيذ هذه العملية إلى أن تتجاوز طول البيانات الموجودة في الخاصية Value.. لاحظ أن هذه الطريقة مناسبة للاستخدام فقط مع الأمر Write. Update.. وستجد مثالا على هذا في الزر Parameter.Offset في المشروع Write Large Data، وهو يسمح لك بإضافة صورة شعار في العمود Logo2 للناشر الثاني.. لاحظ أننا حملنا كل محتويات ملف الصورة مرة واحدة ووضعناها في الخاصية Value للمعامل، ومن ثم أرسلناها إلى قاعدة البيانات على أجزاء صغيرة.. هذه الطريقة تحتوي على عيب كبير، وهو أن الصورة إذا كانت ضخمة جدا، فستستهلك مساحة كبيرة من الذاكرة وقد تسبب بطء البرنامج.. لهذا لا تستخدم هذه الطريقة إلا إذا كان حجم الصورة معقولا، أما إذا كان ضخما، فالأفضل أن تحمل أجزاء من الملف وترسلها إلى قاعدة البيانات بالطريقة التي استخدمناها في الزر Write. Update في نفس البرنامج.

نوع بيانات سيكويل SqlDbType:

تحدد نوع المعامل من بين أنواع البيانات في سيكويل سيرفر، وهي تأخذ إحدى قيم المرقم SqlDbType، وهي تحمل نفس أسماء أنواع بيانات سيكويل سيرفر التي تعرفنا عليها بالتفصيل في الفصل الثالث.. والقيمة الافتراضية لهذه الخاصية هي NVarChar.

لاحظ أن هذه الخاصية مرتبطة بالخاصية DbType، لهذا لو غيرت قيمة إحداهما فستتغير الأخرى تلقائيا، بحيث تحتوي الخاصية SqlDbType دائما على أنسب نوع من أنواع سيكويل سيرفر يوافق نوع المتغير الموجود في الخاصية DbType.. جرب مثلا:

```
SqlParameter P = new SqlParameter( );
P.SqlDbType = SqlDbType.Money;
MessageBox.Show(P.DbType.ToString( )); // Currency
P.DbType = DbType.Int64;
MessageBox.Show(P.SqlDbType.ToString( )); // BigInt
```

اسم النوع TypeName:

إذا كنت تستخدم المعامل لإرسال قيمة إلى معامل جدول Table-Valued، فضع اسم هذا النوع في هذه الخاصية، متضمنا اسم المالك Owner.. فمثلا، للتعامل مع النوع AuthorType الذي يستخدمه الإجراء المخزن InsertAuthors، وضعنا في هذه الخاصية النص "dbo.AuthorType" أو باختصار "AuthorType" لأن المالك هنا افتراضي.. وفي هذه الحالة لا بد ان نضع في الخاصية SqlDbType القيمة SqlDbType.Structured.. وهذا هو ما فعلناه في المشروع TableValuedParameters.

اسم المتغير الخاص بالمستخدم UdtTypeName:

إذا كنت تستخدم المعامل لإرسال قيمة إلى نوع من تعريف المستخدم User-Defined Type، فضع اسم هذا النوع في هذه الخاصية، ولا تنسَ أن تضع في الخاصية SqlDbType القيمة SqlDbType.Udt.. وسنتعرف على المتغيرات التي يعرفها المستخدم في سيكويل سيرفر لاحقا.

قيمة سيكويل SqlValue:

مماثلة للخاصية Value، وكلتاهما تقرأ أو تغير قيمة المعامل.

قاعدة بيانات المخطط XmlSchemaCollectionDatabase:

تعيد اسم قاعدة البيانات التي توجد بها مجموعة مخططات XML.. وإذا كانت قيمة هذه الخاصية نصا فارغا، فهذا معناه أن المخططات توجد في قاعدة البيانات الحالية، أو أنه لا توجد مخططات أصلا، وفي الحالة الأخيرة ستكون قيمة الخاصيتين XmlSchemaCollectionName
و XmlSchemaCollectionOwningSchema نصا فارغا.

اسم مجموعة المخططات XmlSchemaCollectionName:

تعيد اسم مجموعة مخططات XML.

XmlSchemaCollectionOwningSchema:

تعيد مخطط العلاقات الرئيسي، الذي يحدد موضع مجموعة مخططات XML.

-٨-

قارئ البيانات DataReader

يتم إنشاء قارئ البيانات DataReader باستدعاء الوسيلة ExecuteReader الخاصة بكائن الأمر Command Object.. ويستقبل قارئ البيانات نتيجة الاستعلام الذي ينفذه كائن الأمر، ويقوم بتخزين ما يصل من البيانات من الخادم في المخزن الوسيط للشبكة Network Buffer الموجود على جهاز العميل، حيث يمكنك المرور عبر السجلات المستلمة واحدًا تلو الآخر على التوالي، مما يوفر ميزتين هامتين:

١- السرعة: حيث يمكنك قراءة السجلات المتوفرة فور وصولها، دون انتظار اكتمال وصول كل السجلات أولا.

٢- عدم استهلاك الذاكرة: لأن قارئ البيانات يحتفظ بسجل واحد فقط في الذاكرة في كل مرة.

لكنّ لهذه الطريقة عيبين أساسيين:

١- عدم القدرة على تحديث سجلات قاعدة البيانات.. بعبارة أخرى: قارئ البيانات للقراءة فقط كما يقول اسمه، وليس للكتابة!

٢- عدم القدرة على التراجع إلى الخلف، أو القفز مباشرة إلى سجل في موضع معين في النتيجة دون المرور على ما قبله من السجلات.

لهذا يوصف قارئ البيانات بأنه "مجرى بيانات للأمام فقط وللقراءة فقط":

Forward-only, Read-only Stream.

لكل هذا، يمكنك استخدام قارئ البيانات في الحالات التالية:

١- لو كنت ستتعامل مع سجلّ واحد فقط.

٢- لو كنت ستقرأ كل سجل مرة واحدة فقط، ولا يعنيك الرجوع إليه مرة أخرى.
٣- لو كانت قاعدة البيانات موجودة على نفس الجهاز، مما يعني سرعة الحصول على البيانات منها مباشرة، دون الحاجة إلى تحميلها في الذاكرة.
٤- عندما تريد قراءة النتائج دون الحاجة إلى تغيير أي جزء منها في قاعدة البيانات.

واجهة سجل البيانات IDataRecord Interface

تقدم هذه الواجهة الخصائص والوسائل اللازمة لقراءة محتويات السجل الحالي في قارئ البيانات.

وتمتلك هذه الواجهة الخاصيتين التاليتين:

عدد الحقول FieldCount:

تعيد عدد الأعمدة في السجل.. هذا يتيح لك كتابة حلقة تكرار Loop للمرور عبر كل الأعمدة بدءا من العمود رقم صفر إلى العمود رقم FieldCount - 1.. هذا مفيد لاختصار الكود عندما تستخدم استعلام يعيد عددا كبيرا من الحقول.

المفهرس Indexer:

يستقبل رقم العمود أو اسمه كمعامل، ويعيد كائنا Object يحتوي على القيمة الموجودة في السجل الحالي في هذا العمود.. والمثال التالي يعرض قيمة الخانة الاولى في الصف:

```
MessageBox.Show(Reader[0].ToString( ));
```

كما تمتلك هذه الواجهة الوسائل التالية:

معرفة الاسم GetName:

أرسل إلى هذه الوسيلة رقم العمود، لتعيد إليك اسمه.

معرفة الترتيب GetOrdinal:

أرسل إلى هذه الوسيلة اسم العمود، لتعيد إليك رقمه.

معرفة نوع الحقل GetFieldType:

أرسل إلى هذه الوسيلة رقم العمود، لتعيد إليك كائن النوع Type الذي يمثل نوع بياناته.

معرفة اسم نوع البيانات GetDataTypeName:

أرسل إلى هذه الوسيلة رقم العمود، لتعيد إليك نصا يمثل اسم نوع بياناته.

معرفة القيمة GetValue:

أرسل إلى هذه الوسيلة رقم العمود، لتعيد إليك كائنا Object يحتوي على القيمة الموجودة في السجل الحالي في هذا العمود.

معرفة القيم GetValues:

أرسل إلى هذه الوسيلة مصفوفة كائنات Object Array، ليتم ملؤها بالبيانات الموجودة في خانات الصف الحالي في قارئ البيانات.. لاحظ أنك لو أرسلت مصفوفة أقصر من عدد خانات الصف الحالي فلن يحدث خطأ، بل سيتم نسخ جزء من الخانات فقط إلى المصفوفة وإهمال الباقي.. أما لو أرسلت مصفوفة أطول من عدد خانات الصف الحالي، فسيتم نسخ كل الخانات إلى جزء من المصفوفة وترك باقي المصفوفة فارغا.. وفي كل الأحوال، تعيد هذه الوسيلة عدد الخانات التي تم نسخها إلى المصفوفة.

قراءة الوحدات الثنائية GetBytes:

تستخدم عندما يكون قارئ البيانات تتابعيا Sequential، وهي تعيد مصفوفة وحدات ثنائية Bytes، تحتوي على البيانات الموجودة في أحد الأعمدة.. ولهذه الوسيلة المعاملات التالية:

- رقم العمود.
- موضع بداية القراءة من محتويات العمود.
- مصفوفة وحدات ثنائية Bytes لاستقبال البيانات.. لاحظ أن خطأ سيحدث لو كانت المصفوفة أقصر من البيانات المطلوبة.
- موضع بداية الكتابة في المصفوفة.

- عدد الوحدات الثنائية Bytes المطلوب قراءتها من محتويات العمود، بدءا من الموضع المحدد في المعامل الثاني.. لاحظ أن خطأ سيحدث لو كان الطول المطلوب أكبر من البيانات المتبقية في العمود.

وتعيد هذه الوسيلة عدد الوحدات التي تم نسخها إلى المصفوفة، ولو أرسلت إلى هذه الوسيلة مصفوفة فارغة null، فستعيد إليك العدد الإجمالي للوحدات الثنائية المتاحة في الخانة.

وقد استخدمنا هذه الوسيلة في المشروع ReadLargeData لقراءة بيانات الصورة، حيث نقرأ ١٠٠ وحدة ثنائية Byte من بداية الصورة ونحفظها في الملف، ثم نقرأ ١٠٠ وحدة تالية ونحفظها في الملف، ونستمر في فعل هذا إلى أن نكمل قراءة الصورة.. لاحظ أن شرط التوقف عن القراءة، هو أن تكون القيمة العائدة من الوسيلة GetBytes أصغر من عدد البيانات الذي طلبنا قراءته، مما يعني أن هذه هي آخر بيانات متاحة في الخانة.

قراءة الحروف GetChars:

تستخدم عندما يكون قارئ البيانات تتابعيا Sequential، وهي تعيد مصفوفة حروف Char Array تحتوي على الحروف الموجودة في عمود نصي، وهي مماثلة للوسيلة السابقة في المعاملات والقيمة العائدة، فيما عدا أن المعامل الثالث يستقبل مصفوفة حروف بدلا من مصفوفة الوحدات الثنائية.

هل القيمة منعدمة IsDBNull:

تعيد true إذا كانت الخانة التي أرسلت رقمها كمعامل فارغة DbNull.. لاحظ أن عليك استخدام هذه الوسيلة قبل محاولة قراءة قيمة أي خانة، فقارئ البيانات يسبب خطأ إذا كانت قيمة الخانة NULL.. هكذا مثلا يمكنك محاولة قراءة الخانة الموجودة في العمود الأول في الصف الحالي:

```
if (! Reader.IsDBNull(0))
    MessageBox.Show(Reader[0].ToString( ));
```

كما تمتلك هذه الواجهة عددا من الوسائل التي تستقبل رقم العمـــود، وتعيـــد قيمـــة الخانـــة الموجودة في هذا العمود في الصف الحالي.. وتختلف هذه الوسائل في نوع القيمة العائـــدة منها، حيث تقوم كل منها بتحويل بيانات الخانة إلى أحد أنواع إطار العمل الأساســية، كمـــا هو موضح في الجدول التالي:

	الوسيلة	النوع الذي تعيده
	**GetByte**	وحدة ثنائية Byte.
	**GetChar**	حرف Char.
	**GetBoolean**	قيمة منطقية Boolean.
	**GetInt16**	عدد قصير Short.
	**GetInt32**	عدد صحيح Integer.
	**GetInt64**	عدد طويل Long.
	**GetFloat**	عدد مفرد float.
	**GetDouble**	عدد مزدوج Double.
	**GetDecimal**	عدد عشري Decimal.
	**GetDateTime**	تاريخ ووقت DateTime.
	**GetString**	نص String.
	**GetGuid**	سجل المعرف المتفرد Guid Structure.

وتسبب هذه الوسائل خطأ في البرنامج إذا فشلت في تحويل البيانات إلى النوع المطلوب.

فئة سجل البيانات DbDataRecord Class

هذه الفئة تمثل الواجهة IDataRecord، وهي تمتلك كل وسائلها وخصائصها دون أن تزيد عليها شيئا.

وتستخدم هذه الفئة مع واجهة العداد للمرور عبر سجلات قارئ البيانات، كما سنرى لاحقا.

واجهة قارئ البيانات IDataReader Interface

ترث هذه الواجهة كلا من الواجهتين IDisposable و IDataRecord.
وإضافة إلى ما ترثه من خصائص، تمتلك هذه الواجهة الخصائص الجديدة التالية:

العمق Depth:

تعيد رقما يمثل عمق الجدول الحالي، إذا كانت النتيجة تحتوي على جداول متداخلة (خانات بها جداول، بها خانات بها جداول... إلخ)، مع ملاحظة أن الجدول الخارجي يكون عمقه صفرا، وأول جدولا داخلي عمقه ١ ... وهكذا.

هل هو مغلق IsClosed:

تعيد true إذا تم إغلاق قارئ البيانات.

السجلات المتأثرة RecordsAffected:

تعيد عدد الصفوف التي تأثرت بأوامر الإضافة أو التحديث أو الحذف.. وتعيد هذه الوسيلة ٠ إذا لم تتأثر أية سجلات أو فشل تنفيذ الاستعلام، وتعيد -١ إذا كان الاستعلام يستخدم الأمر SELECT.
لاحظ أن هذه الخاصية لا تعطيك القيمة الصحيحة إلا بعد إغلاق قارئ البيانات، لهذا عليك أن تتأكد أولا أن للخاصية IsClosed القيمة true، أو تستخدم الخاصة RecordsAffected بعد استخدام الوسيلة Close.

كما تمتلك هذه الواجهة هذه الوسائل الجديدة:

قراءة Read:

تجعل قارئ البيانات ينتقل إلى السجل التالي، وتعيد true.. أما إذا كان السجل الحالي هو آخر سجل ولا يوجد سجل تال، فإنها تعيد false، وعليك التوقف عن القراء في هذه الحالة، وإلا حدث خطأ.

لاحظ أن قارئ البيانات يشير مبدئيا إلى السجل رقم -١، أي أنه يشير إلى السجل السابق لأول سجل، وهذا سيجعل محاولة القراءة تسبب خطأ في البرنامج، حيث ستخبرك رسالة الخطأ أن هذه محاولة غير مسموح بها للقراءة بينما لا توجد بيانات حاليا:

Invalid attempt to read when no data is present.

لهذا عليك استدعاء الوسيلة Read أولا للانتقال إلى أول سجل وقراءته، ثم الاستمرار في استدعائها إلى أن تعيد false، وذلك على الصيغة التالية:

```
while (Reader.Read( )) {
        // الكود اللازم لقراءة السجل الحالي
}
```

النتيجة التالية NextResult:

عند استخدام كائن الأمر لتنفيذ أكثر من جملة SQL، أو تنفيذ إجراء مخزن يعيد أكثر من نتيجة، فإن قارئ البيانات يشير مبدئيا إلى أول نتيجة، وعليك بعد قراءة كل سجلاتها أن تستخدم هذه الوسيلة لجعل قارئ البيانات يشير إلى النتيجة التالية.. وتعيد هذه الوسيلة true إذا وجدت نتيجة تالية، لهذا عليك أن تستمر في استدعائها إلى أن تعيد false، وذلك على الصيغة التالية:

```
do
{
   while (Reader.Read( ))
   {
      // الكود اللازم لقراءة السجل الحالي
   }
} while (Reader.NextResult( ));
```

قراءة جدول المخطط GetSchemaTable:

تعيد كائن جدول DataTable فارغا يحتوي على مخطط النتيجة التي يتعامل معها قارئ البيانات.. وسنتعرف على كائن الجدول لاحقا.

لاحظ أن المخطط يبحتوي على اسماء الأعمدة وأنواع بياناتها وأحجامها.. ويمكنك أن ترى هذا المخطط بشكل عملي في المشروع SchemaTable.. في هذا المشروع استخدمنا قارئ بيانات ليحمل جدول المؤلفين، واستخدمنا الوسيلة GetSchemaTable للحصول على مخطط جدول المؤلفين، وعرضناه في جدول عرض البيانات DataGridView الذي سنتعرف عليه بالتفصيل في فصل لاحق.

إغلاق Close:

تغلق قارئ البيانات.. هذا ضروري لتحرير كائن الاتصال المرتبط بقارئ البيانات، لأنك لن تستطيع استخدام كائن الاتصال في أي عملية أخرى طالما كان قارئ البيانات يستخدمه.

وكما ذكرنا من قبل، لو أنشأت قارئ البيانات باستخدام الوسيلة ExecuteReader بالصيغة التالية (حيث Cmd هو اسم كائن الأمر):

```
var Dr = Cmd.ExecuteReader(
        CommandBehavior.CloseConnection);
```

فإن كائن الأمر Dr سيقوم بإغلاق الاتصال بقاعدة البيانات تلقائيا بمجرد استدعاء الوسيلة Close.

فئة قارئ البيانات DbDataReader Class

هذه الفئة أساسية مجردة تجب وراثتها، وهي موجودة في النطاق:
System.Data.Common
وتمثل هذه الفئة الواجهة IDataReader، مما يعني أنها تمثل أيضا الواجهتين IDataRecord و IDisposable.. كما أنها تمثل أيضا واجهة القابلية للعد IEnumerable، مما يعني أنها تمتلك الوسيلة GetEnumerator التي تعيد عدادا يمر عبر سجلات النتيجة واحدا بعد الآخر، مع ملاحظة أن كل عنصر في هذا العداد هو من نوع الفئة DbDataRecord.. لهذا تستطيع المرور عبر سجلات قارئ البيانات باستخدام الوسيلة Read كما شرحنا سابقا، أو باستخدام حلقة التكرار foreach على الصيغة التالية:

```
foreach (DbDataRecord R in Dr)
{
   MessageBox.Show(R[0].ToString( ));
   MessageBox.Show(R[1].ToString( ));
}
```

هذا الكود سيعرض محتويات أول وثاني حقل في كل سجل من سجلات قارئ البيانات، بافتراض أن قارئ البيانات اسمه Dr وأن النتيجة بها حقلان أو أكثر.
وإضافة إلى ما ترثه من خصائص، تمتلك الفئة DbDataReader الخاصيتين الجديدتين التاليتين:

HasRows به صفوف:

تعيد true إذا كان قارئ البيانات يتعامل مع نتيجة بها صفوف.

VisibleFieldCount عدد الحقول المرئية:

تعيد عدد الأعمدة غير الخفية في قارئ البيانات.

كما تمتلك هذه الفئة الوسائل الجديدة التالية:

معرفة نوع الحقل طبقا للمزود GetProviderSpecificFieldType:

أرسل إلى هذه الوسيلة رقم العمود، لتعيد إليك كائن النوع Type الذي يمثل نوع بياناته.. هذا النوع لن يكون من أنواع إطار العمل الأساسية، بل سيكون من الأنواع الخاصة بمزود البيانات.. على سبيل المثال، لو كنت تتعامل مع مزود سيكويل سيرفر، وكان العمود يحتوي على نصوص، فإن هذه الوسيلة ستعيد كائنا يمثل نوع الفئة SqlString وليس الفئةString.

قراءة القيمة طبقا للمزود GetProviderSpecificValue:

أرسل إلى هذه الوسيلة رقم العمود، لتعيد إليك كائنا Object يحمل قيمته.. لاحظ أن هذه القيمة ستكون من الأنواع الخاصة بمزود البيانات، لهذا سيسبب المثال التالي خطأ (بافتراض أن العمود رقم صفر عمود نصي):

```
var Name = (string) Dr.GetProviderSpecificValue(0);
MessageBox.Show(Name);
```

والصواب أن تستخدم الكود التالي:

```
var Name = (SqlString) Dr.GetProviderSpecificValue(0);
MessageBox.Show(Name.Value);
```

قراءة القيم طبقا للمزود GetProviderSpecificValues:

أرسل إلى هذه الوسيلة مصفوفة كائنات Object Array، لتملأها لك ببيانات الصف الحالي في قارئ البيانات.. وتعيد هذه الوسيلة عدد الخانات التي تم ملؤها في المصفوفة.

والفئات التالية ترث الفئة DbDataReader:

١- DataTableReader Class
٢- OdbcDataReader Class
٣- OleDbDataReader Class
٤- SqlDataReader Class
٥- OracleDataReader Class

وسنكتفي هنا بالتعرف على الفئة SqlDataReader، وسنتعرف في فصل لاحق على الفئة DataTableReader.

فئة قارئ بيانات سيكويل SqlDataReader Class

هذه الفئة ترث الفئة DbDataReader بكل وسائلها وخصائصها، وهي تمكنك من قراءة البيانات القادمة من خادم سيكويل.

وليس لهذه الفئة حدث إنشاء، ولكنك تستطيع الحصول على نسخة منها باستدعاء الوسيلة ExecuteReader الخاصة بأمر سيكويل SqlCommand.

ولا تمتلك هذه الفئة أية خصائص جديدة غير ما ترثه من الفئة الأم، ولكنها تمتلك العديد من الوسائل الجديدة، وهي تقوم بقراءة البيانات من العمود الذي أرسلت رقمه إليها كمعامل، وتحولها إلى نوع البيانات المطلوب.. ولأسماء هذه الوسائل الصيغة العامة GetX، حيث X هو اسم نوع البيانات الذي سيتم التحويل إليه.. وهذه الوسائل هي:

- GetSqlBinary
- GetSqlByte
- GetSqlChars
- GetSqlDecimal
- GetSqlGuid
- GetSqlInt32
- GetSqlMoney
- GetSqlString
- GetSqlValues
- GetTimeSpan
- GetSqlBoolean
- GetSqlBytes
- GetSqlDateTime
- GetSqlDouble
- GetSqlInt16
- GetSqlInt64
- GetSqlSingle
- GetSqlValue
- GetSqlXml

على سبيل المثال، الوسيلة GetSqlBinary تعيد كائنا من النوع SqlBinary، وهي مناسبة لقراءة الأعمدة التي تحتوي بيانات من النوع image أو varbinary(MAX)، لهذا استخدماها في الزر GetSqlBinary في المشروع ReadLargeData، لقراءة صورة أول ناشر من العمود Logo2 (ويمكنك استخدامها أيضا للقراءة من العمود

Logo)، وحفظها في ملف.. لاحظ أن عملية القراءة ستتم هنا بطريقة مباشرة (غير تتابعية)، وأن حجم الصورة سيؤثر على كفاءة هذه العملية، فلو كانت ضخمة فسيأخذ نقلها وقتا طويلا، وسيتم تحميلها في الذاكرة كاملة قبل حفظها في الملف!

بهذا نكون قد أكملنا تعرفنا على قارئ البيانات.. وللتدريب على ما تعلمناه حتى الآن، يمكنك فحص المشروع AuthorBooks_Reader.. في هذا المشروع نسمح للمستخدم بكتابة اسم المؤلف في مربع نص، وعندما يضغط الزر "عرض الكتب"، نستخدم قارئ البيانات لكتابة كتب هذا المؤلف في مربع نص متعدد الأسطر.

لا تنسَ نسخ قاعدة بيانات الكتب من القرص الضوئي إلى المحرك C: والتأكد من أنها ليست للقراءة فقط، لكي يعمل المثال بشكل صحيح.

ملحوظة:

لاستخدام الإجراء المخزن GetAuthorBooks للحصول على كتب المؤلف الذي ترسل إلى هذا الإجراء اسمه كمعامل، أجر التعديلات التالية على المشروع AuthorBooks_Reader:

- ضع في الخاصيّة CommandText الخاصة بكائن الأمر اسم الإجراء المخزّن GetAuthorBooks.
- ضع في الخاصية CommandType الخاصة بكائن الأمر القيمة StoredProcedure.
- لا تُجرِ أية تعديلات على المعامل Author@ الذي أضفناه إلى مجموعة معاملات كائن الأمر.

هذا فقط هو كل المطلوب، وسيعمل البرنامج بشكل سليم، وسيعطي نفس النتائج التي كان يعطيها سابقا، مع اختلاف واحد: أنه يستخدم الإجراء المخزن بدلا من جملة SQL.

والمشروع AuthorBooks_Reader2 يحتوي بالفعل على هذه التعديلات.

– ٩ –

مهيئ البيانات DataAdapter

مهيئ البيانات هو مجرد حلقة وصل بين كائن الاتصال Connection Object ومجموعة البيانات DataSet، ومهمته هي إيصال البيانات من الخادم إلى مجموعة البيانات أو العكس.

ويتكون مهيئ البيانات في الحقيقة من أربعة كائنات أوامر Command Objects، تحتوي على جمل SQL اللازمة لتحديد البيانات من الخادم SELECT، وتحديثها UPDATE وإدراج سجلات جديدة INSERT وحذف سجلات موجودة DELETE، وبهذا يتيح لمجموعة البيانات حرية التعامل مع قاعدة البيانات في كلا الاتجاهين (الاستقبال والإرسال)، على عكس قارئ البيانات DataReader، الذي يقرأ البيانات فقط.

في الحقيقة، فإن مهيئ البيانات يستخدم قارئ البيانات Data Reader داخليا لتنفيذ أمر التحديد.. لكن هذا مجرد جزء من قدرات مهيئ البيانات، فهو يستطيع إرسال التغييرات من مجموعة البيانات إلى قاعدة البيانات، كما يتيح لك عمل خراط للجداول Table Mapping ، وذلك بإعادة تسمية الجداول والأعمدة بأسماء خاصة بك، وربطها بالأسماء الحقيقية في قاعدة البيانات، وهو ما سنتعرف عليه بالتفصيل في نهاية هذا الفصل.

دعنا إذن نتعرف على الواجهات والفئات التي تصنع مهيئ البيانات.

واجهة مهيئ البيانات IDataAdapter Interface

تقدم هذه الواجهة العناصر الأساسية لمهيئ البيانات، وهي تمتلك الخصائص التالية:

خرائط الجداول TableMappings:

تعيد كائنا يمثل المجموعة ITableMappingCollection، يحتوي على خرائط الجداول التي يستخدمها مهيئ البيانات لربط الجداول من مجموعة البيانات بالجداول الأصلية في قاعدة البيانات.. وسنتعرف على هذا الموضوع بالتفصيل لاحقا.

التصرف عند غياب الخريطة MissingMappingAction:

تحدد ماذا سيحدث في حالة عدم وجود خريطة لبعض الجداول في مجموعة البيانات، وهي تأخذ إحدى قيم المرقم MissingMappingAction التي سنتعرف عليها لاحقا.

والقيمة الافتراضية لهذه الخاصية هي Passthrough، مما يعني أن مجموعة البيانات تستخدم نفس أسماء جداول قاعدة البيانات، ولا حاجة إلى وجود خريطة للجداول في هذه الحالة.

التصرف عند غياب المخطط MissingSchemaAction:

تحدد ماذا سيحدث في حالة عدم وجود بعض الجداول أو الأعمدة في مجموعة البيانات، وهي تأخذ إحدى قيم المرقم MissingSchemaAction التي سنتعرف عليها لاحقا.

والقيمة الافتراضية لهذه الخاصية هي Add، مما يعني أن الجدول الناقص أو العمود الناقص سيضاف إلى مجموعة البيانات تلقائيا عند ملئها بالبيانات.

كما تمتلك هذه الواجهة الوسائل التالية:

ملء Fill:

أرسل إلى هذه الوسيلة كائن مجموعة البيانات DataSet التي تريد ملأهـا بالجـداول والسجلات الناتجة من تنفيذ أمر التحديد SELECT الخاص بمهيـئ البيانـات.. وإذا كانت مجموعة البيانات تحتوي على السجلات فعلا، فسيحدث أحد الاحتمالين التاليين:

١- إذا كـان هنـاك مفتـاح أساسـي Primary Key أو قيـد تفـرد Unique Constraint لسجلات مجموعة البيانات، فسيتم إنعاشها مـن جديـد بأحدث قيم موجودة في قاعدة البيانات.

٢- إذا لم يكن هناك ما يميز كل سجل ويمنع تكراره، فستضـاف السـجلات إلـى مجموعة البيانات مرة أخرى، مما يجعلها مكررة!

وتعيد هذه الوسيلة عدد السجلات التي تمت إضافتها أو تحديثها.

لاحظ أنك لا تحتاج إلى فتح الاتصال مع قاعدة البيانات أولا، فهذه الوسيلة تقوم بفتحه إن كان مغلقا ثم تعيد إغلاقه.. أما لو كان الاتصال مفتوحا قبل استدعاء هذه الوسـيلة، فإنها تستخدمه ثم تتركه مفتوحا كما هو.

ويسمى كل جدول يضاف إلى مجموعة البيانـات تبعـا لخريطـة الجـدول Table Mapping إن وجدت.. فإن لم توجد هذه الخريطة، تستخدم قواعد التسمية التالية:

١- إذا كان أمر التحديد يعيد سجلات جدول واحد فقط، فإنها تضاف في جدول يسمى Table.

٢- إذا كان أمر التحديد ينفذ أكثر من استعلام ويعيد سجلات أكثر من جدول، فإن كل نتيجة منها توضع في جدول مستقل، ويتم تسمية هذه الجداول بالترتيب Table و Table1 و Table2... وهكذا.. لاحظ أن حدوث أي خطـأ فـي أي اسـتعلام، سيمنع تنفيذ الاستعلامات التالية له ولن توضع باقي النتائج في مجموعة البيانات.

٣- يمكنك استخدام أكثر من مهيئ بيانات لملء نفس مجموعة البيانات بالجداول.. في هذه الحالة سيضيف مهيئ البيانات الأول جدولا اسمه Table، وسيحاول مهيـئ البيانات الثاني أن يضيف جدولا اسمه Table أيضا، لكن نظرا لأنـه موجـود، فسترفضه مجموعة البيانات ولن تتم إضافته، لكن لن يحدث خطأ في البرنامج!..

هذا يوضح لك ضرورة استخدام خريطة الجدول لتسمية كل من الجدولين باسمين مختلفين لحل هذه المشكلة.

٤- يسمى كل عمود في مجموعة البيانات، بنفس اسمه في الجدول الأصلي في قاعدة البيانات.

٥- إذا كانت النتيجة تحتوي على أكثر من عمود بنفس الاسم (بسبب استخدام استعلام يجمعها من أكثر من جدول من قاعدة البيانات) فإنها توضع في الجدول بعد إضافة الأرقام (١، ٢، ٣ ...) إلى نهاية اسم العمود لمنع التشابه.

٦- إذا كانت بعض الأعمدة بدون أسماء (لأنها ناتجة عن دوال تجميع مثلا) فإنها تعطى الأسماء الافتراضية Column1 و Column2 و Column3... وهكذا.

ويجب عليك ألا تضيف إلى مجموعة البيانات جداول أو أعمدة خاصة بك وتسميها بهذه الأسماء الافتراضية، كي لا يحدث أي تعارض أو خطأ بسببها.. وستجد مثالا على استخدام الوسيلة Fill في المشروع DataGridViewAuthorBooks.. في هذا المشروع نستتخدم مهيئ بيانات اسمه DAAuthors لتحميل سجلات المؤلفين من قاعدة البيانات، كما نستخدم مهيئ بيانات اسمه DABooks لتحميل سجلات الكتب من قاعدة البيانات.. ولملء مجموعة بيانات اسمها Ds بسجلات المؤلفين والكتب، نستخدم الكود التالي في حدث تحميل النموذج Form1_Load:

```
DAAuthors.Fill(Ds);
DABooks.Fill(Ds);
```

ملء المخطط FillSchema:

تملأ مجموعة البيانات بمخطط البيانات Schema.. هذا معناه أن الجداول الناتجة عن الاستعلام وتفاصيل أعمدتها ستضاف إلى مجموعة البيانات، لكن دون إضافة أي سجلات إليها.. بتعبير آخر: سيتم ملء مجموعة البيانات بجداول فارغة.

وتستقبل هذه الوسيلة معاملين:

- كائن مجموعة البيانات DataSet الذي سيتم ملؤه بالمخطط.

- إحدى قيم المرقم SchemaType، تحدد ماذا سيحدث لو كان مهيئ البيانات يحتوي مسبقا على خرائط للجداول والأعمدة Mappings، وهذه القيم هي:

Source	تجاهل خرائط الجداول وخرائط الأعمدة، وملء مجموعة البيانات بنفس أسماء الجداول والأعمدة الأصلية الموجودة في المخطط Schema.
Mapped	استخدام خرائط الجداول وخرائط الأعمدة، وملء مجموعة البيانات بالأسماء الموجودة في هذه الخرائط بدلا من أسماء الجداول والأعمدة الأصلية.

ويتضمن المخطط الذي يتم ملء مجموعة البيانات به التفاصيل التالية:

- اسم الجدول، وأسماء الأعمدة.
- خصائص كل عمود، مثل:

أ. السماح بتركه فارغا AllowDBNull.

ب. هل هو متفرد Unique.

ج. هل هو للقراءة فقط ReadOnly.

د. هل يزيد تلقائيا AutoIncrement.. لكن عليك أنت تحديد معدل الزيادة وبداية العداد، فهما لا يضافان تلقائيا.

هـ. أقصى طول للبيانات في العمود MaxLength.

- إذا كان المفتاح الأساسي Primary Key موجودا ضمن أعمدة النتيجة، يتم استخدامه كمفتاح أساسي للجدول في مجموعة البيانات.. وإذا لم يوجد مفتاح أساسي وكان هناك حقل متفرد القيمة Unique، يتم استخدامه كمفتاح أساسي للجدول في مجموعة البيانات، بشرط ألا يكون مسموحا بتركه فارغا (AllowDbNull = False).. أما إذا كان الحقل المتفرد يقبل القيمة NULL، فلن يستخدم كمفتاح أساسي، وستكتفي هذه الوسيلة بإضافة قيد التفرد

UniqueConstraint الخاص بهذا العمود إلى مجموعة القيود ConstrainsCollection الخاصة بالجدول.

- أي قيود أخرى غير المفتاح الأساسي وقيد التفرد لا تضاف إلى الجدول، وعليك إضافتها بنفسك!

وتعيد هذه الوسيلة مصفوفة جداول DataTable Array، تحتوي على كائنات الجداول التي تمت إضافة مخططاتها إلى مجموعة البيانات.

ويمكنك بعد استخدام هذه الوسيلة، استخدام الوسيلة Fill لملء الجداول بالبيانات.. لكن كما ذكرنا سابقا، فإن استدعاء الوسيلة Fill بمفردها يضيف مخططات الجداول والسجلات معا إلى مجموعة البيانات، مما يغني في معظم الحالات عن استدعاء الوسيلة FillSchema أولا.

إذن.. فما فائدة هذه الوسيلة؟

تفيدك هذه الوسيلة إذا أردت عرض الجداول فارغة للمستخدم ليدخل بيانات جديدة دون أن يعبث بالبيانات القديمة.. هذا هو ما فعلناه في المشروع UpdateErrors، حيث سيعرض جدول البيانات أعمدة جدول المؤلفين، لكنه لن يعرض بيانات أي مؤلفين، وبهذا يستطيع المستخدمن إدخال مؤلفين جددا وحفظهم في قاعدة البيانات، دون أن يغير بيانات المؤلفين السابقين.

تفيدك أيضا إذا أردت إنشاء المفتاح الأساسي في جدول مجموعة البيانات، فاستخدام الوسيلة Fill بمفردها لا ينشئ المفتاح الأساسي في الجدول الذي تضيفه إلى مجموعة البيانات، وهذا قد يسبب لك مشاكل في بعض الحالات التي تحتاج فيها إلى المفتاح الأساسي، كما يحدث عند استخدام الوسيلة DataSet.Merge لدمج السجلات، أو عند البحث عن سجل بدلالة مفتاحه الأساسي... إلخ.

معرفة معاملات الملء GetFillParameters:

تعيد مصفوفة معاملات IDataParameter Array، تحتوي على المعاملات التي استخدمها كائن الأمر عند تنفيذ أمر التحديد SELECT الخاص بمهيئ البيانات.

تحديث Update:

أرسل إلى هذه الوسيلة كائن مجموعة البيانات DataSet التي تريد نقل التغييرات التي حدثت على سجلاتها إلى قاعدة البيانات.. وتعيد هذه الوسيلة عدد السجلات التي نجح تحديثها في قاعدة البيانات.

وتستخدم هذه الوسيلة أوامر الإضافة والحذف والتحديث الخاصة بمهيئ البيانات، لنقل التغييرات من مجموعة البيانات إلى قاعدة البيانات، تبعا للتغيير الذي حدث لكل سجل.. ويحدث خطأ في البرنامج، إذا لم يوفر مهيئ البيانات الأمر المطلوب من هذه الأوامر.

لاحظ أن هذه الوسيلة ذكية، فهي تمر عبر كل سجل في مجموعة البيانات، وترى إن كان هناك أي تغيير قد حدث لهذا السجل، ومن ثم تستخدم الأمر المناسب لإرسال هذا التغيير إلى مجموعة البيانات.. أما السجلات التي لم يحدث بها أي تغيير، فيتم تجاهلها.. وبهذا لا تضيع هذه لوسيلة أي وقت في محاولة حفظ سجلات لم يحدث فيها تغيير.. لذا فأنت لا تحتاج إلى القيام بأية خطوات خاصة لتحسين وظيفة هذه الوسيلة.

لكن، لو كانت مجموعة البيانات تحتوي على أكثر من جدول، فأيها يا ترى سيتم تحديثه؟

يحدد هذا مهيئ البيانات الذي تستخدمه.. مثلا: لو استخدمت مهيئ بيانات لملء مجموعة البيانات بجدول المؤلفين، فإن الوسيلة Update الخاصة به ستتعامل مع سجلات جدول المؤلفين.. ولو استخدمت مهيئ بيانات لملء مجموعة البيانات بجدول الكتب، فإن الوسيلة Update الخاصة به ستتعامل مع سجلات جدول الكتب.. لهذا تحتاج الجملتين التاليتين لحفظ التغييرات:

DaAuthors.Update(Ds);
DaBooks.Update(Ds);

لاحظ أن هذا الكود صحيح ولكنه قد يسبب مشاكل في بعض الحالات، والأفضل أن تحاول تحديث الجدول الفرعي قبل الجدول الرئيسي، فلو كنت حذفت مؤلفا وكتبه، فإن محاولة تحديث جدول المؤلفين أولا ستحاول حذف سجل هذا المؤلف، وهذا سيسبب خطأ إذا كنت فرضت قيد المفتاح الفرعي Foreign Key Constraint، لأن كتب هذا المؤلف ستشير إلى مؤلف غير موجود.. بينما لو عكست العملية، وحدثت جدول الكتب أولا، فسيتم حذف كتب المؤلف بلا مشاكل، ومن ثم يتم حذف المؤلف نفسه عند تحديث جدول المؤلفين.. لهذا عليك استخدام الكود التالي في عملية التحديث:

DaBooks.Update(Ds);
DaAuthors.Update(Ds);

وهذا هو الكود الذي استخدمناه في الزر "حفظ في قاعدة البيانات" في المشروع DataSetSample.

اللهم ارحم أبي واغفر له

واجهة مهيئ بيانات قاعدة البيانات
IDbDataAdapter Interface

هذه الواجهة ترث الواجهة IDataAdapter، وهي تمد مهيئ البيانات بالأوامر اللازمة للتعامل مع قاعدة البيانات.
وإضافة إلى ما ترثه من الواجهة الأم، تمتلك هذه الواجهة الخصائص التالية:

أمر التحديد SelectCommand:

هذه الخاصية من نوع الواجهة IDbCommand، وهي تستقبل كائن الأمر الذي يحتوي على جملة التحديد SELECT التي سيستخدمها مهيئ البيانات للحصول على السجلات من قاعدة البيانات عند استدعائك للوسيلة Fill أو FillSchema.. ويمكن أن يحتوي أمر التحديد على مجموعة من الأوامر SQL Batch، وفي هذه الحالة يضيف مهيئ البيانات أكثر من جدول إلى مجموعة البيانات.

أمر الإدراج InsertCommand:

هذه الخاصية من نوع الواجهة IDbCommand، وهي تستقبل كائن الأمر الذي يحتوي على جملة الإدراج INSERT التي يستخدمها مهيئ البيانات عند استدعاء الوسيلة Update، لإضافة السجلات الجديدة في مجموعة البيانات إلى قاعدة البيانات.
هذا المشروع يستخدم استعلام الإدراج التالي:

```
INSERT INTO Authors
(Author, CountryID, Phone, About)
VALUES (@Author, @CountryID, @Phone, @About)
```

ويريك المشروع CopyAuthors كيف يمكن تعريف أمر الإدراج ومعاملاته، ووضعه في الخاصية InsertCommand، واستدعاء الوسيلة Update لاستخدامه في حفظ السجلات الجديدة من مجموعة البيانات إلى قاعدة البيانات.. لاحظ أن معاملات أمر الإدراج مربوطة بمجموعة البيانات، لأخذ قيمها من أعمدتها مباشرة.. لقد عرفنا من قبل أن كائن المعامل Parameter Object يملك الخاصية

SourceColumn التي نضع فيها اسم العمود الذي سنقرأ القيمة منه من مجموعة البيانات.. وستجد أننا أرسلنا القيمة إلى هذه الخاصية من خلال المعامل الرابع لحدث الإنشاء New عند تعريف كل معامل، كما هو موضح في الكود التالي:

```
InsertCmd.Parameters.AddRange(new SqlParameter[] {
        new SqlParameter("@Author",
                SqlDbType.NVarChar, 0, "Author"),
        new SqlParameter("@CountryID",
                SqlDbType.SmallInt, 0, "CountryID"),
        new SqlParameter("@Phone",
                SqlDbType.VarChar, 0, "Phone"),
        new SqlParameter("@About",
                SqlDbType.NVarChar, 0, "About")});
```

لهذا لا نحتاج إلى كتابة أي كود لقراءة القيم من مجموعة البيانات، فعند استدعاء الوسيلة Update، فإنها تمر عبر كل صف من صفوف مجموعة البيانات، وتأخذ قيم أعمدته وتمررها إلى معاملات أمر الإدراج، وتنفذ أمر الإدراج.

ملحوظة:

لإنعاش السجل المعروض في مجموعة البيانات، يمكنك استخدام جملة SELECT بعد استعلام الإدراج أو التحديث الخاص بقواعد بيانات سيكويل سيرفر، وذلك بوضع فاصلة منقوطة ; بين الأمرين.. مثلا:

```
INSERT INTO Authors
(Author, CountryID, Phone, About)
VALUES (@Author, @CountryID, @Phone, @About);
SELECT * FROM Authors
WHERE ID = SCOPE_IDENTITY( )
```

هذا مفيد في بعض الحالات.. مثلا: لو كان السجل يحتوي على عمود ترقيم تلقائي، فإن الرقم الذي ستعطيه مجموعة البيانات للسجل هو مجرد اقتراح لا تعمل قاعدة البيانات به (لهذا ليست مشكلة أن تضع فيه مجموعة البيانات ٠ أو -١)، حيث تعطي قاعدة البيانات للسجل الرقم الصحيح في الترقيم عند إضافته إليها، لهذا تكون جملية SELECT مفيدة لعرض الترقيم الصحيح للسجل في برنامجك.

أما إذا لم تجد داعيا لإنعاش مجموعة البيانات، فلا تستخدم أمر التحديد.
لاحظ أن الدالة SCOPE_IDENTITY تعيد آخر معرف تم توليده في نطاق التنفيذ الحالي، وهو بالطبع معرف السجل الذي أضفناه للتو.. ولا ينصح باستخدام الدالة @@IDENTITY لأنها قد تتأثر بعوامل أخرى في قاعدة البيانات مثل المنبهات Triggers، مما يجعلها تعيد معرفا غير المعرف الخاص بالسجل الذي أضفته.

أمر الحذف DeleteCommand:

هذه الخاصية من نوع الواجهة IDbCommand، وهي تستقبل كائن الأمر الذي يحتوي على جملة الحذف DELETE التي يستخدمها مهيئ البيانات عند استدعاء الوسيلة Update، لحذف السجلات من قاعدة البيانات إذا كانت قد حذفت من مجموعة البيانات.

أمر التحديث UpdateCommand:

هذه الخاصية من نوع الواجهة IDbCommand، وهي تستقبل كائن الأمر الذي يحتوي على جملة التحديث UPDATE التي يستخدمها مهيئ البيانات عند استدعاء الوسيلة Update، لنقل التغييرات من مجموعة البيانات إلى قاعدة البيانات.

ملحوظة:

إذا كانت مجموعة البيانات تحتوي على مفتاح أساسي، وكنت قد وضعت أمر التحديد في الخاصية SelectCommand، فلست مجبرا في هذه الحالة على إنشاء أوامر الحذف والإدراج والتحديث بنفسك، فمهيئ البيانات يستطيع إنتاج هذه الأوامر آليا عند استدعاء الوسيلة Update الخاصة به، وهو يستخدم لفعل هذا فئات بناء الأوامر CommandBuilders التي سنتعرف عليها لاحقا في هذا الفصل.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

فئة مهيئ البيانات DataAdapter Class

هذه الفئة تمثل الواجهة IDataAdapter، كما أنها ترث الفئة Component.
ولا يمكنك تعريف نسخة جديدة من هذه الفئة، لأن حدث إنشائها Constructor محمي Protected، لكن يمكنك أن تتعامل مع الفئات الفرعية المشتقة منها.

وإضافة إلى ما تمثله من خصائص الواجهة IDataAdapter، تمتلك هذه الفئة الخصائص التالية:

قبول التغييرات أثناء الملء AcceptChangesDuringFill:

إذا جعلت قيمة هذه الخاصية True (وهي القيمة الافتراضية)، فسيتم استدعاء الوسيلة DataRow.AcceptChanges بعد إضافة كل صف إلى مجموعة البيانات، وبهذا ستعتبر مجموعة البيانات أن السجل الذي تمت إضافته لم يحدث به أي تغيير عن السجل الموجود في قاعدة البيانات.. أما لو جعلت قيمة هذه الخاصية False، فستعتبر مجموعة البيانات أن السجل المضاف إليها هو سجل جديد لم يدرج بعد في قاعدة البيانات، وعند إجراء عملية التحديث سيحاول مهيئ البيانات إضافته إلى قاعدة البيانات كسجل جديد، وهو أمر غير مرغوب فيه بالطبع، إلا في بعض الحالات الخاصة، كأن تقوم بتحميل السجلات من أحد الجداول، وتريد حفظها في جدول مؤقت في نفس قاعدة البيانا، أو جدول آخر في قاعدة بيانات أخرى.. لاحظ أنك في هذه الحالة ستفعل ما يلي:

١- تجعل للخاصية AcceptChangesDuringFill لمهيئ البيانات الأول القيمة False.

٢- نستخدم الوسيلة Fill الخاصة بمهيئ البيانات الأول لملء مجموعة البيانات بالسجلات.

٣- تسخدم مهيئ بيانات آخر لتحديث الجدول الجديد باستدعاء الوسيلة Update، مع استخدام أوامر التحديث والإدراج والحذف المناسبة للتعامل مع هذا الجدول الجديد.

والمشروع CopyAuthors يستخدم هذه الطريقة لنسخ المؤلفين من قاعدة بيانات آكسيس وإضافتهم إلى قاعدة بيانات سيكويل سيرفر.. لاحظ أن الوسيلة Update لن تحتاج إلى أوامر التحديد والحذف والتحديث في حالتنا هذه، لهذا لن نعرفها، ولن يحدث خطأ في البرنامج.

قبول التغييرات أثناء التحديث AcceptChangesDuringUpdate:

إذا جعلت قيمة هذه الخاصية True (وهي القيمة الافتراضية)، فسيتم استدعاء الوسيلة DataRow.AcceptChanges بعد نقل التغييرات من كل صف في في مجموعة البيانات إلى قاعدة البيانات، وبهذا تعتبر مجموعة البيانات هذا الصف صفا عاديا لم يحدث به أي تغيير عن السجل الموجود في قاعدة البيانات.. أما لو جعلت قيمة هذه الخاصية False، فستظل مجموعة البيانات تعتبر هذا الصف مختلفا عن الصف الأصلي في قاعدة البيانات، وعند إجراء عملية التحديث مرة أخرى سيعاد تحديثه في قاعدة البيانات، وهو أمر غير مرغوب فيه في معظم الحالات، إلا إذا كنت تريد استخدام نفس مجموعة البيانات لتحديث أكثر من جدول.. مثلا: إذا أردت تحديث جدول المؤلفين في كل من قاعدة بيانات آكسيس وقاعدة بيانات سيكويل سيرفر، فيجب عليك فعل ما يلي:

- وضع القيمة False في الخاصية AcceptChangesDuringUpdate لمهيئ بيانات آكسيس واستدعاء الوسيلة Update.
- وضع القيمة True في الخاصية AcceptChangesDuringUpdate لمهيئ بيانات سيكويل سيرفر واستدعاء الوسيلة Update.. هذا سيجعل مجموعة البيانات تقبل التغييرات، وتعتبر أن كل سجلاتها مطابقة للسجلات الاصلية، فقد استخدمنا هذه التغييرات فعلا ولم نعد نحتاجها.

والمشروع UpdateAll يريك هذه الطريقة عمليا، فهو يحمل سجلات المؤلفين من قاعدة بيانات سيكويل سيرفر، ويملأ بها مجموعة البيانات، ويعرضها للمستخدم في جدول، حيث يستطيع المستخدم إجراء التغيرات التي يريدها على بيانات المؤلفين، أو يضيف أو يحذف بعض المؤلفين، وعندما يضغط زر التحديث، يتم حفظ هذه التغييرات في قاعدتي آكسيس وسيكويل سيرفر معا، وبهذا نضمن أن يظلا متزامنتين دائما.

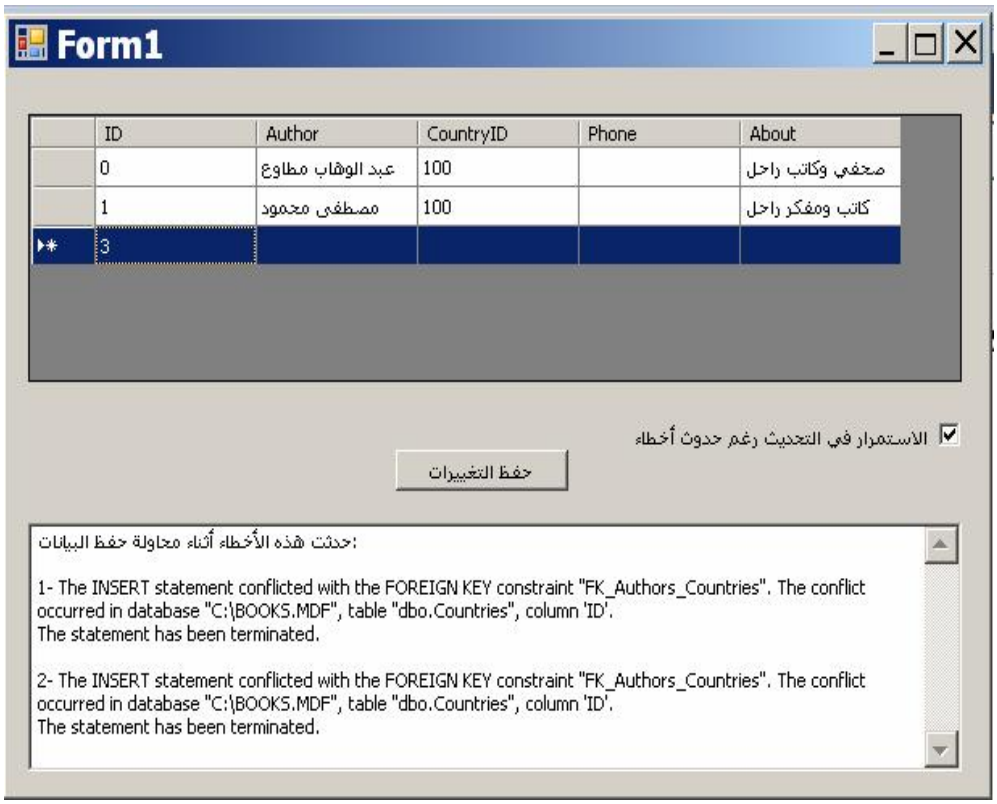
لاحظ أننا في هذا المشروع استخدمنا المعالج السحري Wizard لإنتاج مهيئ البيانات وأوامره، لهذا لن تجد الكثير من الكود في المشروع.. وسنتعرف على هذا المعالج بعد قليل.. ومن المهم أن تلاحظ أن أمر التحديث UPDATE يتعرف على السجل المراد تحديثه في قاعدة البيانات من خلال مفتاحه الأساسي (الحقل ID في مثالنا هذا) وقيمه الأصلية كما شرحنا سابقا.. لهذا لو يكن جدول المؤلفين في قاعدة آكسيس يحتوي على بعض المؤلفين الموجودين في قاعدة بيانات سيكويل سيرفر فلن يحدث خطأ، لكن لن يتم تحديث هؤلاء المؤلفين لأنهم غير موجودين أصلا، ولن تتم إضافتهم أيضا.. هذه المشكلة لن تحدث لو أضفت مؤلفين جددا إلى جدول العرض، ففي هذه الحالة سيتم حفظهم في قاعدتي البيانات بشكل صحيح.. لهذا لو أردت أن تضمن أن يعمل هذا البرنامج بصورة دقيقة، فيجب أن تجعل جدول المؤلفين في كلتا القاعدتين متماثلين منذ البداية، لبحافظ عليهما البرنامج هكذا باستمرار.

استمرار التحديث عند الخطأ ContinueUpdateOnError:

إذا جعلت قيمة هذه الخاصية True، فلن يحدث خطأ في البرنامج إذا حدثت مشكلة في تحديث أحد سجلات قاعدة البيانات، وستستمر محاولة تحديث باقي السجلات، مع إرسال تفاصيل المشكلة إلى الخاصية RowError الخاصة بكائن الصف DataRow الذي فشلت عملية تحديثه.

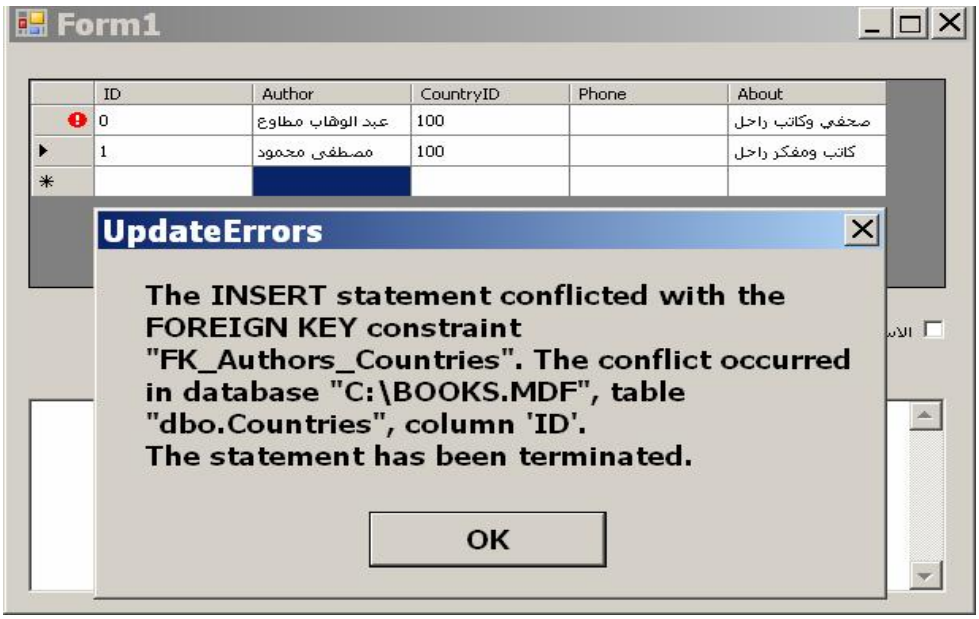
والقيمة الافتراضية لهذه الخاصية هي False، لهذا سيحدث خطأ في البرنامج لو فشل تحديث أحد السجلات، مما سينهي عملية تحديث السجلات في الحال.. وأنت حر في

اختيار الطريقة التي تناسبك أكثر لمعالجة مثل هذه الاخطاء.. والمشروع UpdateErrors يتيح للمستخدم اختيار الطريقة التي تناسبه من خلال مربع الاختيار Check Box الموضوع على النموذج.. فلو اختار "مواصلة التحديث رغم حدوث أخطاء"، فسنعرض له تقريرا بالأخطاء التي حدثت ونطلب منه إصلاحها كما توضح الصورة.



لاحظ أن جدول العرض يمنع معظم الأخطاء، فهو يرفض ترك خانة فارغة إذا كانت لا تقبل القيمة Null، كما يرفض أي نص يتكون من عدد من الحروف أطول مما تقبله الخانة.. لهذا لو أردت اختبار هذا البرنامج، فليس أمامك إلا أن تضع في العمود CountryID رقما أكبر من ٢٢، لأن قيد المفتاح الفرعي FOREIGN KEY constraint بين جدول الدول وجدول المؤلفين، يمنعك من استخدام رقم دولة ليس موجودا في جدول الدول.

أما إن اختار المستخدم إيقاف التحديث عند حدوث أخطاء، فسنستخدم المقطع Try Catch لنعرض له رسالة خطأ عند فشل تحديث أي سجل، ونحدد له هذا السجل في أدوات العرض ليقوم بتصحيحه وإعادة المحاولة.



لاحظ أن السجلات التالية لهذا السجل لم تحفظ حتى الآن، وعند تكرار محاولة الحفظ سيتم حفظها، وقد تظهر أخطاء جديدة في أي سجل منها في تلك اللحظة، حيث يتوجب على المستخدم تصحيحها أيضا وإعادة المحاولة.... وهكذا.. ورغم أن هذا يبدو مملا، إلا أنه أسهل من كتابة تقرير طويل وترك المستخدم يبحث عن السجلات المذكورة في هذا التقرير لتصحيحها.

هذا هو كود زر الحفظ، مع استثناء الكود الذي يجمع أخطاء السجلات في الحالة الأولى، لأننا سنتعرف عليه في فصل لاحق:

```
if (ChkContinue.Checked)
{
   استمرار التحديث رغم حدوث أخطاء //
   DaAuthors.ContinueUpdateOnError = true;
   DaAuthors.Update(Ds);
   علينا جمع الأخطاء التي حدثت وعرضها في تقرير للمستخدم
   // ..............................
}
```

```
else // إيقاف التحديث عند حدوث أي خطأ
{
   DaAuthors.ContinueUpdateOnError = false;
   try
   {
      DaAuthors.Update(Ds);
   }
   catch (SqlException ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

لاحظ أننا لم نكتب أي كود لوضع علامة الخطأ التي تظهر في جدول العرض بجوار السجل الذي سبب المشكلة، فهي تظهر تلقائيا بسبب وجود ربط Binding بين جدول العرض ومجموعة البيانات.. وسنتعرف على تقنية ربط البيانات بالتفصيل في فصل لاحق.

ولعلك تلجأ في برنامجك إلى تعريب أهم رسائل الخطأ التي تتوقع حدوثها، لتعرض للمستخدم العربي رسائل يستطيع فهمها، وكذلك لمنع المستخدم من معرفة أسماء الجداول والأعمدة الموجودة في قاعدة البيانات، والتي قد يستخدمها أي قرصان لمحاولة حقن استعلامات غير مرغوبة وتخريب قاعدة البيانات.. في المثال السابق مثلا، كل ما يهم المستخدم أن يعرفه هو: "لا توجد دولة لها الرقم الذي أدخلته".. طبعا سيحتاج منك الأمر إلى بعض الجهد لكتابة كود يحلل نص الرسالة ويبني النص العربي بناء على أسماء الجداول والأعمدة الموجودة فيها.. لهذا فمن الأذكى أن تقلل احتمالات الخطأ في برنامجك إلى أقصى حد.. ففي البرامج الحقيقية، ليس على المستخدم أن يكتب أرقام الدول، بل عليك أن تعرض له قائمة منسدلة فيها أسماء الدول، وهو يختار منها مباشرة، كما هو موضح في الصورة.. هذا لا يسهل عليه الحياة فقط، بل يمنعه من إحداث أخطاء لا لزوم لها في البرنامج.. وستجد مثالا على هذا في المشروع UpdateErrors2، والذي سيتعذر عليك فيه رؤية أي مثال على أخطاء التحديث.. في هذا المشروع أيضا تخلصنا من رسالة الخطأ العقيم التي يعرضها جدول العرض عند ترك صف فيه أخطاء، وعرضنا رسالة خطأ خاصة بنا

باستخدام الحدث DataGridView.DataError، مما يمنع المستخدم من معرفة أسماء الجداول والأعمدة الموجودة في قاعدة البيانات.

خيار التحميل FillLoadOption:

تحدد هذه الخاصية ماذا سيحدث للنسخة الاصلية Original Version والنسخة الحالية Current Version من السجل عند استخدام الوسيلة Fill لملء مجموعة البيانات.. وتأخذ هذه الخاصية إحدى قيم المرقم LoadOption التالية:

OverwriteChanges	تجاهل التغييرات التي حدثت سابقا للسجل، ووضع القيم القادمة من قارئ البيانات في كل من النسخة الأصلية والنسخة الحالية للسجل.
PreserveChanges	هذه هي القيمة الافتراضية، وهي تحافظ على النسخة الحالية للسجل بدون تغيير، ووضع القيم القادمة من قارئ البيانات في نسخة السجل الأصلية فقط.
Upsert	الاحتفاظ بالنسخة الأصلية للسجل بدون تغيير، ووضع القيم القادمة من قارئ البيانات في نسخة السجل الحالية فقط.

إعادة الأنواع الخاصة بالمزود ReturnProviderSpecificTypes:

إذا جعلت قيمة هذه الخاصية True، فستقوم الوسيلة Fill، باستخدام أنواع البيانات الخاصة بكل مزود (مثل أنواع سيكويل).. والقيمة الافتراضية لهذه الخاصية هي

False، مما يجعل الوسيلة Fill تحول البيانات إلى أنواع البيانات العادية المستخدمة في إطار العمل.

كما تمتلك هذه الفئة الوسائل التالية:

تصفير خيار التحميل ResetFillLoadOption:

تعيد قيمة الخاصية FillLoadOption إلى قيمتها الافتراضية، وتجبر الوسيلة Fill على مراعاة قيمة الخاصية AcceptChangesDuringFill.

حفظ خاصية قبول التغييرات أثناء الملء ShouldSerializeAcceptChangesDuringFill:

إذا جعلت قيمة هذه الخاصية True، فسيتم الاحتفاظ بقيمة الخاصية AcceptChangesDuringFill.

حفظ خاصية خيار التحميل ShouldSerializeFillLoadOption:

إذا جعلت قيمة هذه الخاصية True، فسيتم الاحتفاظ بقيمة الخاصية FillLoadOption.

وتمتلك الفئة DataAdapter الحدث الوحيد التالي:

خطأ الملء FillError:

ينطلق إذا حدث خطأ أثناء ملء مجموعة البيانات.. والمعامل الثاني e لهذا الحدث من النوع FillErrorEventArgs، وله الخصائص التالية:

	DataTable	تعيد كائن جدول البيانات DataTable الذي حدث الخطأ أثناء ملئه.
	Errors	تعيد كائن الاستثناء Exception الذي يحتوي على معلومات الخطأ الذي حدث.

	Values	تعيد مصفوفة كائنات Object Array، تحتوي على القـيم الموجودة بالصف الذي حدث به الخطأ.
	Continue	إذا جعلت قيمتها True، فسيستمر ملء الجدول بالسـجلات وتجاهل الخطأ.. أما إن جعلت قيمتها False فسيتوقف ملء مجموعة البيانات في الحال.

فئة مهيئ بيانات قاعدة البيانات DbDataAdapter Class

هذه الفئة أساسية مجردة تجب وراثتها، وهي ترث الفئة DataAdapter.
وإضافة إلى ما ترثه من الفئة الأم، تمتلك هذه الفئة الخاصيتين التاليتين:

الاسم الافتراضي لجدول المصدر DefaultSourceTableName:

قيمة هذا الثابت هي Table، وهو الاسم الافتراضي الذي يستخدم عند إضافة جدول إلى مجموعة البيانات.

حجم مجموعة أوامر التحديث UpdateBatchSize:

ضع في هذه الخاصية عدد السجلات التي سيقوم مهيئ البيانات بحفظ تغييراتها في قاعدة البيانات في المرة الواحدة.. هذا مفيد لتقيل عدد دورات الاتصال مع الخادم Round Trips أثناء تحديث قاعدة البيانات، مما يجعل أداء البرنامج أفضل.. لكن عليك أن تراعي أن التعامل مع عدد كبير من السجلات في نفس اللحظة قد يؤدي إلى تقليل الكفاءة (استهلاك ذاكرة أكبر، إرسال بيانات أكثر ووقت انتظار أطول)، لهذا عليك اختيار عدد معقول من السجلات يحقق أفضل أداء.
ولا يمكنك استخدام هذه الخاصية مع آكسيس، لأنه لا يسمح بتنفيذ أكثر من أمر في المرة الواحدة.. لكن يمكنك استخدامها مع سكيويل سيرفر وأوراكل.
والجدول التالي يلخص تأثير القيم المختلفة لهذه الخاصية:

١	تحديث سجل واحد في كل مرة.. هذه هي القيمة الافتراضية.
> ١	تحديث العدد المحدد من السجلات، حيث يتم تكوين استعلام لتحديث كل سجل، ودمج هذه الاستعلامات معا بوضع ; بينها.
٠	لا توجد قيود على عدد الأوامر، وهذا معناه استخدام أكبر عدد من الأوامر يمكن لقاعدة البيانات التعامل معه.
< ١	سيحدث خطأ في البرنامج!

وفي حالة استخدام أي قيمة غير الواحد لهذه الخاصية، يجب عليك أن تضع في الخاصية UpdatedRowSource الخاصة بكائن الأمر المستخدم في تنفيذ عملية التحديث القيمة None أو OutputParameters، وإلا حدث خطأ في البرنامج.

كما أن هذه الفئة تضيف عدة صيغ جديدة لكل من الوسيلتين Fill و FillSchema.. دعنا نتعرف على هذه الصيغ:

ملء Fill:

تضيف هذه الفئة ثلاث صيغ جديدة إلى هذه الوسيلة، وهي:

١- الصيغة الأولى تستقبل جدول بيانات DataTable لتملأه بالسجلات.. وقد استخدمنا هذه الصيغة في الوسيلة MyDbConnector.GetTable في المشروع DbTasks لملء كائن جدول DataTable بالبيانات وإعادته للمستخدم.. بعد هذا يمكنك إضافة هذا الجدول إلى مجموعة بيانات، أو عرض بياناته مباشرة في جدول عرض، أو تنفيذ أي عملية تريدها عليه.. وستجد في نفس المشروع مثالا على استخدام الوسيلة GetTable، وذلك بضغط الزر "عرض المؤلفين"، الذي يعرض نموذجا جديدا عليه جدول فيه بيانات المؤلفين.

٢- الصيغة الثانية تستقبل مجموعة البيانات المراد ملئها، ونصا يمثل اسم الجدول المضاف إلى مجموعة البيانات.. الكود التالي مثلا سيضيف جدول الكتب إلى مجموعة البيانات بالاسم TblBooks:

```
DaBooks.Fill(Ds, "TblBooks");
MessageBox.Show(Ds.Tables[0].TableName); //TblBooks
```

٣- الصيغة الثالثة لها أربعة معاملات، هي بالترتيب:

- مجموعة البيانات.
- رقم السجل الذي تريد القراءة بدءا منها، علما بأن أول سجل في الجدول رقمه صفر.

- أقصى عدد من السجلات تريد قراءته من الجدول.. ولن يحدث خطأ إذا كان الجدول يحتوي على عدد من السجلات أقل من هذا العدد.
- نص يمثل اسم الجدول في مجموعة البيانات.

٤- الصيغة الرابعة تفيدك عندما تريد أخذ جزء من السجلات من أمر تحديد يعيد أكثر من جدول، وهي تستقبل ثلاثة معاملات:

- رقم السجل الذي تريد القراءة بدءا منها.
- أقصى عدد من السجلات تريد قراءته من كل جدول.
- مصفوفة معاملات ParamAray تستقبل مصفوفة جداول DataTable Array، ليمكنك أن ترسل إليها الجداول التي تريد ملأها بالسجلات.

وتعيد هذه الوسيلة عدد السجلات التي تمت إضافتها أو تحديثها في مجموعة البيانات.

ملء المخطط FillSchema:

تضيف هذه الفئة صيغتين جديدتين إلى هذه الوسيلة، وهما:

١- الصيغة الأولى تستقبل كائن الجدول DataTable المراد ملؤه، وإحدى قيم المرقم SchemaType التي تعرفنا عليها سابقا.

٢- الصيغة الثانية تستقبل مجموعة البيانات، وإحدى قيم المرقم SchemaType، واسم الجدول في مجموعة البيانات.

وتعيد كلتا الصيغتين كائن الجدول DataTable الذي تم ملؤه بالسجلات.

والفئات التالية ترث الفئة DbDataAdapter:

١. OdbcDataAdapter Class
٢. OleDbDataAdapter Class
٣. SqlDataAdapter Class
٤. OracleDataAdapter Class

وسنكتفي هنا بالتعرف على الفئة SqlDataAdapter.

فئة مهيئ بيانات سيكويل SqlDataAdapter Class

هذه الفئة ترث الفئة DbDataAdapter، وهي تعمل كمهيئ بيانات مخصص للتعامل مع قواعد بيانات سيكويل سيرفر.

ولحدث إنشاء هذه الفئة أربع صيغ:

١- الصيغة الأولى بدون معاملات.

٢- والصيغة الثانية تستقبل أمر التحديد SelectCommand الذي سيستخدمه مهيئ البيانات لملء مجموعة البيانات.

٣- والصيغة الثالثة تستقبل معاملين:

- نص جملة التحديد SELECT.
- كائن الاتصال SqlConnection الذي سيستخدم في الاتصال بقاعدة البيانات.

لاحظ أن مهيئ البيانات سيقوم بإنشاء كائن أمر SqlCommand وسيضع جملة التحديد SELECT في الخاصية CommandText الخاصة به، كما سيضع كائن الاتصال في الخاصية Connection الخاصة به.. بعد هذا سيوضع كائن الأمر في الخاصة SelectCommand الخاصة بمهيئ البيانات.. معنى هذا أن هذه الصيغة تختصر عليك الكثير من الخطوات.

٤- الصيغة الرابعة مماثلة للصيغة السابقة، ولكن معاملها الثاني يستقبل نص الاتصال Connection String اللازم للاتصال بقاعدة البيانات، ليتم استخدامه في إنشاء كائن الاتصال SqlCommand.

وبالإضافة إلى ما ترثه من الفئة الأم من خصائص ووسائل وأحداث، تمتلك هذه الفئة الحدثين التاليين:

يتم تحديث السجل RowUpdating:

عند استدعاء الوسيلة Update الخاصة بمهيئ البيانات، فإنها تقوم بالمرور عبر كل سجل في مجموعة البيانات لاستخدامه في تحديث قاعدة البيانات.. وينطلق هذا الحدث قبل استخدام كل سجل موجود في مجموعة البيانات في عملية التحديث.

والمعامل الثاني e لهذا الحدث من النوع SqlRowUpdatingEventArgs، وهو يمتلك الخصائص التالية:

	Command	تقرأ أو تغير كائن الأمر SqlCommand الذي سيستخدم في عملية التحديث.
	Errors	تعيد كائن الاستثناء Exception الذي يحتوي على معلومات الخطأ الذي حدث في السجل الحالي.. (مفيدة فقط في الحدث RowUpdated).
	Row	تعيد كائن صف البيانات DataRow، الذي يتم التعامل معه حاليا في مجموعة البيانات.
	Statement Type	تعيد نوع جملة الاستعلام التي سيتم تنفيذها، وهي تعيد إحدى قيم المرقم StatementType التالية: - Select: جملة تحديد. - Insert: جملة إدراج. - Update: جملة تحديث. - Delete: جملة حذف. - Batch: استعلام يتكون من مجموعة أوامر.
	Status	تقرأ أو تغير حالة كائن الأمر، وهي تأخذ إحدى قيم المرقم UpdateStatus التالية: - Continue: الاستمرار في تحديث السجل الحالي والسجلات التالية له.

		- ErrorsOccurred: تطلب من مهيئ البيانات التعامل مع عملية التحديث كأنها تسببت في حدوث خطأ.. في هذه الحالة سينطلق خطأ في البرنامج فعلا في السطر الذي استدعيت فيه الوسيلة Update الخاصة بمهيئ البيانات، وعليك معالجة هذا الخطأ بمقطع Try Catch. - SkipCurrentRow: تجاوز السجل الحالي دون استخدامه في تحديث قاعدة البيانات، مع استمرار تحديث باقي السجلات. - SkipAllRemainingRows: تجاوز السجل الحالي والسجلات التالية له وإيقاف عملية التحديث في الحال.
	Table Mapping	تعيد كائن خريطة الجدول DataTableMapping، الذي يستخدم للربط بين الجدول في مجموعة البيانات والجدول الأصلي في قاعدة البيانات.

تم تحديث السجل RowUpdated:

ينطلق هذا الحدث في كل مرة ينتهي فيها مهيئ البيانات من استخدام أحد سجلات مجموعة البيانات في عملية التحديث.

والمعامل الثاني e لهذا الحدث من النوع SqlRowUpdatedEventArgs، وهو يمتلك نفس خصائص الفئة SqlRowUpdatingEventArgs التي تعرفنا عليها في الحدث السابق، ويزيد عليها بالعناصر التالية:

	RecordsAffected	تعيد عدد السجلات التي تأثرت بعملية التحديث في قاعدة البيانات.. هذا العدد يكون صفرا إذا لم يتم العثور على السجل، أو حدث خطأ، ويكون -١ إذا كنت تستخدم جملة SELECT الخاصة بأمر التحديد.. لاحظ أن جملة SELECT

		الموجودة في نهاية أمر التحديث وأمر الإدراج تؤثر على قيمة هذه الخاصية، وتجعل قيمتها صفرا!
	RowCount	هذه الخاصية مفيدة إذا كانت للخاصية UpdateBatchSize الخاصة بمهيئ البيانات قيمة أكبر من ١، ففي هذه الحالة يقوم مهيئ البيانات بتحديث أكثر من سجل في المرة الواحدة، حيث تعيد إليك هذه الخاصية عدد السجلات التي تم التعامل معها.. لاحظ أن هذا العدد قد يكون مساويا لقيمة الخاصية UpdateBatchSize أو أصغر منها (في حالة عدم وجود سجلات كافية في مجموعة البيانات).
	CopyToRows	إذا كانت للخاصية UpdateBatchSize قيمة أكبر من ١، فإن الخاصية e.Row لا تفيدك لمعرفة السجلات التي تم حفظ بياناتها في قاعدة البيانات.. وبدلا منها، يمكنك إرسال مصفوفة صفوف DataRow Array إلى هذه الوسيلة، لتوضع فيها صفوف مجموعة البيانات التي تم استخدامها في عملية التحديث، مع ملاحظة أن هذه العملية مرجعية By Reference، أي أن أي تغيير في الصفوف الموجودة في المصفوفة، سيظهر تأثيره في مجموعة البيانات.. ويجب أن تحتوي المصفوفة المرسلة على الأقل على عدد من الخانات يساوي قيمة الخاصية e.RowCount. وتوجد صيغة أخرى لهذه الوسيلة، لها معامل ثان، يستقبل رقم الخانة، الذي سيتم وضع الصفوف في المصفوفة بدءا منه.

ملحوظة:
لعلك تتساءل لماذا لا يمتلك معامل الحدث RowUpdating الوسيلة CopyToRows.. السبب في هذا أن الحدث RowUpdating ينطلق دائما قبل تحديث كل صف على حدة، حتى لو كان مهيئ البيانات سيستخدم مجموعة أوامر SQL Batch لتحديث مجموعة صفوف دفعة واحدة.. هذا منطقي، لأن مهيئ البيانات يقرأ سجلا تلو سجل من مجموعة البيانات (ويطلق الحدث RowUpdating لكل سجل)، وبعد هذا يكوّن مهيئ البيانات مجموعة أوامر لتحديث السجلات التي قرأها، ويرسل هذه الأوامر المجمعة إلى قاعدة البيانات، ثم يطلق الحدث RowUpdated بعد تنفيذها.

لاحظ أنك لو لم تكتب الإجراء المستجيب للحدث RowUpdated، فإن الوسيلة Update تطلق خطأ في البرنامج إذا حدثت مشكلة تطابق Concurrency Violation عند حفظ أحد السجلات.. أما إذا كتبت الإجراء المستجيب لهذا الحدث، فإن الخطأ لا يحدث، وتتاح لك الفرصة لكتابة الكود الذي يحل مشكلة التطابق.

لكن.. ما هو موضوع التطابق Concurrency هذا؟

هذا هو موضوع الفقرة التالية.

التصارع على تحديث البيانات:

هناك مشكلة رئيسية ستواجهك عند التعامل مع قاعدة بيانات يستخدمها أكثر من موظف في نفس الوقت، وهي التضارب بين التعديرت التي يجريها أكثر من موظف على نفس السجل.. تخيل هذه الحالة:

- قام مستخدم برنامجك بتحميل سجلات الكتب، وقام بتعديل سعر كتاب "عضا الحكيم" من ٥ جنيهات إلى ٧ جنيهات.
- عند محاولة برنامجك حفظ هذه التغييرات في قاعدة البيانات، كان مستخدم آخر قد غير عدد النسخ المتاحة المتاحة من كتاب "عصا الحكيم" من ٢٠٠٠ إلى ٣٠٠٠.

السؤال الآن هو: ماذا نفعل في هذه الحالة؟

لو حفظ برنامجك سجل الكتاب "عضا الحكيم" فسيعدل سعره إلى ٧ جنيهات، لكنه سيعيد عدد النسخ المتاحة منه إلى ٢٠٠٠!

أما لو أبقينا على التعديلات التي أجراها المستخدم الآخر، فهذا معناه الإبقاء على التعديل الذي حدث في عدد النسخ، لكن السعر سيظل ٥ جنيهات!

طبعا في كلتا الحالتين ستحدث مشكلة في العمل.. وعندما سيحاول المدير معاقبة مسئول المخازن في الحالة الأولى فسيقسم له بأغلظ الأيمان إنه غير عدد النسخ المتاحة، وعندما سيحاول معاقبة مسئول المبيعات في الحالة الثانية، فسيقسم له إنه غير ثمن النسخة، وكلاهما صادق في قسمه، وأنت الذي خربت بيته!

تسمّى هذه المشكلة باسم مشكلة التطابق Concurrency Problem.. ويمكن علاجها بأحد الحلين التاليين:

١- التطابق المتشائم Pessimistic Concurrency:

استخدم هذا الحل لو كنت "متشائما" بخصوص تعارض البيانات التي يحفظها المستخدمين، أو كان أي تعارض يمكن أن يؤدي إلى خسائر كبيرة للعمل الذي ينظمه برنامجك، حيث إن التطابق المتشائم يمنع حدوث أي تضارب في البيانات، وذلك بإغلاق Lock سجلات قاعدة البيانات قام أي مستخدم بتحميلها، مما يمنع أي مستخدم

آخر من تغييرها إلى أن يتم يغلق المستخدم الأول الاتصال وتتم إزالة الإغلاق.. ويمكن تنفيذ هذا الحل في دوت نت باستخدام التعاملات Transactions، لهذا سنؤجل تطبيقه إلى الكتاب القادم إن شاء الله.

وعند استخدام هذا الحل، تكون جملة التحديث بسيطة للغاية، لأنك تستخدم المفتاح الأساسي للحقل للعثور عليه في قاعدة البيانات، ومن ثم تغير قيمه مباشرة، لأنك واثق أنه لم يتغير منذ أن قمت بتحميله.. هكذا مثلا ستكون جملة تحديث سجلات المؤلفين:

```
UPDATE Authors
SET Author = @Author,
      CountryID = @CountryID,
      Phone = @Phone,
      About = @About
WHERE ID = @ID;
```

لاحظ أن المعاملات الموجودة في هذا الاستعلام تأخذ قيمها من خانات السجل الذي يتم تحديثه في مجموعة البيانات.

ويعتبر التطابق المتشائم حلا حاسما للمشكلة، لأن أي مستخدم آخر سيحاول تعديل السجلات المتنازع عليها سيحصل على رسالة خطأ تخبره بأنها مغلقة من قبل مستخدم آخر.. وفي هذه الحالة عليك أن تجعل برنامجك ينتظر انتهاء الإغلاق، ومن ثم يعرض للمستخدم السجلات التي يحاول تحديثها، ليتعرف على التغييرات التي تمت عليها، ومن ثم يقرر كيف يوائم بينها وبين التغييرات التي أجراها، ثم يعيد حفظها في قاعدة البيانات.

لكن المشكلة هي أن التطابق المتشائم سيهبط بكفاءة البرنامج إذا استمر إغلاق كل سجل لفترات زمنية طويلة، أو إذا تم تحديث أعداد ضخمة من السجلات على التتابع، وذلك لأن إغلاق السجلات يستهلك جزءا من وقت تشغيل وذاكرة الخادم، كما أنه يحتاج إلى إبقاء قنوات الاتصال مفتوحة مع المستخدمين الذين قاموا بعملية الإغلاق، مما يحرم مستخدمين آخرين من الاتصال بقاعدة البيانات في ذلك الوقت.. لكن يظل التطابق المتشائم الحل الأفضل عند التعامل مع قاعدة بيانات يتصل بها عدد كبير من المستخدمين في نفس اللحظة، ويتصارعون على تحديث نفس السجلات، لأن استخدام

كثرة عمليات التراجع عن التعـــاملات Transactions Rollback لاســـتعادة القـــيم الأصلية قبل التضارب، تستهلك الخادم في هذه الحالة بأكثر مما تفعل عمليات الإغلاق.

٢- التطابق المتفائل Optimistic Concurrency:

هذا هو الحل المفضل والأسهل في تقنية ADO.NET، وقد سمي بهـــذا الاســـم لأنـــه يفترض أن المستخدمين لن يحاولوا تعديل قاعدة البيانات، أثناء تعاملك مع بياناتها فـــي برنامجك، إلا في حالات نادرة.

لكن ماذا يحدث لو حدث التعارض فعلا؟.. كيف نحل المشكلة في هذه الحالة؟

في الحقيقة هناك عدة حلول متبعة، وعليك اختيار لحل الذي يناسبك منهـــا تبعـــا لمـــا يناسبك.. وهذه الحلول هي:

أ. عند تحديث السجل، يتم البحث عنه في قاعدة البيانات بواسطة قيم كل حقولـــه، وليس فقط بواسطة المفتاح الأساسي.. هذه مثلا جملة الاستعلام الخاصة بتحديث سجلات المؤلفين:

```
UPDATE Authors
SET Author = @Author,
      CountryID = @CountryID,
      Phone = @Phone,
      About = @About
WHERE (ID = @Original_ID) AND
      (Author = @Original_Author) AND
      (CountryID = @Original_CountryID) AND
      (@IsNull_Phone = 1) AND (Phone IS NULL) OR
      (ID = @Original_ID) AND
      (Author = @Original_Author) AND
      (CountryID = @Original_CountryID) AND
      (Phone = @Original_Phone)
```

هذا الاستعلام يضمن لك أنه لو حدث أي تغيير في السجل من قبل مســـتخدمين آخرين، فلن يعثر عليه برنامجك، وبالتالي لن يتم حفظ التغييرات التي قام بهـــا مستخدم برنامجك.

لعلك تلاحظ في هذا الاستعلام وجود معاملين للتعامل مع كل حقل.. مثلا، يتم التعامل مع حقل المؤلفين من خلال المعاملين @Author و @Original_Author.. فما هو الفارق بينهما؟

لكي تفهم هذا الفارق، عليك أن تعرف أن مجموعة البيانات DataSet تحتفظ بنسختين من كل سجل يتم وضعه فيها:

- النسخة الأصلية Original Version:

وهي تحتوي على بيانات السجل الأصلي كما تم تحميلها من قاعدة البيانات، لاستخدامها بعد ذلك في البحث عن السجل الأصلي في قاعدة البيانات لتحديثه.

- النسخة الحالية Current Version:

وهي تحتوي على بيانات السجل بعد التغييرات التي أجراها المستخدم عليها، وذلك لاستخدامها في تحديث السجل الأصلي في قاعدة البيانات.

وكل ما يفعله استعلام التحديث السابق، هو تعريف معاملين لكل حقل، أحدهما يقرأ قيمته الحالية (مثل @Author) ويتم استخدامه لحفظ التغييرات في قاعدة البيانات، والآخر يقرأ قيمته الأصلية (مثل @Original_Author) ويستخدم للبحث عن السجل الأصلي في قاعدة البيانات.. ويتم التفريق بين هذين المعاملين باستخدام الخاصية SourceVersion الخاصة بكائن المعامل DataParameter، والذي يمكن إرسال قيمته من خلال المعامل التاسع لحدث الإنشاء New.. هكذا مثلا يتم تعريف المعامل @Author.. لاحظ أننا لن نرسل قيمة المعامل SourceVersion، لهذا سيتم سيقرأ القيمة الحالية افتراضيا:

```
SqlParameter P2 = new SqlParameter("@Author",
    SqlDbType.NVarChar, 0, "Author");
```

وهكذا يتم تعريف المعامل @Original_Author:

```
var P2 = new SqlParameter("@Original_Author",
      SqlDbType.NVarChar, 0,
      ParameterDirection.Input, false, 0, 0,
      "Author", DataRowVersion.Original, null);
```

لكن.. لماذا لا يوجد شرط على الحقل About في المقطع Where؟

السبب في هذا أننا عرفنا هذا الحقل من النوع nvarchar(MAX)، وهذا معناه أنه يتسع لنص قد يصل إلى ٢ مليار حرف، وهذا حجم هائل، وستكون مقارنة هذا الحقل مضيعة للوقت.. لكن لو كنت مصرا، فيمكنك تعديل الاستعلام.. لا أنصحك بفعل هذا من نافذة الخصائص، لأنها ستعجز عن إنشاء المعامل @Original_About بشكل صحيح، وبلا من هذا يمكنك إضافة هذا الكود في بداية حدث تحميل النموذج:

```
// إضافة شرط إلى نهاية استعلام التحديث
//سيحدث خطأ لو كان هناك استعلام تحديد في نهاية استعلام التحديث
DaAuthors.UpdateCommand.CommandText +=
            " And About = @Original_About";
// تعريف معامل جديد
var P = new SqlParameter("@Original_About",
      SqlDbType.NVarChar, -1,
      ParameterDirection.Input, false, 0, 0, "About",
      DataRowVersion.Original, null);
// إضافة المعامل إلى مجموعة معاملات أمر التحديث
DaAuthors.UpdateCommand.Parameters.Add(P);
```

أو يمكنك فتح ملف تصميم النموذج Form1_Designer.cs، وتعديل استعلام التحديث مباشرة، وإن كنت لا أنصح بهذا.

لاحظ أن من الأفضل تغيير نوع الحقل About ليكون أكثر ملاءمة لوظيفته.. يمكنك افتراض أن أطول نبذة لا تزيد عن ٥٠٠ حرف مثلا، وتعريف هذا الحقل منن النوع nvarchar(50).

دعنا نعد إلى استعلام التحديث السابق، فمازال هناك نوع ثالث من المعاملات لم نتطرق إليه.. هذا المعامل مخصص للتعامل مع القيم المنعدمة NULL (مثل المعامل @IsNull_Phone).. وسبب احتياجنا إلى هذا المعامل، هو أن أي عملية مقارنة مع خانة منعدمة تعطي دائما False، لهذا لو كانت خانة الهاتف فارغة في قاعدة البيانات، وكانت فارغة أيضا في النسخة الأصلية من السجل، فإن مقارنتهما ستعطي False، وهذا يعني أن برنامجك لن يستطيع تحديث خانة الهاتف أبدا!

لحل هذه المشكلة، نستخدم الشرط التالي:

```
(@IsNull_Phone = 1) AND (Phone IS NULL)
OR (ID = @Original_ID)
```

هذا الشرط يتأكد من أن خانة الهاتف فارغة في مجموعة البيانات، وأنها فارغة أيضا في قاعدة البيانات، أو أن الخانتين فيهما قيمتان متساويتان.

ويتم تعريف المعامل @IsNull_Phone) بوضع القيمة True في الخاصية SourceColumnNullMapping الخاصة بكائن المعامل، وهو ما يمكن فعله بإرسال القيمة True إلى المعامل التاسع في إحدى صيغ حدث الإنشاء New كالتالي:

```
var P3 = new SqlParameter("@IsNull_Phone",
    SqlDbType.Int, 0, ParameterDirection.Input,
    0, 0, "Phone", DataRowVersion.Original,
    true, null, "", "", "");
```

لاحظ أن مهيئ البيانات يستخدم استعلام التحديث السابق بصورة افتراضية، لكن هذا قد يهبط بكفاءة برنامجك، إذا كان الجدول يحتوي على عدد كبير من السجلات، مما يعقد استعلام التحديث، ويستهلك وقتا ملموسا من سيكويل سيرفر للبحث عن السجل في قاعدة البيانات، لأنه سيقارن هنا كل الخانات، وليس من المتوقع وجود فهارس لكل أعمدة قاعدة البيانات.

ب. عند تحديث السجل، يتم البحث عنه في قاعدة البيانات بواسطة مفتاحه الأساسي فقط (كما فعلنا في التطابق المتشائم).. ميزة هذه الطريقة أنها تبسط استعلام

التحديث، وتجعل العثور على السجل في قاعدة البيانات أسرع لأن المفتاح الأساسي مفهرس Indexed، وهي ميزة هائلة في قواعد البيانات الضخمة.. لكن عيب هذه الطريقة هي أنها تستخدم مبدأ "آخر تحديث يكسب!".. حيث إنّ السجلات يتمّ حفظها إلى قاعدة البيانات، حتّى ولو كانت هناك تعديلات قد أجراها مستخدم آخر عليها.. إنّك تفرض سجلاتك على قاعدة البيانات رغم أنف الجميع (وهذا سيخرب بيت مدير المخازن عند تعديل سعر كتاب عصا الحكيم).. لكن أحيانا تكون هذه الطريقة مقبولة، كما في أنظمة حجز رحلات الطيران، لأنّ آخر تعديل في مواعيد الحجز هو الأولى بالاعتبار.

ج. عند تحديث السجل، يتم البحث عنه في قاعدة البيانات بواسطة المفتاح الأساسي وطابع الوقت Timestamp.. لفعل هذا عليك إضافة عمود من النوع timestamp إلى الجدول.. هذا العمود يتغير تلقائيا كلما تم تعديل السجل، وبهذا لو كان طابع الوقت الذي تحتفظ به مجموعة البيانات مختلفا عن طابع الوقت الموجود في قاعدة البيانات، فهذا معناه أن السجل قد تغير، وفي هذه الحالة لن يحفظ برنامجك التغييرات في هذا السجل.. لاحظ أن المعالج السحري لمهيئ البيانات ينتج أوامر تحديث تعتمد على طابع الوقت إذا وجده في استعلام التحديد، أما إذا لم يجده، فإنه ينتج أوامر تحديث تقارن كل الحقول كما في الطريقة أ.

لاحظ أن الطريقتين أ و ج هما الأكثر شيوعا، لكن بهما مشكلة كبيرة، وهي أن برنامجك سيكتفي بعرض رسالة خطأ للمستخدم تخبره بأن أحد الصفوف يتعارض مع قاعدة البيانات، دون أن يعرف التغييرات التي حدثت في قاعدة البيانات، ودون أن يستطيع إعادة حفظ السجل، لأن نفس الخطأ سيستمر في الحدوث!!

ولحل هذه المشكلة، عليك استخدام الحدث RowUpdated الخاص بمهيئ البيانات بالطريقة التالية:

- إذا كانت قيمة الخاصية e.RecordsAffected تساوي صفرا، فهذا معناه أن أمر التحديث لم يؤثر على قاعدة البيانات، لأنه لم يجد السجل المطلوب تحديثه، إما لأن مستخدما آخر حذفه أو عدل بياناته.. هذا هو التعارض الذي نبحث عنه.. لاحظ أن وجود جملة SELECT في نهاية أمر التحديث سيجعل الخاصية RecordsAffected تعيد الرقم ٠ دائما.. لهذا إذا أردت أن تستفيد من هذه الخاصية في معرفة إن كان التحديث قد تم أم لا، فعليك أن تزيل جملة التحديد من نهاية أمر التحديث.. وسنعرف كيف نفعل هذا من خلال المعالج السحري لمهيئ البيانات بعد قليل.
- ضع نصا يدل على حدوث خطأ في الخاصية e.Row.RowError، مثل "حدث تعارض مع السجل الأصلي لأن أحد المستخدمين قام بتعديله أو حذفه".. هذا سيجعل أيقونة الخطأ تظهر بجوار السجل في جدول العرض، وعند التحليق فوقها بالفأرة سيظهر تلميح على الشاشة يعرض للمستخدم النص الذي كتبته في هذه الخاصية.
- إذا أردت أن يحدث خطأ في البرنامج في سطر استدعاء الوسيلة Update لتعالجه بطريقتك الخاصة، فضع في الخاصية e.Status القيمة ErrorsOccurred.. أما إذا أردت مواصلة عملية التحديث، فضع فيها القيمة Continue أو SkipCurrentRow.. دعنا نستخدم القيمة الأخيرة.
- يبدو الأمر رائعا حتى الآن، وسيلاحظ المستخدم ظهور أيقونات الخطأ بجوار السجلات التي فشل تحديثها.. لكن المشكلة أن المستخدم لا يعرف التعديل الذي أدخله المستخدمون الآخرون على السجل الأصلي في قاعدة البيانات.. لهذا سنلجأ إلى طريقة مبتكرة، وهي استخدام مهيئ بيانات اسمه DaErrAuthor لتحميل السجل الأصلي مرة أخرى في مجموعة بيانات خاصة اسمها DsErr، وعرضه في جدول عرض آخر اسمه DgErrors، ليقارن المستخدم بين البيانات التي يريد حفظها، والبيانات التي حفظها مستخدم آخر، ويتخذ قراره بناء على هذا، كما هو موضح في الصورة.

لاحظ أن السجلات التي حذفها مستخدم آخر من قاعدة البيانات لن تظهر في جدول السجلات المعدلة.. من السهل أن يفهم المستخدم أن السجل قد حذف إذا لم يجده، لكننا أيضا نستطيع التسهيل عليه، بتغيير رسالة الخطأ إذا كان استعلام التحديث لا يعيد أية سجلات، وذلك كالتالي:

```
if (DaErrAuthor.Fill(DsErr, "Authors") > 0)
   e.Row.RowError = "حدث تعارض مع السجل الأصلي" +
                    "لأن أحد المستخدمين قام بتعديله أو حذفه ";
else
   e.Row.RowError = "هذا السجل حذفه مستخدم" +
                    "آخر من قاعدة البيانات ";
```

- بقيت أمامنا خطوة أخيرة، وهي: كيف نسمح للمستخدم بتعديل السجل المعدل أو إعادة السجل المحذوف، إن قرر هذا؟.. أنا أرى أن أفضل حل، هو عرض قائمة موضعية حينما يضغط الصف الذي به خطأ في الجدول العلوي، ومن هذه القائمة يختار ما يناسبه مما يلي:

أ. الأمر "أريد حفظ تعديلاتي":

سنستخدم هذا الأمر عندما يغير مستخدم آخر السجل، وهو ينسخ القيم الأصلية من سجل قاعدة البيانات المعدل ويجعلها القيم الأصلية للسجل الخاص بالمستخدم، وهذا حتى ينجح استعلام التحديث في العثور على السجل الأصلي في قاعدة البيانات، ومن ثم لو ضغط المستخدم زر الحفظ يتم حفظ تعديلاته.. والمستخدم هو المسئول عن نقل أية قيمة يدويا من السجل الأصلي المعروض في جدول العرض السفلي إلى السجل الخاص به قبل ضغط زر الحفظ.. والسجل الذي تنجح محاولة حفظه مرة أخرى، علينا أن نزيل سجل الخطأ المناظر له من جدول العرض السفلي.

لاحظ أنك في البرامج العملية قد تحتاج إلى مراعاة أولويات المستخدمين في إجراء التغيير.. مثلا: لو كان المدير هو من قام بتعديل السجل، فسيستشيط غضبا لو قام أحد الموظفين بإلغاء تعديله!.. لهذا قد تحتاج إلى إضافة حقل اسمه UserID إلى الجدول، ليربطه بجدول المستخدمين Users، بحيث تضع رقم المستخدم الذي أجرى آخر تعديل، وعند حدوث التعارض في البرنامج، لا تسمح للمستخدم باتخاذ قرار حفظ تعديلاته إلا إذا كان المستخدم الآخر أقل أولوية منه أو على الأقل له نفس الأولوية في إجراء التعديلات.. ويمكنك معرفة أولويات المستخدمين من الجدول User، الذي لا بد أن يحتوي على عمود يوضح وظيفة المستخدم، أو عمود يوضح ترتيبه في السلم الوظيفي أو مدى صلاحياته.

ب. إلغاء تعديلاتي:

سنستخدم هذا الأمر عندما يغير مستخدم آخر السجل.. هذا الأمر مفيد عندما يقرر مستخدم البرنامج إلغاء تعديلاته هو، وكل ما سنفعله هو نقل السجل المعدل إلى مجموعة البيانات ليحل محل السجل الذي عدله مستخدم

البرنامج، مما يلغي تعديلاته، ويحافظ على التعديل القادم من قاعدة البيانات.

ج. الأمر "أريد إعادة إدراج السجل المحذوف":

سنستخدم هذا الأمر إذا حذف مستخدم آخر السجل من قاعدة البيانات.. وكل ما يفعله هذا الأمر، هو تغيير حالة السجل الحالي إلى Added، ليعتبره أمر التحديث سجلا جديدا ويضيفه إلى قاعدة البيانات.

د. الأمر "إزالة السجل المحذوف":

سنستخدم هذا الأمر إذا حذف مستخدم آخر السجل من قاعدة البيانات.. وكل ما يفعله هذا الأمر، هو حذف السجل الخاص بالمستخدم من مجموعة البيانات.

وستجد الكود الكامل الذي ينفذ كل هذه الأفكار في المشروع OptimisticConcurrency.. وتوجد نسخة أخرى منه في المشروع OptimisticConcurrencyWithTimeStamp، نستخدم فيها طابع الوقت، حيث عرفنا عمودا اسمه RowVersion في جدول المؤلفين نوعه Timstamp.. لاحظ أن عرض طابع الوقت في جدول العرض DatagridView يسبب أخطاء لأنه يحاول رسم طابع الوقت باعتباره صورة!.. ولحل المشكلة، عليك إخفاء عمود طابع الوقت، فلا يوجد مبرر أصلا لعرضه للمستخدم!.. لفعل هذا استخدمنا الجملة التالية:

DgAuthors.Columns["RowVersion"].Visible = false;

لاحظ أن هناك مشكلة ستواجهنا في هذا البرنامج، بسبب عدم استخدامنا جملة تحديد Select بعد جملة التحديث Update، وذلك لأن طابع الوقت يتغير في قاعدة البيانات باستمرار بعد كل عملية تحديث، ولو لم ننعش مجموعة البيانات بالقيم الجديدة له، فستحدث مشكلة تطابق بلا داع.. ولحل هذه المشكلة، استخدمنا مهيئ بيانات اسمه DaTimestamp، مهمته الحصول على السجل الذي تم تحديثه، ووضعه في مجموعة البيانات لإنعاشها.. لهذا يستخدم أمر التحديد الخاص بهذا المهيئ جملة التحديد التالية:

```
Select * From Authors
Where ID = @ID
```

وأنسب مكان لاستخدام هذا المهيئ، هو الحدث RowUpdated، لأنه ينطلق مباشرة بعد تحديث الصف، لهذا يمكننا أن نقرأ الصف مرة أخرى بعد أن غيرت قاعدة البيانات طابع الوقت الخاص به.. لهذا طورنا جملة الشرط التي نستخدمها في هذا الحدث، بإضافة المقطع Else كالتالي:

```
if (e.RecordsAffected == 0)
{
   الكود المناسب لحل مشكلة التطابق //
}
else if (e.StatementType != StatementType.Delete)
{
   TimestampCmd.Parameters[0].Value = e.Row["ID"];
   DaTimestamp.Fill(Ds, "Authors");
}
```

لاحظ أننا استخدمنا شرطا لاستثناء حالة حذف سجل، فالسجل سيحذف من مجموعة البيانات كما حذف من قاعدة البيانات، وليست لدينا مشكلة.

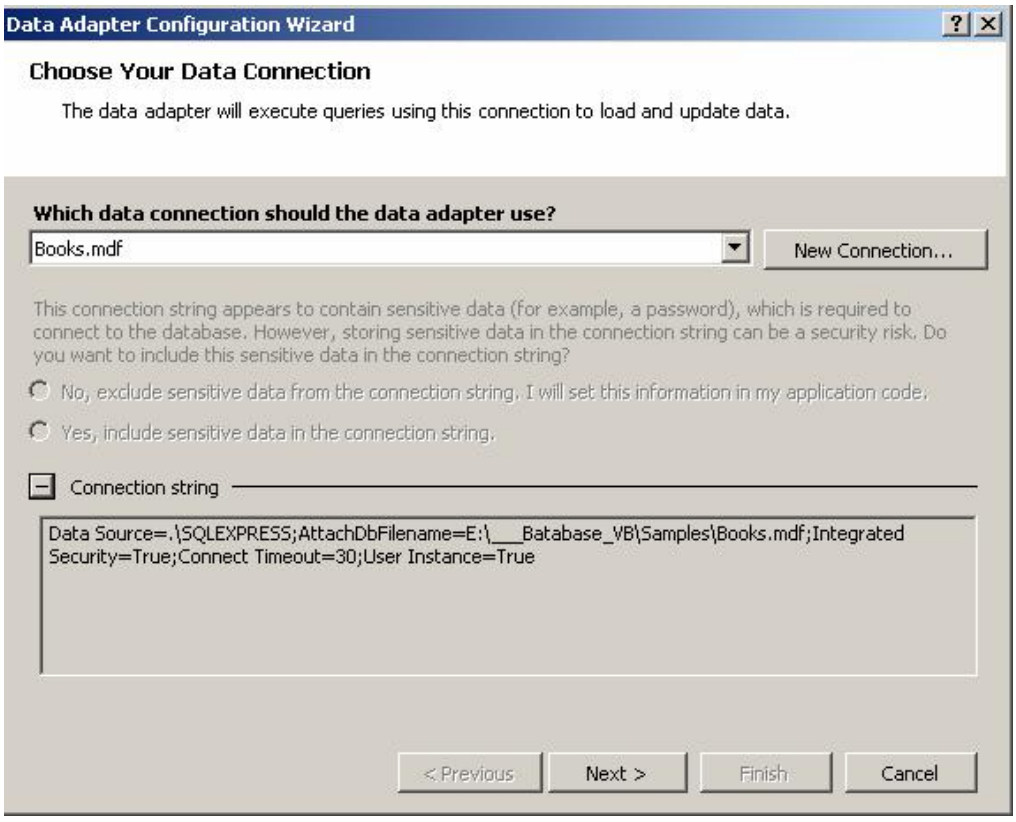
غير هذا لن تجد أي اختلاف في كود هذا المشروع عن المشروع OptimisticConcurrency، فالفروق كلها تنحصر في صيغة استعلامات التحديث، التي تزيد كفاءة برنامجك بسبب استخدامها لطابع الوقت، بديلا عن مقارنة كل القيم الموجودة في خانات الصف.

المعالج السحري لإعداد مهيئ البيانات
Data Adapter Configuration Wizard

استخدم هذا المعالج لتسهيل ضبط وظيفة وخصائص مهيئ البيانات.. ويمكنك تشغيل هذا المعالج باتباع أي مما يلي:

- النقر مرتين على أيقونة مهيئ البيانات SqlDataAdapter في صندوق الأدوات ToolBox.
- سحب أيقونة مهيئ البيانات من صندوق الأدوات وإلقائها على النموذج.
- ضغط مهيئ البيانات بعد إضافته إلى صينية مكونات النموذج Component Tray بزرّ الفأرة الأيمن، ومن القائمة الموضعيّة ضغط الأمر Configure Data Adapter.

ويبدأ هذا المعالج بنافذة تطلب منك اختيار قاعدة البيانات التي تريد الاتصال بها:



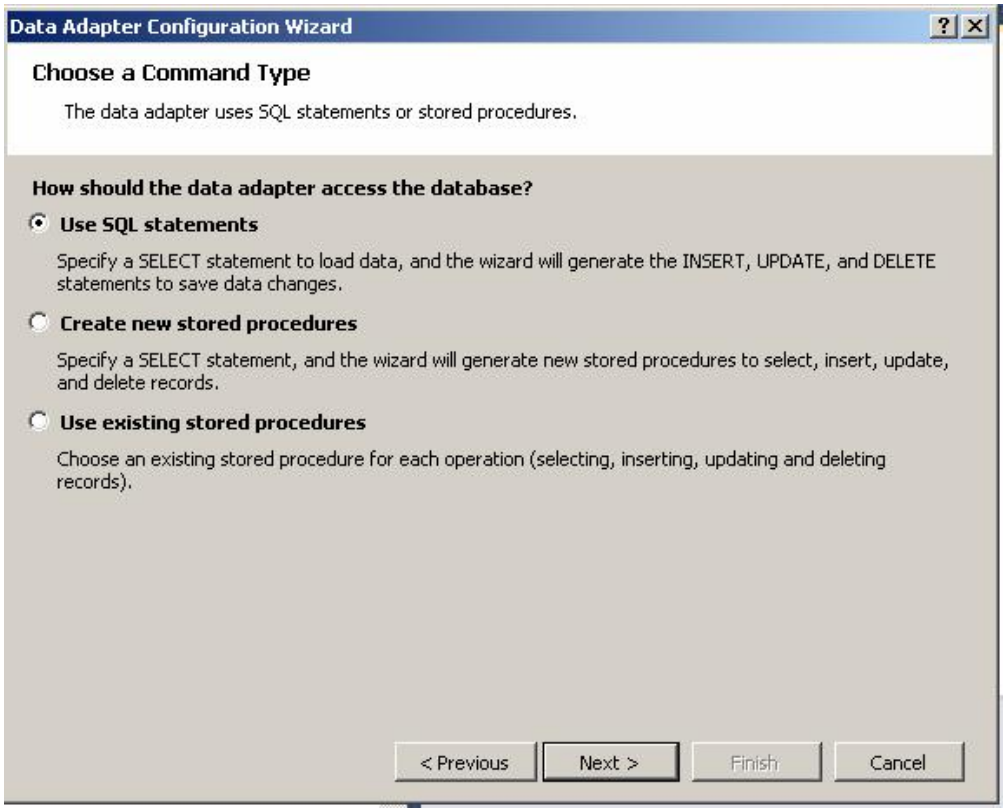
في هذه النافذة، يمكنك استخدام القائمة المنسدلة لاختيار اسم إحدى قواعد بيانات سيكويل سيرفر التي أضفت اتصالا بها من قبل في متصفح الخوادم Server Explorer.

ولو أردت إنشاء اتصال جديد بقاعدة بيانات أخرى، فاضغط الزرّ New Connection لتظهر لك نافذة إضافة اتصال Add Connection التي تعرفنا عليها من قبل في متصفح الخوادم.

ولو ضغطت العلامة + المجاورة للجملة Connection String في الجزء السفلي من النافذة، فسيتم عرض مربع نص قابل للقراءة فقط، به نص الاتصال بقاعدة البيانات التي اخترتها.. ويمكنك نسخ هذا النص لاستخدامه في أي موضع آخر في البرنامج لو أردت.

جرب على سبيل المثال اختيار قاعدة بيانات الكتب Books.mdf، واضغط الزر Next.

في النافذة التالية يمكنك اختيار نوع الاستعلام الذي ستستخدمه لإحضار البيانات من قاعدة البيانات:

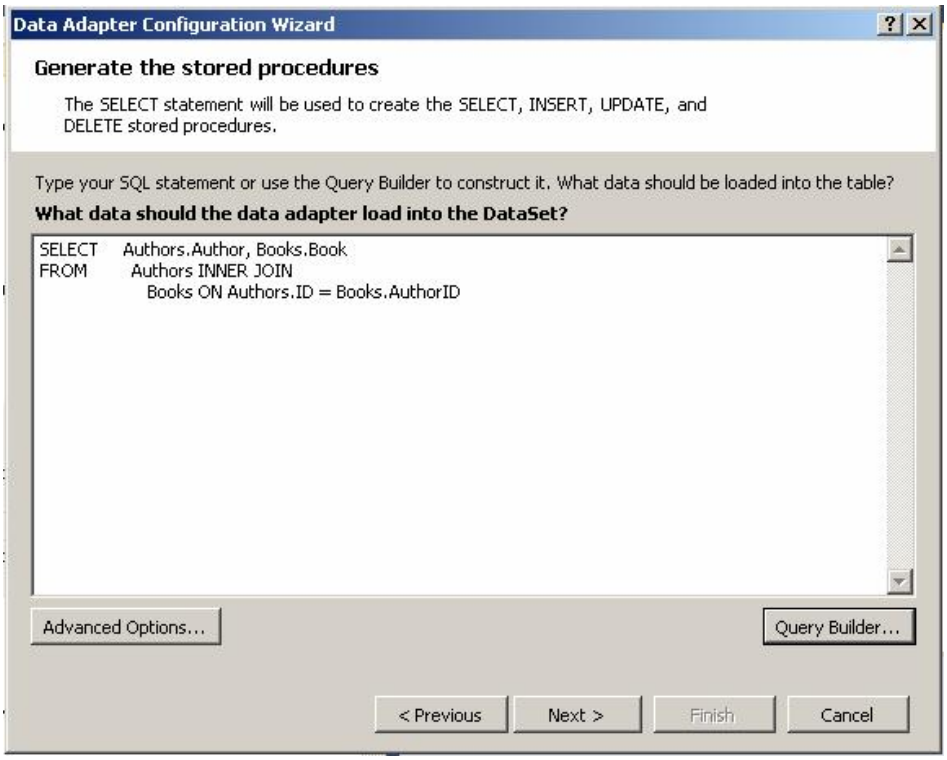


هذه النافذة تتيح لك اختيار واحد مما يلي:

- استخدام جمل SQL (Use SQL Statements).
- إنشاء إجراءات مخزّنة جديدة (Create New Stored Procedures.).
- استخدام إجراءات مخزّنة موجود سابقا في قاعدة البيانات (Use Existing Stored Procedures).

ويحدد اختيارك، النافذة التالية التي ستظهر عندما تضغط الزر Next، وذلك كالتالي:

- إذا اخترت استخدام جملة SQL أو إنشاء إجراء مخزن جديد، فستظهر لك النافذة التالية:



هذه النافذة تقدم لك مربع نص تستطيع أن تضيف فيه يدويا أو باللصق، نص جملة الاستعلام أو جملة إنشاء الإجراء المخزن الجديد.

ويمكنك ضغط الزرّ Query Builder لاستخدام باني الاستعلام في إنشاء جملة SQL، وعند إغلاق باني الاستعلام ستجد هذه الجملة مضافة إلى مربع النص..

لاحظ أن هذا الزر موجود أيضا في حالة إنشاء إجراء مخزن جديد، لأنك قد تحتاج إلى إضافة جملة استعلام داخل الإجراء المخزن، لهذا من الأسهل أن تنشئ هذه الجملة بباني الاستعلام، ثم تضيف إلى مربع النص بعد هذا صيغة الإجراء المخزن الذي يحتويها.. وإن كنت أنصحك بعدم إنشاء الإجراءات المخزنة بهذه الطريقة، لأن إنشاء الإجراء المخزن باستخدام متصفح الخوادم Sever Explorer أسهل وأفضل تنسيقا، ويتيح لك ايضا استخدام باني الاستعلام، مع مراجعة صياغة جمل الإجراء المخزن، واختبار نتائجه.

ملحوظة:

لا يسمح المعالج السحري بكتابة أكثر من جملة SQL مفصولة بالعلامة ; جرب مثلا استخدام الجملة التالية لملء مجموعة البيانات بجدولي المؤلفين والكتب كاملين:

SELECT * FROM Authors;
SELECT * FROM Books

لو ضغطت الزر Next فستظهر نافذة تخبرك بوجود خطأ في جملة الاستعلام، وسترفض مواصلة الخطوات ما لم تصحح هذا الخطأ.

لكن لو كنت مصرا على وضع أكثر من جملة استعلام في مهيئ البيانات، ليقوم بملء مجموعة البيانات بأكثر من جدول، فاتبع الخطوات التالية:

- حدد مهيئ البيانات في صينية مكونات النموذج، واضغط F4 لعرض نافذة الخصائص.
- حدد الخاصية SelectCommand في نافذة الخصائص، واضغط العلامة + المجاورة لها لإسدال خصائص كائن الأمر.
- حدد الخاصية CommandText، واكتب في قيمتها جملة الاستعلام المكونة من أكثر من أمر.
- اضغط زر الحفظ أو انتقل إلى أي خاصية أخرى أو إلى النموذج.. ستظهر رسالة تسألك إن كنت تريد تحديث معاملات هذا الامر.. اضغط زر الموافقة.

الآن سيكون كل شيء على ما يرام، وسيملأ مهيئ البيانات مجموعة البيانات بجدولي المؤلفين والكتب!
الطريف أنك لو أعدت فتح المعالج السحري فستجد جملة الاستعلام المركبة من أكثر من أمر معروضة في مربع النص، ولكنك ستظل تحصل على خطأ لو حاولت الانتقال إلى الخطوة التالية!

وتحتوي هذه النافذة أيضا على الزرّ "خيارات متقدّمة" Advanced Options، ولو ضغطته فستظهر لك نافذة بها الخيارات التالية:

١. **Generate Insert, Update And Delete Statements:**

استخدم هذا الخيار إذا كان برنامجك سيجري تعديلات في السجلات التي يحملها من قاعدة البيانات.. في هذه الحالة سيتمّ إنتاج جمل التحديث والإدراج والحذف آليا، بالاعتماد على أسماء الجداول والأعمدة الموجودة في جملة Select التي أنشأتها.

٢. **استخدم التطابق المتفائل Use Optimistic Concurrency:**

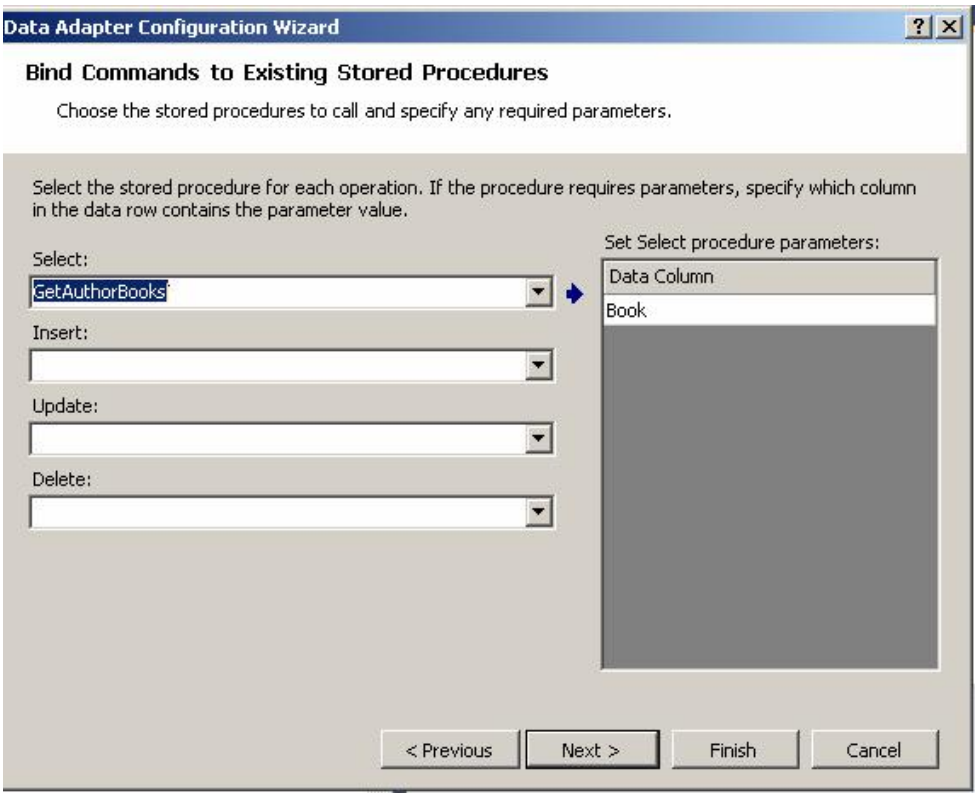
هذا الخيار يتيح لك استخدام التطابق المتفائل عند تحديث البيانات.. لاحظ أن إزالة علامة الاختيار يعني أنك تريد استخدام التطابق المتشائم.. هذا يؤثر فقط على صيغة جملة التحديث UpdateCommand، لكنه لن يكتب لك الكود

المناسب لإغلاق Lock سجلات قاعدة البيانات، وعليك أن تكتب هـذا الكـود بنفسك من خلال كائن التعاملات Transaction Object.

٣. إنعاش مجموعة البيانات Refresh The DataSet:

هذا الخيار يضيف جملة تحديد SELECT بعد جملتـي الإدراج والتحـديث، وذلك لإنعاش مجموعة البيانات بعد تنفيذ أوامر الإدراج والتحديث، كما شرحنا سابقا.. لاحظ أن هذا الخيار غير متاح في قواعد بيانات Access، وذلك لأنـه لا يسمح أصلا بتنفيذ أكثر من استعلام في المرة الواحدة.

- أما لو اخترت استخدام إجراءات مخزنة موجودة سابقا في قاعدة البيانات، فستظهر لك النافذة التالية:



هذه النافذة بها أربع قوائم منسدلة، تعرض كل منها أسماء الإجراءات المخزنة الموجودة في قاعدة البيانات، لتختار منها إجراء للتحديد Select والإدراج Insert والتحديث Update والحذف Delete.

ويوجد على يمين النافذة جدول يعرض بعض التفاصيل الخاصة بالإجراء المخزن.. فبالنسبة للإجراء الخاص بالتحديد، يعرض الجدول أسماء الأعمدة التي يعيد الإجراء المخزن محتوياتها (على سبيل المثال: الإجراء GetAuthorBooks يعيد العمود Book، الذي يحتوي على أسماء الكتب الخاصة بالمؤلف المطلوب.

أما بالنسبة لإجراءات الإدراج والتحديث والحذف، فيعرض هذا الجدول عمودين، أولهما يحتوي على المعاملات المعرفة في الإجراء المخزن، والثاني يحتوي على أسماء حقول مجموعة البيانات، التي سيتم ملء هذه المعاملات من قيمها لإرسالها إلى الإجراء المخزن.. وتحتوي كل خانة في هذا العمود على قائمة منسدلة يمكنك اختيار اسم الحقل منها.

وبعد الانتهاء من أي من النافذتين السابقتين، سيؤدي ضغط الزر Next إلى ظهور نافذة تعرض ملخّصا لخياراتك.. اضغط الزر Finish لإنهاء المعالج السحري وتنفيذ هذه الاختيارات، أو اضغط Cancel لإلغاء العملية.

لاحظ أنك تستطيع في كل نافذة من نوافذ هذا المعالج، الرجوع إلى النافذة السابقة بضغط الزر Previous.

بعد انتهاء المعالج ستجد أن خصائص مهيئ البيانات قد تم ضبطها لتوافق اختياراتك، كما ستجد كائن اتصال SqlConnection اسمه الافتراضي SqlConnection1 قد أضيف إلى صينية مكونات النموذج، ليستخدمه مهيئ البيانات في الاتصال بقاعدة البيانات.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

فئة باني أوامر قاعدة البيانات
DbCommandBuilder Class

هذه الفئة هذه الفئة أساسية مجردة Abstract Base Class، تجب وراثتها MustInherit، وهي ترث فئة المكون Component Class، ويمكنك استخدامها لبناء أوامر التحديث والإدراج والحذف اللازمة لنقل التغييرات من أحد جداول مجموعة البيانات DataSet إلى قاعدة البيانات.. ولكي تفعل هذه الفئة هذا، عليك ربطها بمهيئ البيانات، بشرط أن يحتوي مهيئ البيانات على أمر التحديد SELECT Command.. وتقوم هذه الفئة بالاستجابة للحدث RowUpdating الخاص بمهيئ البيانات، حيث تستخلص المعلومات الأساسية عن تركيب الصف من أمر التحديد SELECT Command (كاسم الجدول وأسماء الأعمدة)، وتبني الأمر المناسب لتحديث أو حذف أو إدراج السجل الذي أطلق الحدث.

ويتم ربط مهيئ بيانات واحد فقط بباني أوامر واحد فقط.

وتمتلك هذه الفئة الخصائص التالية:

مهيئ البيانات DataAdapter:

تقرأ أو تغير مهيئ البيانات الذي سيتم إنشاء أوامره.. لاحظ أن مهيئ البيانات يجب أن يحقق الشروط التالية:

١- أن يحتوي على أمر تحديث SELECT Command.

٢- أن يكون ضمن أعمدة النتيجة التي يعيدا أمر التحديد المفتاح الأساسي أو عمود متفرد Unique يميز كل صف.

٣- أن يعيد أمر التحديد النتائج من جدول واحد فقط.. استعلامات الربط بين أكثر من جدول مرفوضة.

وعند الإخلال بأي من هذه الشروط، سيحدث خطأ ويرفض باني الأوامر إنتاج أوامر التحديث والحذف والإدراج.. ويمكنك استدعاء الوسيلة Dispose الخاصة بباني الأوامر، لإنهاء ارتباطه بمهيئ البيانات، والوقوف عن استخدام الأوامر التي أنشأها.

وضع جميع القيم SetAllValues:

إذا جعلت قيمتها True، ينتج باني الأوامر أمر تحديث Update يحدث جميع قيم السجل.. أما إذا جعلتها False، فسينتج أمر تحديث يحدث فقط قيم الحقول التي تغيرت في مجموعة البيانات.. لاحظ أن باني الأوامر يتابع الحدث RowUpdating الخاص بمهيئ البيانات، يفحص كل صف قبل تحديثه، ومن ثم ينتج استعلام التحديث المناسب لهذا الصف تبعا للتغييرات التي حدثت فيه في مجموعة البيانات.

طبعا هذا أذكى وأسرع وأقل عبئا على خادم سيكويل، لكنه قد يؤدي إلى نتائج غير متوقعة إذا كنت تحل مشاكل التوافق Concurrency Conflicts باستخدام طريقة "الأخير يكسب"، بحيث تحفظ تغييراتك مباشرة، فهذا قد يجعل قيم السجل تتكون من مزيج من تغييراتك وتغييرات مستخدم آخر.. أما إذا كنت تستخدم طابع الوقت أو تقارن كل الحقول، فلا خوف من هذه المشكلة.

خيار التعارض ConflictOption:

تحدد كيف يتم إنشاء أمر التحديث UPDATE والحذف DELETE لتلافي مشاكل التطابق Concurrency Conflicts.. وتأخذ هذه الخاصية إحدى قيم المرقم ConflictOption التالية:

CompareAll SearchableValues	يتم البحث عن السجل المراد تحديثه في قاعدة البيانات، بمقارنة جميع قيم الحقول العددية والنصية الصغيرة، للتأكد من أن السجل لم تدخل عليه أية تعديلات.
Compare RowVersion	يتم البحث عن السجل المراد تحديثه في قاعدة البيانات، بمقارنة حقل الإصدار.. هذا يتطلب وجود حقل من النوع TimeStamp في الجدول.
Overwrite Changes	يتم البحث عن السجل المراد تحديثه في قاعدة البيانات، باستخدام مفتاحه الأساسي فقط، وهذا يعني أن التغييرات الخاصة ببرنامجك سيتم حفظها في السجل لتلغي أية تغييرات أدخلها المستخدمون الآخرون.

قوس الفتح QuotePrefix:

تقرأ أو تغير النص المستخدم كقوس فتح، لاستخدامه مع أسماء الجداول والأعمدة التي تحتوي على مسافات أو حروف غير مقبولة.

قوس الإغلاق QuoteSuffix:

تقرأ أو تغير النص المستخدم كقوس إغلاق.

مع قواعد البيانات المألوفة، يكون قوسا الفتح والإغلاق [ ].

موضع الفهرس CatalogLocation:

تقرأ أو تغير الموضع الذي سيوضع فيه اسم قاعدة البيانات، عند تكوين المسارات الكاملة لأسماء الجداول في أوامر SQL.. وهي تأخذ إحدى قيمتي المرقم CatalogLocation التاليتين:

Start	يوضع اسم قاعدة البيانات في بداية المسار.
End	يوضع اسم قاعدة البيانات في نهاية المسار.

فاصل الفهرس CatalogSeparator:

أو تغير النص المستخدم كفاصل بين اسم قاعدة البيانات واسم الجدول عند كتابة المسار الكامل.. المألوف أن تستخدم النقطة . كفاصل، مثل Books.Author .

فاصل المخطط SchemaSeparator:

تقرأ أو تغير النص المستخدم كفاصل بين اسم المخطط Schema واسم معرف Identifier موجود في هذا المخطط.. والمألوف أن تستخدم النقطتان المتعامدتان : كفاصل، مثل: Person:CustomerName.

كما تملك هذه الفئة الوسائل التالية:

إنعاش المخطط RefreshSchema:

تحذف أوامر التحديث والحذف والإدراج التي قام باني الأوامر ببنائها.. هذا ضروري إذا أردت أن تجعل باني الأوامر ينعش معلوماته عن مهيئ البيانات، حيث إن باني الأوامر يبني أوامره بعد أول عملية تحديث لقاعدة البيانات، ويستخدمها كما هي بعد هذا.. لهذا عليك استدعاء هذه الوسيلة إذا قمت بتغيير استعلام التحديد أو وقت الانتظار CommandTimeout أو كائن التعاملات Transaction الذي يستخدمه أمر التحديد، ليعيد باني الأوامر إنشاء أوامر التحديث والحذف والإدراج لتلائم هذه التغييرات.

معرفة أمر التحديث GetUpdateCommand:

تعيد كائن الأمر DbCommand الذي تم إنتاجه لتحديث قاعدة البيانات.

وهناك صيغة ثانية لهذه الوسيلة، تستقبل معاملا منطقيا، إذا جعلته False، فسيستخدم باني الأوامر في أمر التحديث، معاملات لها الأسماء P1 و P2 و P3 وهكذا... (مثل SET Author = @P1).. وهذه هي الحالة الافتراضية في الصيغة الأولى لهذه الوسيلة.

أما إذا جعلت قيمة هذا المعامل True، فسيتم إنتاج معاملات لها نفس أسماء الأعمدة ، كلما كان هذا ممكنا (مثل SET Author = @Author).. لاحظ أن محاولة إنتاج هذه الأسماء ستسبب خطأ إلا إذا جعلت كائن DbMetaDataColumnNames يلتزم بالشروط التالية:

١- تحديد أقصى طول ممكن لأسماء المعاملات، من خلال الخاصية ParameterNameMaxLength.

٢- توضيح صيغة أسماء المعاملات، من خلال الخاصية ParameterNamePattern.

٣- تحديد تنسيق العلامة المميزة للمعامل، من خلال الخاصية ParameterMarkerFormat.

معرفة أمر الإدراج GetInsertCommand:

تعيد كائن الأمر DbCommand الذي تم إنتاجه لإدراج صف جديد في قاعدة البيانات.. وهي مماثلة في صيغتيها للوسيلة GetUpdateCommand.

معرفة أمر الحذف GetDeleteCommand:

تعيد كائن الأمر DbCommand الذي تم إنتاجه لحذف صف من قاعدة البيانات.. وهي مماثلة في صيغتيها للوسيلة GetUpdateCommand.

تقويس المعرف QuoteIdentifier:

أرسل إلى هذه الوسيلة نصا يمثل مسارا كاملا لأحد عناصر قاعدة البيانات (مثل Books.Author.ID)، لتعيد إليك نفس المسار بعد وضع كل أسماء العناصر الموجودة فيه بين قوسين (مثل [Books].[Author].[ID]).. لاحظ أن المعرف المحاط بقوسين فعلا سيتم تجاهله.

إزالة تقويس المعرف UnquoteIdentifier:

أرسل إلى هذه الوسيلة نصا يمثل مسارا مقوسا لأحد عناصر قاعدة البيانات (مثل [Books].[Author].[ID])، لتعيد إليك نفس المسار بعد إزالة جميع الأقواس منه (مثل Books.Author.ID).

والفئات التالية ترث الفئة DbCommandBuilder:

١. OdbcCommandBuilder Class.
٢. OleDbCommandBuilder Class.
٣. OracleCommandBuilder Class.
٤. SqlCommandBuilder Class.

وسنتعرف هنا فقط على الفئة SqlCommandBuilder.

فئة باني أوامر سيكويل
SqlCommandBuilder Class

هذه الفئة ترث الفئة DbCommandBuilder، وهي تمتلك نفس خصائصها ووسائلها، مع فارق بسيط أنها مخصصة للتعامل مع سيكويل سيرفر وأوامره SqlCommand.
ولحدث إنشاء هذه الفئة صيغتان:

١- الأولى بدون معاملات.

٢- والثانية تستقبل مهيئ البيانات SqlDataAdapter الذي سيرتبط به باني الأوامر.

وتمتلك هذه الفئة الوسيلة الجديدة التالية:

اشتقاق المعاملات DeriveParameters:

أرسل إلى هذه الوسيلة كائن أمر SqlCommand مجهز لتنفيذ إجراء مخزن، لتقوم هذه الوسيلة بالاتصال بقاعدة البيانات، والحصول على معلومات عن معاملات الإجراء المخزن، واستخدامها لإضافة المعاملات المناسبة غلى مجموعة المعاملات Parameters Collection الخاصة بكائن الأمر.. لاحظ أن خطأ سيحدث لو أرسلت إلى هذه الوسيلة كائن أمر يتعامل مع استعلام SQL أو يحتوي على اسم إجراء مخزن غير صحيح.
ويعيب هذه الوسيلة أنها تحتاج إلى الاتصال بقاعدة البيانات مرة إضافية لإحضار بيانات المعاملات، لأنك بالتأكيد ستتصل مرة ثانية لتنفيذ الأمر.

والمشروع CommandBuilder يريك مثالا على استخدام هذه الفئة لإنتاج أوامر التحديث والإدراج والحذف.

خرائط البيانات Data Mapping:

يتيح لك مهيئ البيانات عمل خراط للجداول Table Mapping، وذلك بإعادة تسمية الجداول والأعمدة بأسماء خاصة بك، وربطها بالأسماء الحقيقية في قاعدة البيانات.. هذا يحقق لك الفوائد التالية:

١- تغيير أسماء الجداول أو الأعمدة في مجموعة البيانات كما يناسب برنامجك، دون العبث بقاعدة البيانات الأصلية.
٢- عرض أسماء الأعمدة للمستخدم بالطريقة التي تناسبك.
٣- إذا كان لديك جدولان لهما نفس الاسم في قاعدتي بيانات مختلفتين، فإن بإمكانك إضافتهما إلى نفس مجموعة البيانات، وذلك بتغيير اسميهما من خلال خريطة الجدول الخاصة بكل منهما.

وتنقسم خرائط البيانات إلى نوعين:

١- خريطة الجدول Table Mapping:

حيث تكون لكل جدول خريطة، يذكر فيها اسمه الأصلي في قاعدة البيانات واسمه الجديد في مجموعة البيانات.. وتوضع خرائط الجداول في المجموعة TableMappings في مهيئ البيانات.

وهناك نقطة هامة يجب أن تنتبه إليها عند إنشاء هذه الخرائط، هي ان مهيئ البيانات يعطي أسماء افتراضية للجداول، تختلف عن أسمائها الأصلية (مثل Table و Table1... إلخ).. هذا قد يسبب لك ارتباكا وأنت تنشئ خرائط الجداول، فستبدو لك وظيفتها عكسية، فبدلا من أن تعطي الجدول الأصلي اسما جديدا، ستحاول أن تعيد تسمية الاسم الافتراضي الخاص بمهيئ البيانات باسم الجدول الأصلي!

لكن بقليل من التأمل، ستفهم لماذا يفعل مهيئ البيانات هذا.. فاستعلام التحديد في معظم الحالات لا يعيد جدولا من قاعدة البيانات، بل قد يعيد أجزاء من عدة جداول

(كما في حالة الربط Joining) أو قد يعيد نتائج محسوبة من جدول أو أكثر (كما في حالة التجميع Aggregation).. لهذا يريح مهيئ البيانات نفسه من كل هذه الاحتمالات المعقدة، ويسمي الجداول الناتجة من الاستعلام بأسماء افتراضية، ويترك لك حرية إنشاء خريطة الجداول التي تصحح فيها الأسماء بطريقتك.

والمشروع Mapping يريك مثالا على هذا.. فنحن نستخدم استعلام ربط يعيد المؤلفين وكتبهم.. نتيجة هذا الاستعلام ستحتوي على جدول مخلق، سيعطيه مهيئ البيانات الاسم الافتراضي Table، لهذا استخدمنا خريطة الجدول لإعادة تسميته Authors-Books.

٢- خريطة العمود Column Mapping:

حيث تكون لكل عمود خريطة، يذكر فيها اسمه الأصلي في قاعدة البيانات واسمه الجديد في مجموعة البيانات.. وتوضع خرائط الأعمدة في المجموعة ColumnMappings في خريطة الجدول الذي تنتمي إليه.

ولا يحتاج مهيئ البيانات إلى تسمية الأعمدة بأسماء افتراضية، لسبب بسيط: هو أن كل عمود يتم ذكره صراحة في استعلام التحديد، وحتى الأعمدة المولدة (الأعمدة المحسوبة) يتم تسميتها إجباريا باستخدام الفقرة As، لهذا فإن مهيئ البيانات يعرف يقينا اسم كل عمود في النتيجة.. ولا يتدخل مهيئ البيانات لإعادة تسمية العمود، إلا في حالة وجود عمودين بنفس الاسم (يمكن أن يحدث هذا لو كنت تستخدم أكثر من جملة SELECT في أمر التحديد مثلا).

وأهم استخدام لخرائط الأعمدة، هو إعادة تسمية الأعمدة بطريقة تصلح لعرضها للمستخدم.. والمشروع Mapping يريك مثالا على هذا، حيث استخدمنا خريطة الأعمدة لإعادة تسمية العمود Author بالاسم "المؤلف"، والعمود Books بالاسم "الكتاب".. هذان الاسمان سيظهران في جدول العرض وهذا مناسب للمستخدم العربي للبرنامج.

لاحظ أنك بعد عمل خرائط الربط، ستستخدم اسم الجدول الجديد واسمي العمودين العربيين في الكود عند التعامل معهما من خلال مجموعة البيانات.. مثلا:

```
var T = Ds.Tables["Authors-Books"];
MeesageBox.Show(T.Columns["المؤلف"].MaxLength.ToString( ));
```

في الحقيقة هناك حل آخر لعرض أسماء الأعمدة بأسماء عربية دون استخدام خريطة الأعمدة، وذلك باستخدام خصائص جدول العرض نفسه لإعادة تسمية عنوان العمود، كما سنرى فيها بعد.

والآن، فلنتعرف على خرائط الجداول والأعمدة، وكيفية استخدامها.. وسنتفق هنا على استخدام التعبير "اسم الجدول الأصلي" للإشارة إلى الاسم الذي يمنحه مهيئ البيانات للجدول، مع ملاحظة أن هذا الاسم حساس لحالة الأحرف.. كما سنستخدم التعبير "اسم العمود الأصلي" للإشارة إلى اسم العمود في قاعدة البيانات، أو الاسم البديل الذي سماه به مهيئا البيانات إن حدث تعارض بين عمودين، مع ملاحظة أن هذا الاسم غير حساس لحالة الأحرف.

واجهة مجموعة خرائط الجداول
ITableMappingCollection Interface

هذه الواجهة ترث واجهة القائمة IList، وهي قائمة تحتوي على خرائط الجداول.
وإضافة إلى ما ترثه من واجهة القائمة من عناصر، تمتلك الوسيلة الوحيدة التالية:

معرفة الخريطة من جدول مجموعة البيانات GetByDataSetTable:

أرسل إلى هذه الوسيلة اسم جدول موجود في مجموعة البيانات DataSet، لتعيد إليك كائنا تمثل الواجهة ITableMapping، يحتوي على خريطة هذا الجدول.. ومن المتوقع أن يكون هذا الكائن من نوع الفئة DataTableMapping التي سنتعرف عليها لاحقا.

كما تضيف هذه الواجهة صيغة أخرى لبعض عناصر القائمة التقليدية، مثل:

العنصر Item:

تستقبل الصيغة الثانية لهذه الخاصية اسم الجدول الأصلي (وهو حساس لحالة الأحرف)، وتعيد كائنا Object يحتوي على خريطة هذا الجدول إن وجدت في القائمة، وإن لم توجد فسيحدث خطأ.
كما يمكنك استخدام هذه الخاصية لتغيير كائن خريطة الجدول، فهي قابلة للقراءة وللكتابة أيضا.

إضافة Add:

تستقبل الصيغة الثانية لهذه الوسيلة معاملين نصيين Strings، أولها هو اسم الجدول الأصلي (وهو حساس لحالة الأحرف Case-Sensitive)، وثانيهما هو اسم الجدول في مجموعة البيانات.. وتقوم هذه الوسيلة بإنشاء كائن خريطة جدول DataTableMapping يمثل العلاقة بين الجدولين وتضيفه إلى القائمة، وتعيد نسخة من الواجهة ITableMapping تشير إلى هذا الكائن.

تحتوي على Contains:

تستقبل الصيغة الثانية لهذه الوسيلة اسم الجدول الأصلي، وتعيد True إذا كانت هنـــاك خريطة لهذا الجدول في القائمة.

رقم العنصر IndexOf:

تستقبل الصيغة الثانية لهذه الوسيلة اسم الجدول الأصلي، وتعيد رقم الخانة التي يوجـــد بها كائن خريطة هذا الجدول في القائمة إن وجد، أو تعيد -١ إن لم توجد خريطة لهذا الجدول.

حذف من موضع RemoveAt:

تستقبل الصيغة الثانية لهذه الوسيلة اسم الجدول الأصلي، وتبحث في القائمة عن كـــائن خريطة هذا الجدول، وتحذفه إن وجدته.

ملحوظة:
هذه الصيغة تبدو مختلفة في وظيفتها عن الصيغة الأولى المألوفة، التي تستقبل رقـــم خانة في القائمة وتحذفها لإزالة خريطة الجدول الموجودة بها من القائمة.. وإن شئت رأيي، كان المنطقي أن تكون هذه الصيغة الجديدة هـــي الصـــيغة الثانيـــة للوســـيلة Remove وليس RemoveAt منعا للالتباس!!

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

فئة مجموعة خرائط الجداول
DataTableMappingCollection Class

هذه الفئة تمثل الواجهة ITableMappingCollection، وهي تعمل كقائمة، عناصرها من نوع الفئة DataTableMapping التي سنتعرف عليها بعد قليل.
وإضافة إلى ما ترثه من الواجهة ITableMappingCollection والواجهة IList من خصائص ووسائل، تمتلك هذه المجموعة الوسيلتين الجديدتين التاليتين:

معرفة خريطة الجدول GetTableMappingBySchemaAction:

تبحث في مجموعة الخرائط عن الخريطة التي تربط بين اسم الجدول الأصلي واسم الجدول في مجموعة البيانات، فإن وجدته تعيد كائنا من النوع DataTableMapping يمثل هذه الخريطة.. وتستقبل هذه الوسيلة المعاملات التالية بالترتيب:

- مجموعة خرائط الجداول DataTableMappingCollection التي سيتم البحث فيها.
- اسم الجدول الأصلي.
- اسم الجدول في مجموعة البيانات.
- إحدى قيم المرقم MissingMappingAction التي تحدد ماذا سيحدث إذا لم تكن خريطة الجدول موجودة، كما هو موضح في الجدول التالي:

Passthrough	يتم إنشاء خريطة جدول جديدة، تحمل اسم الجدول الأصلي المرسل في المعامل الأول، واسم جدول مجموعة البيانات المرسل في المعامل الثاني.
Ignore	يتم تجاهل الخطأ، وتعيد الوسيلة Nothing.
Error	يتم إطلاق خطأ من النوع InvalidOperationException.

رقم جدول مجموعة البيانات IndexOfDataSetTable:

أرسل إلى هذه الوسيلة اسم الجدول في مجموعة البيانات، لتعيد إليك رقم خريطة الجدول في مجموعة الخرائط الحالية.. وتعيد هذه الوسيلة -١ إذا لم تعثر على خريطة هذا الجدول.

واجهة خريطة الجدول
ITableMapping Interface

تمتلك هذه الواجهة الخصائص اللازمة لرسم خريطة الربط بين الجدول الأصلي والجـــدول الموجود في مجموعة البيانات.. وهذه الخصائص هي:

جدول المصدر SourceTable:

تقرأ أو تغير اسم الجدول الأصلي.

جدول مجموعة البيانات DataSetTable:

تقرأ أو تغير اسم الجدول في مجموعة البيانات.

خرائط الأعمدة ColumnMappings:

تعيد كائنا يمثل واجهة خـــرائط الأعمـــدة IColumnMappingCollection، وهـــو تحديدا من نوع الفئة ColumnMappingCollection، التي يمكنك أن تضيف إليهـــا خرائط الربط بين الأعمدة.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

فئة خريطة الجدول DataTableMapping Class

هذه الفئة تمثل الواجهة DataTableMapping، وهي تحتوي على المعلومات اللازمة لربط الجدول في مجموعة البيانات بالجدول الأصلي.

ولحدث إنشاء هذه الفئة ثلاث صيغ:

١- الصيغة الأولى بدون معاملات.

٢- والصيغة الثانية تستقبل اسم الجدول الأصلي واسم الجدول في مجموعة البيانات.

٣- والصيغة الثالثة تزيد على الصيغة السابقة بمعامل ثالث، يستقبل مصفوفة من النوع DataColumnMapping، تحتوي على معلومات الربط بين أعمدة الجدولين.

وتمتلك هذه الفئة هذه الوسائل الجديدة:

معرفة عمود البيانات GetDataColumn:

تعيد كائن عمود البيانات DataColumn الذي يمثل العمود المحدد بالمعاملات المرسلة، وهي بالترتيب:

- اسم العمود الأصلي.
- كائن النوع Type الذي يمثل نوع هذا العمود.
- كائن الجدول DataTable الذي يمثل الجدول في مجموعة البيانات.
- إحدى قيم المرقم MissingMappingAction التي تحدد ماذا سيحدث لو لم يتم العثور على العمود في خريطة الجدول.. وقد تعرفنا على قيم هذا المرقم سابقا.
- إحدى قيم المرقم MissingSchemaAction التي تحدد ماذا سيحدث لو لم يتم العثور على العمود في مخطط الجدول Schema، وهذه القيم هي:

Add	يضاف العمود إلى مخطط الجدول.
AddWithKey	يضاف العمود والمفتاح الأساسي Primary Key إلى مخطط الجدول.

Ignore	يتم تجاهل العمود.
Error	يتم إطلاق خطأ من النوع InvalidOperationException.

معرفة الجدول GetDataTableBySchemaAction:

تعيد كائن جدول البيانات DataTable الموجود في مجموعة البيانات، والمذكور اسمه في خريطة الربط.. وتستقبل هذه الوسيلة معاملين:

- كائن مجموعة البيانات DataSet.
- إحدى قيم المرقم MissingSchemaAction، التي توضح التصرف المناسب إذا لم يتم العثور على هذا الجدول في مجموعة البيانات.

معرفة خريطة العمود GetColumnMappingBySchemaAction:

تعيد كائن خريطة العمود DataColumnMapping للعمود الذي تريدجه، وهي تستقبل معاملين:

- اسم العمود الأصلي.
- إحدى قيم المرقم MissingMappingAction الذي تعرفنا عليه من قبل، والتي توضح رد الفعل إذا لم يتم العثور على هذا العمود.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

واجهة مجموعة خريطة العمود
IColumnMappingCollection Interface

هذه الواجهة ترث واجهة القائمة IList، وهي تملك وسيلة وحيدة جديدة، وهي:

معرفة الخريطة بواسطة عمود مجموعة البيانات GetByDataSetColumn:

أرسل إلى هذه الوسيلة اسم العمود في مجموعة البيانات، لتعيد إليك كائنا يمثل واجهة خريطة العمود IColumnMapping الذي يحتوي على معلومات ربط هذا العمود بالعمود الأصلي.. وسيكون هذا الكائن من نوع الفئة DataColumnMapping تحديدا.

وتضيف هذه الواجهة صيغة أخرى لبعض خصائص ووسائل القائمة التقليدية، مثل:

العنصر Item:

تستقبل الصيغة الثانية لهذه الخاصية اسم العمود الأصلي (وهو حساس لحالة الأحرف Case-Sensitive)، وتعيد كائنا Object يحتوي على خريطة هذا العمود إن وجدت في القائمة، وإن لم توجد فسيحدث خطأ.

إضافة Add:

تستقبل الصيغة الثانية لهذه الوسيلة معاملين نصيين Strings، أولها هو اسم العمود الأصلي، وثانيهما هو اسم العمود في مجموعة البيانات.. وتقوم هذه الوسيلة بإنشاء كائن خريطة أعمدة DataColumnMapping يمثل العلاقة بين العمودين وتضيفه إلى القائمة، وتعيد نسخة من الواجهة IColumnMapping تشير إلى هذا الكائن.

تحتوي على Contains:

تستقبل الصيغة الثانية لهذه الوسيلة اسم العمود الأصلي (وهو حساس لحالة الأحرف)، وتعيد True إذا كانت هناك خريطة لهذا العمود في القائمة.

رقم العنصر IndexOf:

تستقبل الصيغة الثانية لهذه الوسيلة اسم العمود الأصلي، وتعيد رقم الخانة التي يوجد بها كائن خريطة هذا العمود في القائمة إن وجد، أو تعيد -١ إن لم توجد خريطة لهذا العمود.

حذف من موضع RemoveAt:

تستقبل الصيغة الثانية لهذه الوسيلة اسم العمود الأصلي، وتبحث في القائمة عن كائن خريطة هذا العمود، وتحذفه إن وجدته.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

فئة مجموعة خريطة العمود
DataColumnMappingCollection Class

هذه الفئة تمثل الواجهة IColumnMappingCollection، وهي تعمل كقائمة تحتوي على كائنات من نوع الفئة DataColumnMapping، التي ترسم خرائط الربط بين أعمدة مجموعة البيانات والأعمدة الأصلية.

وإضافة إلى ما تمثله من خصائص ووسائل الواجهة IColumnMappingCollection، تمتلك هذه الفئة الوسائل التالية:

معرفة عمود البيانات GetDataColumn:

مماثلة للوسيلة GetDataColumn الخاصة بالفئة DataTableMapping، مع فارق وحيد، هو أنها هنا وسيلة مشتركة Shared، لهذا تمتلك معاملا زائدا، هو المعامل الأول الذي يستقبل مجموعة خرائط الأعمدة DataColumnMappingCollection التي سيتم البحث فيها.

معرفة خريطة العمود GetColumnMappingBySchemaAction:

مماثلة للوسيلة GetColumnMappingBySchemaAction الخاصة بالفئة DataTableMapping، مع فارق وحيد، هو أنها هنا وسيلة مشتركة Shared، لهذا تمتلك معاملا زائدا، هو المعامل الأول الذي يستقبل مجموعة خرائط الأعمدة DataColumnMappingCollection التي سيتم البحث فيها.

معرفة رقم العمود IndexOfDataSetColumn:

تستقبل اسم العمود في مجموعة البيانات، وتعيد رقم الخانة التي يوجد بها كائن خريطة هذا العمود في القائمة إن وجد، أو تعيد -١ إن لم توجد خريطة لهذا العمود.

واجهة خريطة العمود
IColumnMapping Interface

تمتلك هذه الواجهة خاصيتين فقط، تستخدمان لربط عمود من مجموعة البيانـــات، بـــالعمود الأصلي، وهما:

عمود المصدر SourceColumn:

تحدد اسم العمود الأصلي.

عمود مجموعة البيانات DataSetColumn:

تحدد اسم العمود في مجموعة البيانات.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

فئة خريطة العمود
DataColumnMapping Class

هذه الفئة تمثل الواجهة IColumnMapping والواجهة ICloneable، وهي ترسم خريطة الربط بين عمود مجموعة البيانات والعمود الأصلي.

ولحدث إنشاء هذه الفئة صيغتان:

١- الأولى بدون معاملات.

٢- والثانية لها معاملان نصيان، يستقبلان اسم العمود الأصلي واسم عمود مجموعة البيانات على الترتيب.

وإضافة إلى ما تمثله من خصائص، تمتلك هذه الفئة الوسيلة الوحيدة الجديدة التالية:

معرفة عمود البيانات GetDataColumnBySchemaAction:

تعيد كائن عمود البيانات DataColumn المطلوب تبعا للمعاملات التالية:

- كائن جدول البيانات DataTable الذي يحتوي العمود.. لاحظ أن اسم العمود في الجدول تحدده الخاصية DataSetColumn الخاصة بخريطة العمود الحالية.
- كائن النوع Type الذي يمثل نوع بيانات العمود.
- إحدى قيم المرقم MissingSchemaAction، تحدد ماذا سيحدث إن لم يوجد العمود في مخطط الجدول.

وتوجد صيغة أخرى لهذه الوسيلة، وهي صيغة مشتركة Shared، لهذا تزيد بمعاملين على الصيغة السابقة، هما المعامل الأول والثاني، اللذان يستقبلان اسم العمود الأصلي واسم العمود في مجموعة البيانات على الترتيب.

ولقد استخدمنا الكود التالي في المشروع Mapping لإعادة تسمية الجدول وعموديه:

```
var TM = DaAuthors.TableMappings.Add(
            "Table", "Authors-Books");
TM.ColumnMappings.Add("Author", "المؤلف");
TM.ColumnMappings.Add("Book", "الكتاب");
```

-١٠-

مصانع المزودات Provider Factories

لعلك شعرت بالاستياء من وجود أكثر من نوع من نفس الكائن للتعامل مع مزودات قواعد البيانات المختلفة، مثل:

- كائنات الاتصال مثل OleDbConnection و SqlConnection و OracleConnection.
- كائنات الأمر مثل OleDbCommand و SqlCommand و OracleCommand.
- كائنات قراءة البيانات مثل OleDbDataReader و SqlDataReader و OracleDataReader.
- مهيئات البيانات مثل OleDbDataAdapter و SqlDataAdapter و OracleDataAdapter.

فهذا يجعلك تكتب كودا مختلفا لكل نوع من أنواع قواعد البيانات، رغم أن الاختلاف ينحصر فقط في جمل تعريف الكائنات، وليس في فكرة الكود!

ولقد قدمت دوت نت ٢٠٠٥ حلا لهذه المشكلة بإضافتين هامتين:

١- تعريف الفئات العامة في النطاق System.Data.Common، مثل:

- الفئة DbConnection التي تشتق منها جميع كائنات الاتصال.
- الفئة DbCommand التي تشتق منها جميع كائنات الأوامر.
- الفئة DbDataReader التي تشتق منها كل قارئات البيانات.
- الفئة DbDataAdapter التي تشتق منها كل مهيئات البيانات.

هذا يجعل من الممكن استخدام الفئة الأم للتعامل مع أي نوع من أنواع الفئات المشتقة منها (راجع مفهوم الفئات الأساسية المجردة Abstract Base Classes وتعدد الأسماء Polymorphism في فصل الوراثة في كتاب "من الصفر إلى الاحتراف: سي شارب").

٢- إضافة الفئتين DbProviderFactories و DbProviderFactory إلى النطاق System.Data.Common، لإمدادك بمصنع خاص بمزود البيانات الذي تريد التعامل معه، مما يكمل قدرتك على تعميم الكود، كما سنرى بعد قليل.

وسنتعلم في هذا الفصل كيف نستخدم هاتين الإمكانيتين لكتابة كود واحد للتعامل مع أنواع مختلفة من قواعد البيانات، وسنستخدمه للتعامل مع قاعدة بيانات الكتب في كل من سيكويل سيرفر وآكسيس.

فئة مصانع المزودات DbProviderFactories Class

تعتبر هذه الفئة مجرد مدخل لاستخدام الفئة DbProviderFactory، وهي لا تمتلك إلا وسيلتين مشتركتين، هما:

معرفة فئات المصانع GetFactoryClasses:

تعيد جدول بيانات DataTable، يحتوي على بيانات عن مصانع المزودات المتاحة على جهاز المستخدم.. ويمثل كل صف في هذا الجدول أحد المزودات، بينما تعرض الأعمدة تفاصيل هذا المزود.. وهذه الأعمدة هي:

Name	اسم مزود البيانات.
Description	وصف مختصر لمزود البيانات.
InvariantName	الاسم الثابت للمزود، والذي يمكنك استخدامه للحصول على المصنع الخاص به.
Assembly QualifiedName	الاسم الكامل لمزود البيانات، وهو يحتوي على التفاصيل الكافية عنه، مثل الإصدار والثقافة التي يستخدمها.

ويمكنك رؤية هذه التفاصيل بنفسك في المشروع DataProviders، فهو يعرض ناتج هذه الوسيلة في جدول عرض.

معرفة المصنع GetFactory:

تعيد مصنع المزود DbProviderFactory الذي يتيح لك التعامل مع مزود معين.. ولهذه الوسيلة الصيغتان التاليتان:

١- الصيغة الأولى تستقبل الاسم الثابت للمزود InvariantName.

٢- والصيغة الثانية تستقبل صف البيانات DataRow الذي يحتوي على تفاصيل المزود.. ويمكنك الحصول على هذا الصف من الجدول العائد من الوسيلة GetFactoryClasses.

دعنا إذن نتعرف على الفئة DbProviderFactory.

فئة مصنع المزود DbProviderFactory Class

هذه الفئة أساسية مجردة تجب وراثتها، وهي تمتلك العناصر اللازمة للتعامل مع مزود البيانات الذي خصصت للتعامل معه.
وتمتلك هذه الفئة الخاصية التالية:

يمكنه إنشاء عداد لمصدر البيانات CanCreateDataSourceEnumerator:

تعيد true إذا كان مصنع المزود يسمح باستخدام الفئة DbDataSourceEnumerator للمرور عبر كل خوادم البيانات المتاحة.. وسنتعرف على هذه الفئة بعد قليل.

كما تمتلك هذه الفئة الوسائل التالية:

إنشاء باني نص الاتصال CreateConnectionStringBuilder:

تعيد باني نص الاتصال من النوع العام DbConnectionStringBuilder، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه.

إنشاء كائن اتصال CreateConnection:

تعيد كائن اتصال من النوع العام DbConnection، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه.
لاحظ أن كائن الاتصال الذي ستحصل عليه غير مرتبط بأي نص اتصال، لهذا عليك وضع نص الاتصال في الخاصية ConnectionString الخاصة به قبل محاولة فتح الاتصال.

إنشاء كائن أمر CreateCommand:

تعيد كائن أمر من النوع العام DbCommand، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه.. وأنت تعرف أنك تستطيع الحصول على قارئ البيانات من كائن الأمر باستدعاء الوسيلة ExecuteReader.

لاحظ أن كائن الأمر الذي ستحصل عليه ليس مرتبطا بأي اتصال، لهذا عليك ربطه بكائن الاتصال الذي حصلت عليه من الوسيلة CreateConnection، وهو ما فعلناه في الدالة CreateCommand في المشروع Factories كالتالي:

```
var Command = Fac.CreateCommand( );
Command.Connection = Cn;
```

ويمكنك أداء نفس وظيفة هذه الوسيلة، باستخدام الوسيلة CreateCommand الخاصة بكائن الاتصال، وفي هذه الحالة ستختصر السطر الثاني من الكود السابق:

```
var Command = Cn.CreateCommand( );
```

إنشاء معامل CreateParameter:

تعيد معاملا من النوع العام DbParameter، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه.

ويمكنك أداء نفس الوظيفة، باستخدام الوسيلة CreateParameter الخاصة بكائن الأمر.

إنشاء مهيئ بيانات CreateDataAdapter:

تعيد مهيئ بيانات من النوع العام DbDataAdapter، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه.. وقد استخدمنا هذه الوسيلة في الدالة GetTable في المشروع Factories كالتالي:

```
DataTable Table = new DataTable( );
var Da = Fac.CreateDataAdapter( );
Da.SelectCommand = Cmd;
Da.Fill(Table);
```

إنشاء باني أوامر CreateCommandBuilder:

تعيد باني أوامر من النوع العام DbCommandBuilder، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه.

إنشاء عداد مصادر البيانات CreateDataSourceEnumerator:

تعيد عدادا لمصادر البيانات من النوع العام DbDataSourceEnumerator، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه.. لاحظ أن مزود سيكويل سيرفر هو الوحيد الذي يدعم هذه الإمكانية، لأن قواعد البيانات الخاصة به تعمل على خادم، لهذا ستعيد هذه الوسيلة null إذا استخدمتها مع أي مزود بيانات آخر غير سيكويل سيرفر!

ويمكنك أن تستخدم الخاصية CanCreateDataSourceEnumerator أولا قبل استدعاء هذه الوسيلة، لتعرف إن كان المزود يدعم عداد المصادر أم لا.

إنشاء تصريح CreatePermission:

تعيد تصريحا من النوع العام CodeAccessPermission، لكنه يكون مخصصا للتعامل مع مزود البيانات الذي أنشأت المصنع الحالي للتعامل معه... لاحظ أن الفئة DBDataPermission ترث الفئة CodeAccessPermission، ومنه تشتق فئات التصريح الخاصة بكل مزود بيانات مثل SqlClientPermission.. شرح هذه المواضيع خارج نطاق هذا الكتاب.

وترث الفئات التالية الفئة DbProviderFactory:

١- OdbcFactory Class.

٢- OleDbFactory Class.

٣- OracleClientFactory Class.

٤- SqlClientFactory Class.

ولا يوجد جديد في هذه الفئات يستحق شرحه، فهي تملك نفس وسائل الفئة الأم، لكن مع فارق واحد: أنها تعيد أنواعا خاصة بكل مزود، بدلا من الأنواع العامة التي تعيد وسائل الفئة الأم.

وخير طريقة لإدراك عبقرية مصانع المزودات، هي أن نعيد كتابة المشروع DbTasks بطريقة عامة، تسمح بالتعامل مع أي مزود بيانات.. كما تذكر، فقد أنشأنا في هذا المشروع فئة اسمها MyDbConnector تسهل علينا إجراء أي عملية على قواعد بيانات سيكويل سيرفر.. الآن حان الوقت لنعمم هذه الفئة، بحيث نستطيع استخدامها للتعامل مع باقي أنواع المزودات التي يدعمها إطار العمل.. هذا هو ما فعلناه في المشروع Factories، الذي هو نسخة طبق الأصل من المشروع DbTasks، لكنه يعرض على النموذج زري تحويل Radio Buttons ليستطيع المستخدم اختيار التعامل مع قاعدة بيانات آكسيس أو قاعدة بيانات سيكويل سيرفر، والرائع حقا أن كود الأزرار الموضوعة على النموذج ظل كما هو بدون تغيير، بفضل استخدام مصانع المزودات!

لكننا بالطبع أجرينا بعض التغييرات الضرورية على كود الفئة MyDbConnector، فقد عرفنا فيها مرقما اسمه Providers، واستخدمناه في تعريف معامل ثان لحدث الإنشاء New، ليرسل المستخدم عند إنشاء نسخة من هذه الفئة، نص الاتصال ونوع المزود الذي يريد التعامل معه.

كما عرفنا دالة اسمها GetProviderName، تستقبل قيمة المرقم Providers، وتعيد النص الذي يمثل اسم هذا المزود، لنرسله إلى الوسيلة DbProviderFactories.GetFactor للحصول على مصنع المزود الذي يريد المستخدم التعامل معه.. بعد هذا يصير من السهل استخدام وسائل هذا المصنع للحصول على كائن الاتصال وكائن الأمر ومهيئ البيانات اللازمة للتعامل مع قاعدة البيانات.

ولو نظرت إلى كود الفئة MyDbConnector في هذا المشروع، فستجد أن التعديلات التي أدخلناها طفيفة، لكن تأثيرها هائل، فقد صارت لدينا فئة عامة تستطيع أداء معظم ـــ

إن لم يكن كل ــ الوظائف التي نريدها على أي نوع من أنواع قواعد البيانات، مما يتيح لك استخدامها في مشاريعك لتقليل الكود الذي تكتبه إلى أقل حد ممكن!

لاحظ أننا نغير نوع قاعدة البيانات التي نتعامل معها، في حدث تغير الاختيار CheckedChanged الخاص بزري التحويل، وذلك بالكود البسيط التالي:

```
if (RdSql.Checked)
    DbBooks = new MyDbConnector(
        Properties.Settings.Default.BooksMdfConStr,
        MyDbConnector.Providers.SqlServer)
else
    DbBooks = new MyDbConnector(
        Properties.Settings.Default.BooksMdbConStr,
        MyDbConnector.Providers.OleDb)
```

حيث DbBooks هو متغير معرف على مستوى النموذج، نضع فيه نسخة الفئة MyDbConnector التي نستخدمها لتنفيذ وظائف الأزرار.

لاحظ أيضا أن استخدام الزر "الكتب١" لاستدعاء الإجراء المخزن، يستلزم منك أولا أن تستخدم المشروع AccessStoredProcedure لإضافة الإجراء المخزن GetAuthorBooks إلى قاعدة بيانات الكتب الخاصة بآكسيس.

الطبقات المتعددة N-Tiers:

لعل المشروع السابق يكشف لك أهمية تقسيم مشاريع قواعد البيانات إلى طبقات Layers مستقلة عن بعضها.. هذا يسهل عليك تطوير أي طبقة دون تغيير أي شيء في الطبقات الأخرى.. فنحن هنا مثلا عدلنا كود الفئة MyDbConnector دون أن نغير أي شيء تقريبا في الكود الذي يستخدمها، وهي ميزة تتضح فوائدها الهائلة في المشاريع الضخمة، التي تريد الاستفادة من التطويرات التي تحدث في تقنيات قواعد البيانات، دون إعادة كتابة الكود كله منذ البداية.. فلو أن هذه المشاريع مقسمة إلى طبقات، فسينحصر التطوير على طبقة الاتصال بقاعدة البيانات للاستفادة من التقنيات الجديدة، بينما ستظل الطبقة التي تعرض البيانات للمستخدم كما هي بدون تغيير يذكر.. وتسمى البرامج التي تستخدم هذا التنظيم باسم التطبيقات متعددة الطبقات Multi-Tier Applications أو n-Tier Applications، وتسمى أيضا باسم التطبيقات الموزعة Distributed Applications لأنها مقسمة على أكثر من طبقة.. والشهير أن تكتب مشاريع قواعد على ثلاث طبقات:

١- طبقة البيانات Data Tier:

وهي الطبقة التي توجد فيها قاعدة البيانات بما فيها من جداول وعلاقات وإجراءات مخزنة، ومستخدمين وصلاحيات وقواعد سرية وحماية وصيانة وحفظ نسخ احتياطية من البيانات.. إلخ.. وفي الشركات الكبيرة يكون هناك موظفون مسئولون عن إدارة هذه الطبقة.

٢- طبقة التعامل مع البيانات Data Access Tier:

في هذه الطبقة، يوجد الكود الذي يتصل بقاعدة البيانات ويحضر النتائج منها.. والفئة MyDbConnector هي مجرد مثال مبسط على هذه الطبقة، لكن دوت نت تمنحك إمكانيات أقوى لتصميم هذه الطبقة مثل مجموعة البيانات محددة النوع ومهيئات الجداول التي سنتعرف عليها في الفصل القادم، ومثل LinQ-To-SQL وغير ذلك.

وتمتاز هذه الطبقة بأنك تستطيع استخدامها في أكثر من مشروع، مما يوفر لك الوقت والجهد، كما يمكنك تطويرها دون الحاجة إلى إعادة كتابة المشاريع التي تعتمد عليها.

٣- طبقة عرض البيانات Data Display Tier:

في هذه الطبقة يوجد الكود الذي يعرض البيانات للمستخدم، والمفروض ألا يوجد في هذه الطبقة أي كود يتصل بقاعدة البيانات.

وسنتعرف على التطبيقات متعددة الطبقات بإذن الله بصورة أشمل، في الكتاب المخصص للمواضيع المتقدمة في برمجة قواعد البيانات.

فئة عداد مصادر البيانات
DbDataSourceEnumerator Class

هذه الفئة أساسية مجردة، لكن حتى الآن لا ترثها إلا الفئة SqlDataSourceEnumerator، لأن قواعد بيانات سيكويل سيرفر هي التي تعمل على خادم، سواء أكان خادما محليا Local أو بعيدا Remote.

وتتيح لك هذه الفئة الحصول على معلومات عن الخوادم المتوفرة حاليا على الشبكة التي يتصل بها جهاز العميل.

وتمتلك هذه الفئة الوسيلة الوحيدة التالية:

معرفة مصادر البيانات GetDataSources:

تعيد جدول بيانات DataTable، يحتوي على صفوف فيها تفاصيل الخوادم المتاحة.. ويعرض هذا الجدول الأعمدة التالية:

ServerName	اسم خادم البيانات.
InstanceName	اسم النسخة التي تعمل من الخادم.. لاحظ أن سيكويل سيرفر يتيح تشغيل أكثر من نسخة من الخادم.
IsClustered	true إذا كان الخادم جزءا من تجمع Cluster من الخوادم.
Version	إصدار الخادم.

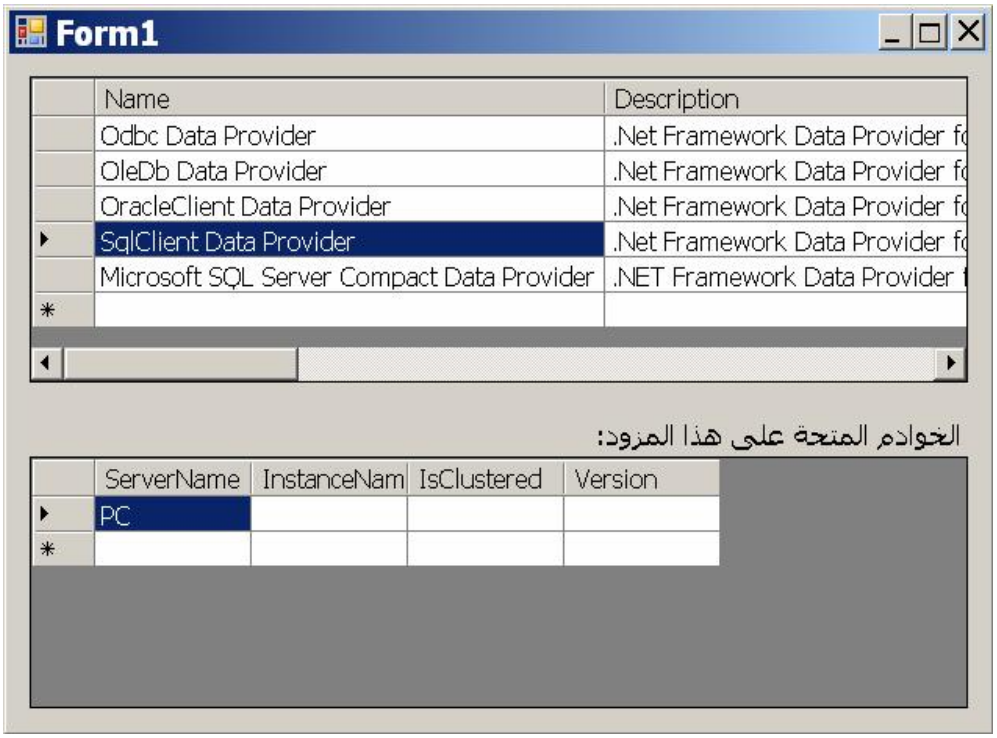
ويمكنك استخدام هذه الوسيلة لتعرض للمستخدم قائمة بأسماء الخوادم المتاحة، ليختار الخادم الذي يريد أن يتصل به.. لكن عليك أن تلاحظ ما يلي:

١- هذه الوسيلة تستهلك وقتا عند تنفيذها، بسبب بحثها عن الخوادم المتاحة على الشبكة.

٢- ناتج هذه الوسيلة قد يختلف من مرة إلى أخرى، بسبب ظهور بعض الخوادم أو اختفائها!

٣- هذه الوسيلة قد لا تعيد كل الخوادم المتاحة فعلا، لهذا عليك أن تعرض للمستخدم مربع نص أيضا، ليكتب اسم الخادم بنفسه إذا لم يجده في القائمة.

وقد استخدمنا هذه الوسيلة في المشروع DataProviders لنعرض في الجدول السفلي، الخوادم المتاحة على المزود المحدد في الجدول العلوي، كما هو موضح في الصورة:



لفعل هذا، استخدمنا الحدث RowEnter الخاص بجدول العرض، وفيه استخدمنا رقم الصف للحصول على كائن صف البيانات DataRow المناظر له في جدول المزودات، وأرسلنا هذا الصف إلى الوسيلة DbProviderFactories.GetFactory للحصول على مصنع مزود البيانات:

```
var R = TblProviders.Rows[e.RowIndex];
var Pf = DbProviderFactories.GetFactory(R);
```

بعد هذا استخدمنا الوسيلة CanCreateDataSourceEnumerator للتأكد من أن المـــزود يتيح عرض الخـــوادم، ومـــن اســـتخدمنا الوســـيلة CreateDataSourceEnumerator للحصول على عداد الخوادم، ومنه حصلنا على الجدول الذي يحتوي على تفاصـــيل هـــذه الخوادم باستخدام الوسيلة GetDataSources وعرضناه في جدول العرض:

```
if (Pf.CanCreateDataSourceEnumerator)
{
   var Se = Pf.CreateDataSourceEnumerator( );
   var TblServers = Se.GetDataSources( );
   DgServers.DataSource = TblServers;
}
else
   DgServers.DataSource = null;
```

عند تجربة هذا البرنامج على جهازك، لن تظهر أية خوادم إلا عنـــد اختيـــار مـــزود سيكويل سيرفر، حيث سيظهر الخادم المحلي Local Server المعرف على جهـــازك (وهو يمتلك نفس اسم جهازك) وفي الغالب لن يظهر الخادم SQLEXPRESS الذي يعمل على هذا الخادم المحلي!

فئة عداد مصادر بيانات سيكويل سيرفر
SqlDataSourceEnumerator Class

هذه الفئة موجودة في النطاق System.Data.Sql، وهي ترث الفئة DbDataSourceEnumerator.

وتتعامل هذه الفئة مع عداد مخصص للمرور عبر خوادم سيكويل سيرفر المتوفرة على الشبكة الحالية.

وتملك هذه الفئة خاصية واحدة جديدة، وهي:

النسخة Instance:

تعيد نسخة جديدة من الفئة SqlDataSourceEnumerator، مما يغنيك عن استخدام مصنع المزود أولا للوصول إليها.

والمشروع SqlServers يريك كيف يمكن استخدام هذه الخاصية لعرض الخوادم المتوفرة على جهازك، وهو لا يحتاج لفعل هذا، إلا إلى هذا السطر الوحيد من الكود:

```
DgServers.DataSource =
  SqlDataSourceEnumerator.Instance.GetDataSources( );
```

-١١-

مجموعة البيانات DataSet

مجموعة البيانات هي وعاء مصغر لقاعدة البيانات في برنامجك، يتيح لك أن تحمل في الذاكرة، بعض الجداول أو أجزاء منها (تبعا للاستعلام المستخدم) مع قدرتك إنشاء العلاقات بينها ووضع القيود عليها، وبهذا يمكنك التعامل مع هذه البيانات على جهازك بعد قطع الاتصال مع الخادم.

وتمتاز مجموعة البيانات بأنها عامة، فهي تستطيع التعامل مع أي نوع من أنواع قواعد البيانات، ويمكنها استيعاب الجداول دون أن يعنيها مصدرها، بينما تترك مهمة التعامل مع مصدر البيانات لمهيئ البيانات وكائن الاتصال.. لهذا يوجد في دوت نت نوع واحد فقط من مجموعة البيانات، على عكس الكائنات التي تعرفنا عليها سابقا، والتي يوجد منه نوع خاص بكل مزود.

وتستخدم مجموعة البيانات داخليا كود XML لحفظ مخططات الجداول والأعمدة Schema، وبيانات الصفوف.. ويتيح لك هذا حفظ مخططات وبيانات الجداول من مجموعة البيانات إلى جهازك في صورة وثائق XML، ومن ثم إعادة تحميلها في مجموعة البيانات مرة أخرى لاحقا.

وكما ذكرنا من قبل، تحتفظ مجموعة البيانات بنسختين من كل سجل:

١- النسخة الأصلية Original Version التي تم تحميلها من قاعدة البيانات.

٢- النسخة الحالية Current Version التي تحتوي على السجل بعد حدوث تغييرات به.

ويمتلك كل سجل الخاصية RowState التي توضح حالته، وهل دخلت عليه تغييرات وهل تم حفظ هذه التغييرات إلى قاعدة البيانات أم لا.. وبهذا التنظيم تستطيع مجموعة البيانات نقل التغييرات إلى قاعدة البيانات وتحديثها، بالاعتماد على مهيئ البيانات، الذي يعيد فتح الاتصال مع الخادم.. هذا يجعل مجموعة البيانات أفضل من قارئ البيانات في الحالات التالية:

١- إذا كان البرنامج يتعامل مع الكثير من الجداول والسجلات، ويستخدمها أكثر من مرة بدون ترتيب معين.. في هذه الحالة يكون المرور عبرها على التوالي باستخدام قارئ البيانات أمرا غير عملي.

٢- إذا كان المطلوب عرض البيانات للمستخدم والسماح له بالتعامل معها وتعديلها بحرية، والإضافة إليها والحذف منها.. أنت تعرف أن قارئ البيانات لا يقوم بتحديث السجلات، فهو للقراءة فقط.

٣- إذا كانت هناك علاقات بين الجداول وقيود مفروضة عليها، ومن المهم التعامل معها في البرنامج عند الإضافة والحذف، فمجموعة البيانات تسمح بالتعامل مع العلاقات والقيود، وهذا غير متوفر في قارئ البيانات.

لكن على الجانب الآخر، تعاني مجموعة البيانات من العيبين التاليين:

١- تعتبر مجموعة البيانات عبئا على ذاكرة الجهاز، لهذا يجب عليك تحميلها بأقل قدر ممكن تحتاجه من البيانات، ولا تضع فيها الجداول بكامل صفوفها بدون فائدة، وبدلا من هذا استخدم شرطا في جملة التحديد SELECT لتحصل على السجلات المطلوبة بالضبط. أيضا، لا تحمل من الجداول أعمدة لا يحتاجها المستخدم.

٢- قد تسبب مجموعة البيانات مشاكل عند تحديث قاعدة البيانات، وذلك إذا كان مستخدمون آخرون قد غيروا قيم بعض السجلات في قاعدة البيانات أثناء قطع الاتصال وتعاملك معها في مجموعة البيانات، فيما يسمى بمشاكل التطابق Concurrency Violations.. وقد رأينا في الفصل السابق كيف يمكن حل هذه المشكلة.

والآن، دعنا نتعرف على فئة مجموعة البيانات.

فئة مجموعة البيانات DataSet Class

هذه الفئة توجد في النطاق System.Data وهي تمثل واجهة مصدر القائمة IListSource التي سنتعرف عليها لاحقا.

ولحدث إنشاء هذه الفئة الصيغتان التاليتان:

١- الصيغة الأولى بدون معاملات.

٢- الصيغة الثانية تستقبل معاملا نصيا، يمثل اسم مجموعة البيانات، الذي سيستخدم عند حفظ مجموعة البيانات في ملف XML.

وتمتلك هذه الفئة الخصائص التالية:

اسم مجموعة البيانات DataSetName:

تحدد اسم مجموعة البيانات، ليتم استخدامه كاسم لعنصر الوثيقة Document Element في كود XML عند حفظ مجموعة البيانات.

نطاق الاسم Namespace:

تحدد اسم النطاق الذي سيتم تحته حفظ مجموعة البيانات في كود XML.

البادئة Prefix:

تحدد البادئة التي ستستخدم لتمييز العناصر التي تنتمي إلى نطاق مجموعة البيانات.. هذا مفيد إذا كان ملف XML يحتوي على نطاق الاسم فيه أكثر من مجموعة بيانات، وتريد تمييز كل منها تحت نطاق فرعي خاص بها.

ملحوظة:
عند استخدام الوسيلتين ReadXml و ReadXmlSchema لتحميل البيانات أو المخطط في مجموعة البيانات، فإنهما تبحثان في ملف XML عن نطاق الاسم الموضح في الخاصية DataSetName، ومجوعة البيانات المميزة بالبادئة الموضحة في الخاصية Prefix، فإذا لم تعثر في الملف عن مجموعة بيانات تحقق هذين الشرطين، لا يتم تحميل أي شيء من الملف.

ويريك المشروع DataSetSample مثالا على استخدام هذه الخصائص.. ستجد هذا الكود مثلا حدث تحميل النموذج:

```
Ds.Namespace = "My Project";
Ds.Prefix = "Authors-Books";
Ds.DataSetName = "DsBooks";
```

ويظهر تأثير هذه الخصائص عند ضغط الزر "حفظ المخطط في ملف"، حيث ستجد أسماء هذه الخصائص مستخدمة في تعريف مخطط مجموعة في الملف C:\DsBooksSchema.xml.

حساسة لحالة الأحرف CaseSensitive:

إذا جعلت قيمة هذه الخاصية True، فستصير عمليات المقارنة والترشيح Filtering حساسة لحالة الأحرف.. هذا يؤثر في نتائج الوسيلة DataTable.Select والخاصية DataColumn.Expression.. والقيمة الافتراضية لهذه الخاصية هي False.

لاحظ أن تغيير قيمة هذه الخاصية، سيغير تلقائيا قيمة الخاصية CaseSensitive الخاصة بكل جدول في مجموعة البيانات.

المحل Locale:

تقرأ أو تغير كائن معلومات الثقافة CultureInfo، الذي يحتوي على تفاصيل اللغة التي تستخدم لمقارنة وترتيب النصوص الموجودة في جداول مجموعة البيانات.

لاحظ أن تغيير قيمة هذه الخاصية، سيغير تلقائيا قيمة الخاصية Local الخاصة بكل جدول في مجموعة البيانات.

فرض القيود EnforceConstraints:

إذا جعلت قيمة هذه الخاصية True (وهي القيمة الافتراضية)، فسيتم التأكد من صحة القيود المفروضة على الجداول عند إجراء عمليات التحديث والإدراج والحذف.

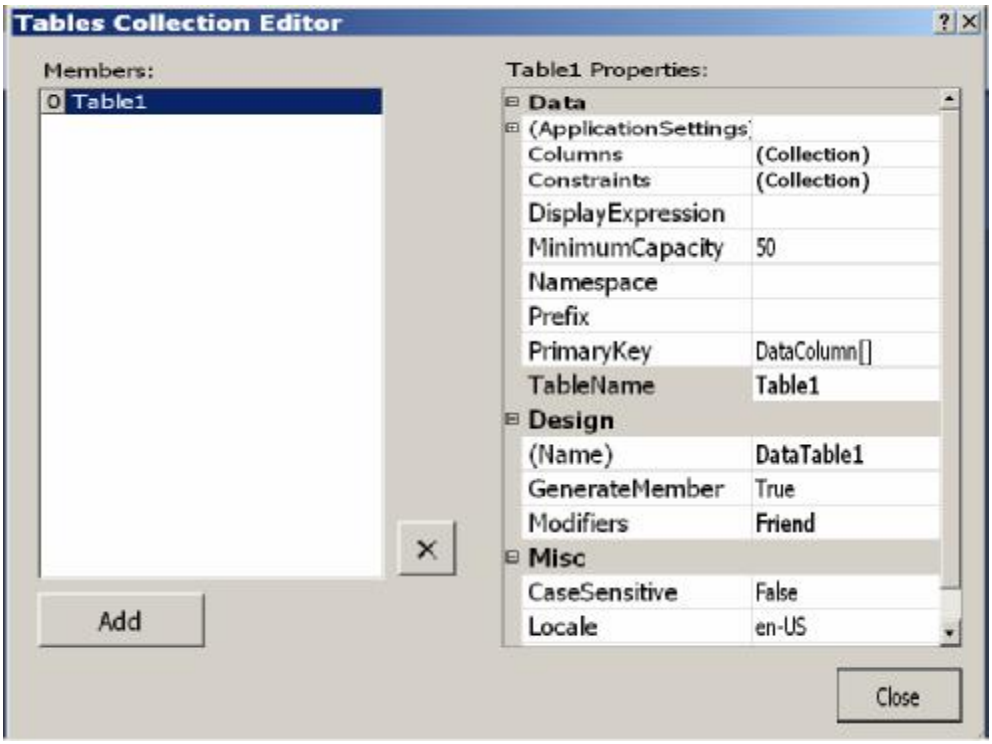
توجد بها أخطاء HasErrors:

تعيد True إذا كان أي من جداول مجموعة البيانات قد حدثت به أخطاء أثناء عملية التحديث.. ويمكنك فحص الخاصية HasErrors الخاصة بكل جدول لمعرفة الجدول الذي تسبب في الخطأ.

الجداول Tables:

تعيد مجموعة جداول البيانات DataTableCollection، التي تحتوي على كائنات الجداول DataTable Objects الموجودة في مجموعة البيانات.. وسنتعرف على هذه المجموعة بالتفصيل في الفصل التالي.

ما يعنيا هنا هو أنك تستطيع تحرير هذه المجموعة من خلال نافذة الخصائص.. فلو ضغطت زر الانتقال الموجود في خانة قيمة هذه الخاصية، فستظهر لك نافذة محرر مجموعة الجداول Table Collection Editor كما هو موضح في الصورة:



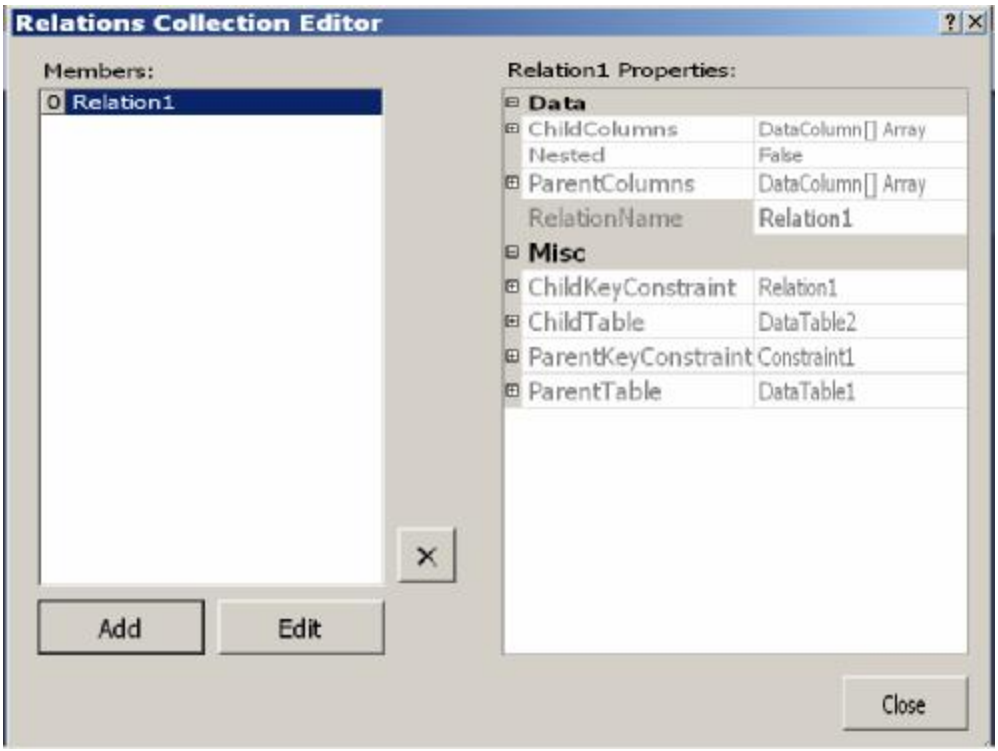
في هذه النافذة يمكنك إضافة جداول جديد بضغط الزر Add، ثم استخدام الخصائص الموجودة في القسم الأيمن من النافذة لتغيير اسم هذا الجدول وطريقة عرضه والأعمدة الموجودة به.. وسنتعرف على هذه الخصائص بالتفصيل في الفصل اللاحق.

العلاقات Relations:

تعيد مجموعة علاقات البيانات DataRelationCollection، التي تحتوي على كائنات العلاقات DataRelation Objects الموجودة في مجموعة البيانات.. وسنتعرف على هذه المجموعة بالتفصيل في الفصل التالي.

لاحظ أن ملء مجموعة البيانات بالجداول والسجلات لا يضيف العلاقات بين الجداول تلقائيا إلى مجموعة العلاقات Relations.. لهذا عليك أن تضيف هذه العلاقات بنفسك إلى مجموعة العلاقات، سواء من الكود أو باستخدام نافذة المخطط كما سنرى لاحقا.. كما يمكنك إضافة العلاقات بطريقة مرئية من خلال نافذة الخصائص.. فلو ضغطت

زر الانتقال الموجود في خانة قيمة هذه الخاصية، فستظهر لك نافذة محرر مجموعة العلاقات Relations Collection Editor كما هو موضح في الصورة:



اضغط الزر Add لإضافة علاقة جديدة.. ستظهر نافذة إنشاء العلاقة لتسمح لك بتحديد الجدولين والأعمدة المشتركة في العلاقة كما تعلمنا من قبل.. وبعد أن تضغط OK لإغلاق نافذة العلاقة، ستظهر العلاقة في القائمة اليسرى، وتظهر خصائصها في القائمة اليمنى.. ولو أردت تغيير عناصر العلاقة ، فاضغط الزر Edit لعرض نافذة العلاقة مرة أخرى.

مدير العرض الافتراضي DefaultViewManager:

تعيد كائن مدير عرض البيانات DataViewManager Object الذي يتحكم في البيانات التي تعرضها مجموعة البيانات.. وسنتعرف على هذا الكائن بالتفصيل لاحقا.

الخصائص الإضافية ExtendedProperties:

تعيد مجموعة الخصائص PropertyCollection التي تحتوي على الخصائص الإضافية التي تضيفها إلى مجموعة البيانات.

والمجموعة PropertyCollection ترث الجدول المختلط Hashtable، مما يتيح لك إضافة اسم الخاصية كمفتاح Key، والقيمة التي تريد حفظها فيها كقيمة Value.

والمثال التالي يضيف إلى مجموعة البيانات خاصية إضافية تحتفظ باسم البرنامج الخاص بك، ثم يعرض قيمة هذه الخاصية في رسالة:

```
Ds.ExtendedProperties.Add("ProgName", "MyProg");
Console.WriteLine(Ds.ExtendedProperties("ProgName"));
```

تنسيق التراسل RemotingFormat:

تحدد التنسيق الذي سيتم به إرسال البيانات من جهاز إلى آخر، عندما تتعامل مجموعة البيانات مع برنامج يستخدم التحكم عن بعد Remoting، وهي تأخذ إحدى قيمتي المرقم SerializationFormat التاليتين:

Xml	يتم إرسال البيانات في صورة نصوص XML.. هذه هي القيمة الافتراضية.
Binary	يتم إرسال البيانات في صورة أرقام ثنائية Binary.. هذا متاح فقط بدءا من إصدار إطار العمل الثاني.

لاحظ أن تغيير قيمة هذه الخاصية، يغير قيمة الخاصية RemotingFormat الخاصة بكل جدول في مجموعة البيانات.

طريقة سلسلة المخطط SchemaSerializationMode:

تحدد كيف سيتم التعامل مع مخطط البيانات عند سلسلة مجموعة البيانات محددة النوع Typed DataSet.. والسلسلة Serialization: هي تحويل محتويات كائن موجود في الذاكرة إلى بيانات يمكن حفظها في ملف أو إرسالها عبر الشبكة.. هذا يتيح لك

الاحتفاظ بحالة الكائن بعد إغلاق البرنامج لاستعادتها مرة أخرى بعد إعادة تشغيله، أو إرسالها إلى جهاز آخر للتحكم في هذا الكائن عن بعد Remoting.. وسنتعرف على السلسلة Serialization والتحكم البعيد Remoting بالتفصيل في كتاب خاص بالمواضيع المتقدمة في برمجة إطار العمل بإذن الله.

وتأخذ هذه الخاصية إحدى قيمتي المرقم SchemaSerializationMode:

IncludeSchema	إضافة مخطط البيانات ضمن عملية السلسلة.. هذه هي القيمة الافتراضية.
ExcludeSchema	عدم إضافة مخطط البيانات ضمن عملية السلسلة، مع الاكتفاء بسلسلة خصائص مجموعة البيانات التي تغيرت قيمها عن القيمة الافتراضية، وبهذا يتم تقليل حجم البيانات المسلسلة بشكل كبير.. لاحظ أن هذه القيمة لا تصلح إذا كانت مجموعة البيانات عادية (غير محددة النوع Un-typed DataSet).

وسنتعرف على مجموعة البيانات محددة النوع لاحقا في هذا الفصل.

كما تمتلك هذه الفئة الوسائل التالية:

محو Clear:

تمحو كل السجلات من كل جداول مجموعة البيانات، لكنها لا تمحو الجداول نفسها، ولا العلاقات بينها.. لاحظ أن هذه الوسيلة ستسبب خطأ في البرنامج لو كانت مجموعة البيانات تتعامل مع وثيقة XML من النوع XmlDataDocument.

تصفير Reset:

تفرغ مجموعة البيانات من جميع محتوياتها، بما في ذلك الجداول والعلاقات والقيود.

نسخ Clone:

تنسخ تركيب مجموعة البيانات (مخططات الجداول، والعلاقات والقيود) إلى مجموعة بيانات جديدة وتعيد مرجعا إليها.. لكنها لا تنسخ أي سجلات.

نسخ Copy:

تنسخ مجموعة البيانات كاملة (مخططات الجداول، والعلاقات والقيود والسجلات أيضا) إلى مجموعة بيانات جديدة وتعيد مرجعا إليها.

معرفة التغيرات GetChanges:

تعيد مجموعة بيانات جديدة، تحتوي جداولها على الصفوف التي تمّ تعديلها أو إضافتها أو حذفها منذ ملء مجموعة البيانات الأصلية، أو منذ آخر استدعاء للوسيلة AcceptChanges.. وتعيد هذه الوسيلة Nothing إذا لم تجد أية تغيرات في مجموعة البيانات.

لاحظ أن هذه الوسيلة قد تضيف إلى مجموعة البيانات الجديدة بعض الصفوف التي لم تتغير بياناتها، وذلك للمحافظة على صحة العلاقات والقيود بين الجداول، مما يتيح لك إعادة دمج مجموعة البيانات الجديدة بمجموعة البيانات الأصلية إذا أردت، دون حدوث أية أخطاء.

وتوجد صيغة ثانية لهذه الوسيلة، تستقبل إحدى قيم المرقم DataRowState، التي تمكنك من الحصول على السجلات التي حدث بها نوع محدد من التغيير دون سواه.. وهذه القيم هي:

مستقلّ Detached	تمّ إنشاء هذا السجلّ ولكنه لم يوضع بعد في مجموعة السجلات Rows الخاصة بأي جدول، أو أنه حذف للتو من مجموعة سجلات أحد الجداول.
لم يتغير Unchanged	لم تتغير بيانات هذا السجلّ، منذ أن تمّ تحميله من قاعدة البيانات أو منذ آخر استدعاء للوسيلة AcceptChanges.

مُضاف Added	هذا السجلّ ليس موجودا في قاعدة البيانات، وإنما تمّت إضافته كسجل جديد إلى مجموعة البيانات.
محذوف Deleted	تمّ حذف هذا السجلّ من مجموعة البيانات، ولكنه ما زال موجودا في قاعدة البيانات.
معدّل Modified	تمّ تعديل هذا السجلّ، ولكن لم يتمّ حفظ التعديلات في قاعدة البيانات بعد.

ويمكنك دمج أكثر من قيمة من قيم هذا المرقم معا، باستخدام المعامل OR.

تم تغييرها HasChanges:

تعيد True، إذا كانت مجموعة البيانات تحتوي على سجلات قد تم تعديلها أو إضافتها أو حذفها، ولم تحفظ بعد في قاعدة البيانات.

ويمكنك استخدام هذه الوسيلة في حدث إغلاق النموذج FormClosing، لسؤال المستخدم إن كان يريد حفظ البيانات قبل إغلاق البرنامج أم لا، وهو ما فعلناه في المشروع CustomDataSet.. وقد استخدمنا هذه الوسيلة في الزر "تحميل من قاعدة البيانات" في المشروع DataSetSample، لحفظ أية تغييرات قبل إعادة تحميل البيانات.

وتوجد صيغة ثانية لهذه الوسيلة، تستقبل إحدى قيم المرقم DataRowState التي تعرفنا عليها من قبل.. وتعيد هذه الصيغة True إذا كانت مجموعة البيانات تحتوي على سجلات وقع عليها نوع التغيير المرسل كمعامل.. والجملة التالية تخبرك إن كانت هناك سجلات جديدة أضيفت إلى مجموعة البيانات أم لا:

```
Console.WriteLine (Ds.HasChanges(DataRowState.Added));
```

قبول التغييرات AcceptChanges:

تجبر كل جداول مجموعة البيانات على استدعاء الوسيلة AcceptChanges الخاصة بها.

رفض التغييرات RejectChanges:

تجبر كل جداول مجموعة البيانات على استدعاء الوسيلة RejectChanges الخاصة بها.

إنشاء قارئ بيانات CreateDataReader:

تعيد قارئ بيانات الجداول DataTableReader، الذي يمكنك من خلاله المرور عبر سجلات كل جداول مجموعة البيانات.

وتنشئ هذه الوسيلة مجموعة نتائج Result Set لكل جدول، بنفس ترتيب الجداول في مجموعة الجداول DataSet.Tables، وإذا كان أحد الجداول خاليا من السجلات، فستوضع مقابله في قارئ البيانات مجموعة نتائج فارغة، وذلك للحفاظ على الترتيب.. ويمكنك الانتقال من قراءة سجلات جدول إلى سجلات الجدول التالي باستخدام الوسيلة NextResult الخاصة بقارئ البيانات كما تعلمنا من قبل.

وتوجد صيغة ثانية لهذه الوسيلة تتيح لك التحكم في ترتيب النتائج، حيث تستقبل مصفوفة جداول DataTable Array، تحتوي على جداول مجموعة البيانات التي تريد أن تقرأها، مع ملاحظة أن الجدول الذي يظهر في هذه المصفوفة أولا سيعرض قارئ البيانات سجلاته أولا.

ويريك الزر "إنشاء قارئ بيانات" كيف يمكنك استخدام هذه الوسيلة لعرض كل محتويات مجموعة البيانات في نافذة المخرجات Output Window.

تحميل Load:

تتيح لك هذه الوسيلة استخدام قارئ البيانات DataReader لإضافة المزيد من السجلات إلى مجموعة البيانات.. ولهذه الوسيلة ثلاث صيغ:

١- الصيغة الأولى تستقبل ثلاثة معاملات:

- معامل من نوع الواجهة IDataReader يستقبل قارئ البيانات.

- إحدى قيم المرقم LoadOption التي تحدد ماذا سيحدث إذا كانت بعض السجلات موجودة سابقا في مجموعة البيانات، وهل سيتم تحديث النسخة الأصلية من السجل Original Version أم النسخة الحالية Current Version.. وقد تعرفنا على هذا المرقم في الفصل السابق.
- مصفوفة جداول DataTable Array، تحتوي على بعض الجداول الموجودة في مجموعة الجداول DataSet.Tables، ليتم ملؤها بالسجلات من قارئ البيانات، حيث ستوضع سجلات كل مجموعة من النتائج ResultSet في الجدول المناظر لها في الترتيب في المصفوفة.

٢- الصيغة الثانية مماثلة للصيغة السابقة، إلا أن معاملها الثالث يستقبل مصفوفة نصية تحتوي على أسماء الجداول بدلا من كائنات الجداول.

٣- الصيغة الثالثة تزيد بمعامل إضافي على الصيغة الأولى.. هذا المعامل يأتي في الموضع الثالث في ترتيب المعاملات، وهو مندوب Delegate من النوع FillErrorEventHandler، وهو المندوب المستخدم في تعريف الحدث FillError الخاص بمهيئ البيانات.. ويمكنك أن ترسل إلى هذا المندوب عنوان إجراء مناسب، ليتم استدعاؤه لو حدث خطأ عند إضافة أحد السجلات إلى مجموعة البيانات.

دمج Merge:

تمزج بعض السجلات بسجلات مجموعة البيانات.. والمزج يعني أنّ السجلات الجديدة ستتمّ إضافتها إلى مجموعة البيانات، أمّا السجلات الموجودة سابقا، فسيتمّ وضع السجلات المضافة بدلا منها.. وتتم مطابقة السجلات من خلال المفتاح الأساسي لكل منها، لهذا يجب أن يحتوي جدول مجموعة البيانات على مفتاح أساسي، وإلا أدت عملية الدمج إلى تكرار نفس الصفوف مرتين.

ولهذه الوسيلة العديد من الصيغ:

١- بعض الصيغ ذات معامل واحد، يستقبل البيانات المراد مزجها، سواء كانت قادمة من مجموعة بيانات DataSet، أو جدول DataTable أو مصفوفة سجلات DataRow Array.

٢- بعض الصيغ تزيد على الصيغ السابقة بمعامل ثان، إذا جعلته True فستحتفظ مجموعة البيانات الأصليّة بالنسخة الحالية للسجلات، وسيتم المزج فقط على مستوى النسخة الأصلية... دعنا نفهم هذا بمثال صغير: افترض أن لدينا سجلا في مجموعة البيانات، فيه خانة قيمتها الأصلية ١، وقيمتها الحالية ٢.. نريد أن نمزج هذا السجل بسجل مماثل له، لكن القيمة الأصلية لهذه الخانة فيه هي ٣، وقيمتها الحالية هي ٤.. لو كانت قيمة هذا المعامل True، فستصير القيمة الأصلية لهذه الخانة في مجموعة البيانات بعد المزج ٣، لكن ستظل قيمتها الحالية ٢.. أما إذا جعلت قيمتها False، فستصير القيمة الأصلية لهذه الخانة في مجموعة البيانات بعد المزج ٣، وقيمتها الحالية ٤.. الجدول التالي يلخص هذا المثال:

	القيمة الأصلية	القيمة الحالية
سجل مجموعة البيانات	١	٢
السجل الممزوج	٣	٤
سجل مجموعة البيانات بعد المزج (قيمة المعامل True)	٣	٢
سجل مجموعة البيانات بعد المزج (قيمة المعامل False)	٣	٤

لاحظ أن جعل هذا المعامل True، هو الطريقة الوحيدة التي تستطيع بها تغيير القيمة الأصلية دون تغيير القيمة الحالية، لأن صيغ الوسيلة DataRow.Item التي تتيح لك تحديد النسخة التي تتعامل معها، قابلة للقراءة فقط، ولا يمكن استخدامها للكتابة!

وقد استخدمنا الوسيلة Merge في المشروع OptimisticConcurrency مرتين:

- مرة في حدث ضغط القائمة الموضعية "أريد حفظ تعديلاتي"، وقد أرسلنا إلى المعامل الثاني لهذه الوسيلة القيمة True لتغيير النسخة الأصلية للسجل المراد إعادة حفظه، مع الاحتفاظ بتغييرات المستخدم لحفظها في قاعدة البيانات.
- ومرة في حدث ضغط القائمة الموضعية "إلغاء تعديلاتي"، وقد أرسلنا إلى المعامل الثاني لهذه الوسيلة القيمة False للتخلص من السجل القديم، ووضع السجل القادم من قاعدة البيانات بدلا منه (يشمل هذا النسخة الأصلية والنسخة الحالية للسجل).

٣- بعض الصيغ تزيد على الصيغ السابقة بمعامل ثالث، يحدد ردّ الفعل الذي سيتخذ لو كان تركيب مجموعتي البيانات مختلفا (كعدم وجود بعض الجداول أو الأعمدة في مجموعة البيانات الحالية)، وهو يأخذ إحدى قيم المرقم MissingSchemaAction التي تعرفنا عليها من قبل.. تذكر أن القيمة الافتراضية في الصيغ التي لا تستقبل هذا المعامل هي Add، بمعنى إضافة الجداول والأعمدة اللازمة إلى مجموعة البيانات الحالية لاستقبال البيانات الجديدة من مجموعة البيانات المضافة.

ولا يتمّ التحقق من صحّة القيود Constrains، إلا بعد اكتمال عمليّة المزج.. فإذا كانت هناك سجلات تعارض القيود المفروضة، يحدث ما يلي:

- ينطلق خطأ في البرنامج من النوع ConstraintException.
- توضع القيمة False في الخاصية DataSet.EnforceConstraints لإيقاف تطبيق القيود، وذلك حتى يمكن الاحتفاظ بالبيانات الممزوجة إلى أن ترى كيف تحل المشكلة.
- يوضع نص الخطأ في الخاصية RowError الخاصة بكل سجل يتعارض مع القيود المفروضة، لهذا عليك فحص هذه الأخطاء وإصلاحها بالطريقة المناسبة،

قبل محاولة وضع القيمة True في الخاصية EnforceConstraints من جديد لتطبيق القيود.

تخمين المخطط InferXmlSchema:

تقرأ كود XML، وتحاول استنتاج مخططات الجداول من بيانات السجلات الموجودة فيها، وتحمل هذا المخطط في مجموعة البيانات.. ولهذه الوسيلة عدة صيغ، كل منها لها معاملان:

- المعامل الأول يحدد الملف الذي يوجد به كود XML، سواء كان ذلك في صورة مسار الملف، أو كائن مجرى بيانات Stream، أو قارئ نصي TextReader، أو "قارئ XML" XmlReader.
- المعامل الثاني يستقبل مصفوفة نصية، تحتوي على أسماء عناوين المواقع Url التي تريد استبعادها عند استخلاص المخطط من الملف.

الحصول على كود المخطط GetXmlSchema:

تعيد نصا يحتوي على كود XML الذي يمثل مخطط الجداول الموجودة في مجموعة البيانات.

الحصول على الكود GetXml:

تعيد نصا يحتوي على كود XML الذي يمثل البيانات الموجودة في مجموعة البيانات.

كتابة كود المخطط WriteXmlSchema:

تحفظ كود XML الذي يمثل مخطط جداول مجموعة البيانات، في الملف المرسل إليها كمعامل، سواء كان في صورة مسار الملف، أو كائن مجرى بيانات Stream، أو قارئ نصي TextReader، أو "قارئ XML" XmlReader.

وتوجد عدة صيغ لهذه الوسيلة تزيد على الصيغ السابقة بمعامل ثان من نوع المندوب Converter(Of Type, String)، وهو يستقبل عنوان أي دالة لها معامل من النوع

Type وتعيد String.. هذا مفيد إذا كانت مجموعة البيانات تحتوي على عمود يتعامل مع نوع بيانات مركب لا يمكن تحويله إلى نص مباشرة، وهنا يمكنك كتابة دالة مناسبة توضح كيف يمكن تحويل بياناته إلى نص، وترسلها إلى هذا المعامل.

كتابة الكود WriteXml:

مماثلة للوسيلة السابقة، إلا أنها تحفظ سجلات مجموعة البيانات في ملف XML.. وهناك صيغة أخرى لهذه الوسيلة، لها معامل ثانٍ من نوع المرقم XmlWriteMode الذي يمتلك القيم التالية:

IgnoreSchema	كتابة السجلات فقط بدون كتابة مخطط البيانات.. هذه هي القيمة الافتراضية.
WriteSchema	كتابة السجلات ومخطط البيانات معا في الملف.
DiffGram	كتابة كل محتويات مجموعة البيانات في الملف، بما في ذلك النسخة الأصلية Original Version والحالية Current Version لكل السجلات، حتى لو لم تتغير النسخة الحالية للسجل عن النسخة الأصلية.

وقد استخدمنا هذه الوسيلة في الزر "حفظ البيانات في ملف" في المشروع DataSetSample، وأرسلنا إلى المعامل الثاني القيمة WriteSchema لحفظ المخطط مع البيانات.. هذا يضمن لنا حفظ العلاقة بين الجدولين والقيود المفروضة عليهما، والمفاتيح الأساسية والفرعية.

ملحوظة:
إذا أردت حفظ السجلات التي تغيرت فقط، فعليك استخدام الوسيلة DataSet.GetChanges للحصول على مجموعة بيانات جديدة بها السجلات التي تغيرت فقط، واستخدام الوسيلة WriteXml الخاصة بهذه المجموعة الجديدة لحفظ سجلاتها.

قراءة كود المخطط ReadXmlSchema:

مماثلة للوسيلة WriteXmlSchema في معاملاتها، ولكنها تقوم بالوظيفة العكسية، حيث تقرأ المخطط من ملف XML وتحمّله في مجموعة البيانات.. لاحظ أن هذه الوسيلة قد تتسبب في حدوث أخطاء إذا كانت مجموعة البيانات تحتوي على مخطط بالفعل، لهذا عليك استدعاء الوسيلة DataSet.Reset أولا لمحو كل بياناتها ومخططاتها أولا، قبل استدعاء الوسيلة ReadXmlSchema.

قراءة الكود ReadXml:

مماثلة للوسيلة السابقة، إلا أنها تقرأ بيانات السجلات من ملف XML وتحملها في مجموعة البيانات.. وهناك صيغة أخرى لهذه الوسيلة، لها معامل ثانٍ من نوع المرقم XmlReadMode الذي يمتلك القيم التالية:

Auto	القيمة الافتراضية.
Read Schema	قراءة السجلات، وقراءة المخطط إن وجد في الملف (يجب أن ترسل إلى المعامل الثاني للوسيلة WriteXml القيمة WriteSchema ليتم حفظ المخطط مع البيانات، وبالتالي يمكنك قراءته).. وإذا كان بمجموعة البيانات مخطط بالفعل، تتم إضافة الجداول الجديدة إليه، لكن خطأ سيحدث لو كانت مجموعة البيانات تحتوي على جدول له نفس اسم جدول موجود في المخطط. ويؤدي طلب قراءة المخطط من ملف يحتوي على البيانات فقط، إلى عدم تحميل أي منهما في مجموعة البيانات!
Ignore Schema	قراءة السجلات فقط، مع تجاهل أي مخطط موجود.
Infer Schema	تتجاهل أي مخطط في الملف، وتحاول استنتاج المخطط من بيانات السجلات، وتضيف المخطط والسجلات إلى مجموعة البيانات..

	ويحدث خطأ إذا كانت مجموعة البيانات تحتوي على مخطط بالفعل، وكان يحتوي على عمود تتعارض تفاصيله مع عمود موجود في المخطط المضاف.
Infer Typed Schema	مماثلة للقيمة السابقة، إلا أنها تستنتج نوع بيانات كل عمود، فإن فشلت تعتبر أن نوع العمود String.
Diff Gram	تقرأ السجلات الأصلية والحالية من الملف، وذلك إذا كنت حفظتها فيه سابقا باستخدام القيمة DiffGram.. وإذا كانت مجموعة البيانات تحتوي على سجلات بالفعل فستحتفظ بها، وستضاف إليها السجلات الجديدة.
Fragment	استخدم هذه القيمة إذا كان الملف يحتوي على أجزاء من كود XML وليس كود وثيقة كاملة.

وقد استخدمنا هذه الوسيلة في الزر "قراءة البيانات من ملف" في المشروع DataSetSample، وأرسلنا إلى المعامل الثاني القيمة ReadSchema لقراءة المخطط مع البيانات.. هذا يضمن لنا إنشاء العلاقة بين الجدولين في مجموعة البيانات، لأن وظيفة البرنامج تحتاجها.

لاحظ أن الوسيلة ReadXml لا تستدعي الوسيلة AcceptChanges تلقائيا كما تفعل الوسيلة DataAdapter.Fill، لهذا فإن السجلات التي يتم تحميلها في مجموعة البيانات ستعتبر سجلات جديدة Addedd، ولو ضغطت زر الحفظ في قاعدة البيانات، فسيتم إضافة كل هذه السجلات مرة أخرى إلى جدول المؤلفين وجدول الكتب، وهذا سيجعل البيانات مكررة!.. ولحل هذه المشكلة، عليك استدعاء الوسيلة AcceptChanges مباشرة بعد تحميل السجلات إلى مجموعة البيانات، وبهذا يتم اعتبار أنها لم تتغير، ولا يتم حفظها في مجموعة البيانات.

لكنك قد تريد اعتبار السجلات جديدة في بعض المواقف، وذلك إذا كنت تملك البيانات في ملف XML وتريد إضافتها إلى قاعدة بيانات فارغة.

وهناك ملاحظة بسيطة أخرى، وهي أن هذه الوسيلة لا يهمها امتداد الملف، بل يهمها فقط صحة محتوياته.. لهذا فقد أعطينا للملفات الخاصة بنا في المشروع CustomDataSet الامتداد dsf.، وهي امتداد من اختراعنا (اختصار للتعبير DataSet Format)، وجعلنا مربع حوار فتح ملف لا يعرض سوى الملفات التي لها هذا الامتداد، وبهذا نضمن أن الملفات التي نحاول قراءتها سيكون لها الصيغة المناسبة لمجموعة البيانات، فملفات XML تستطيع حمل أي نوع من البيانات وبأي تنسيق، لكنها لن تكون جميعا صالحة للعرض في برنامجنا.

وتمتلك مجموعة البيانات الحدث التالي:

فشل الدمج MergeFailed:

ينطلق إذا فشلت عمليّة دمج بيانات جدولين باستخدام الوسيلة Merge.. يحدث هذا مثلا، إذا كان العمود المستخدم كمفتاح أساسي في السجل القادم، مختلفا عن العمود المستخدم كمفتاح أساسي في السجل الموجود مجموعة البيانات.

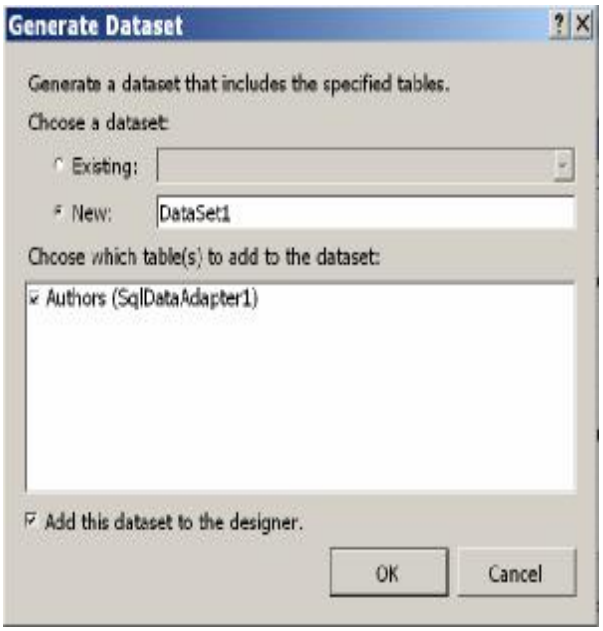
والمعامل الثاني e لهذا الحدث من النوع MergeFailedEventArgs، وهو يمتلك الخاصيتين التاليتين:

	Table	تعيد كائن الجدول DataTable الذي رفض عملية الدمج.
	Conflict	تعيد نصا يشرح سبب التعارض الذي أدى إلى فشل عملية الدمج.

المعالج السحري لإنشاء مجموعة البيانات
Generate DataSet Wezard

تتيح لك دوت نت طريقة مرئية لإنشاء مجموعة البيانات آليا.. لفعل هذا، أضف مهيئ بيانات Data Adapter إلى صينية مكونات النموذج، واضبط خصائصه كما تعلمنا من قبل، ثم اضغطه بزرّ الفأرة الأيمن، ومن القائمة الموضعيّة اضغط الأمر "إنتاج مجموعة البيانات" Generate Dataset.. وستجد نفس الأمر في القائمة الرئيسيّة Data أعلى النافذة.

سيظهر لك مربّع حوار "إنتاج مجموعة البيانات" كما هو موضح بالصورة:



في هذه النافذة يمكنك اختيار إنشاء مجموعة بيانات من مخطط مجموعة بيانات موجود سابقا في البرنامج، أو إنشاء مخطط جديد اسمه DataSet1.. لاحظ أنك تستطيع تغيير هذا الاسم، والأفضل اختيار اسم أكثر تعبيرا عن وظيفة مجموعة البيانات.

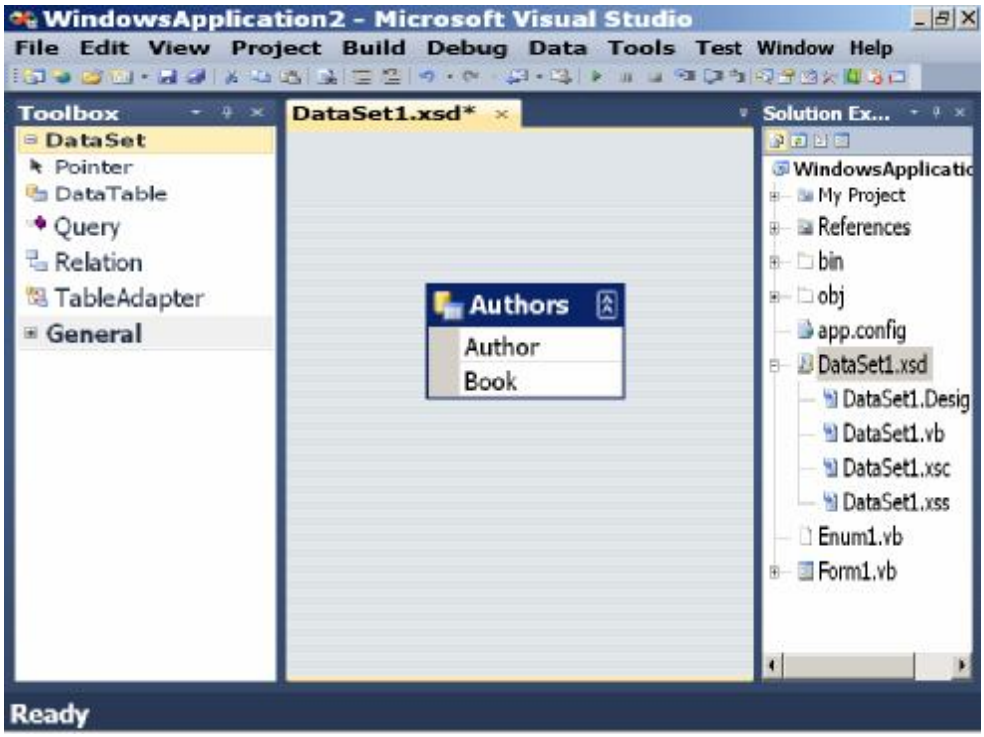
وتعرض لك النافذة قائمة بأسماء الجداول التي يوفرها مهيئ البيانات، ليمكنك اختيار إضافتها جميعا إلى مجموعة البيانات أو حذف بعضها.

ويوجد اختيار أسفل النافذة، يحدد إذا كنت تريد إضافة نسخة من مجموعة البيانات إلى النموذج أم لا.

بعد أن تحدد اختياراتك اضغط Ok لإغلاق النافذة.. سيؤدي هذا إلى ما يلي:

- إضافة ملفّ اسمه DataSet1.xsd إلى ملفات المشروع التي يعرضها متصفح المشاريع Solution Explorer.. والامتداد xsd هو اختصار للتعبير "لغة تعريف

المخطط" Xml Schema Definition، لهذا لو فتحت هذا الملف من مجلد المشروع باستخدام برنامج Notepad، فستجده يحتوي على كود XML الذي يعرف مخطط مجموعة البيانات (الجداول والأعمدة والعلاقات والقيود التي تحتويها).. أما لو نقرت هذا الملف مرتين بالفأرة في متصفح المشاريع، فستعرض لك دوت نت نافذة مصمم المخطط Schema Designer، وستجد فيها رسما مبسطا يمثل الجداول والأعمدة الموجودة في المخطط، كما هو موضح بالصورة:



- إنشاء فئة خاصة اسمها DataSet1 ترث فئة مجموعة البيانات DataSet Class.. كود هذه الفئة يوضع في الملف DataSet1.Designer.cs، والذي ستجده في العناصر الفرعية للمخطط DataSet1.xsd إذا عرضت كل ملفات المشروع بضغط الزر Show All Files الموجود أعلى متصفح المشاريع.

وتسمى الفئة DataSet1 بمجموعة البيانات محددة النوع Typed DataSet، وسنتعرف بعد قليل على معنى هذا المسمى وفائدته.

- إضافة مجموعة بيانات اسمها DataSet11 إلى صينية مكونات النموذج Component Tray.. هذه المجموعة هي نسخة معرفة من الفئة DataSet1، وستجد جملة تعريفها في ملف خصائص النموذج كالتالي:

internal DataSet1 DataSet11;

لاحظ أن الاسم الافتراضي DataSet11 يشير إلى أن هذه هي النسخة رقم ١ من الفئة dataSet1.. ولو كنت سميت مجموعة البيانات منذ البداية DsAuthorBooks مثلا، لكان اسم هذه النسخة هو DsAuthorBooks1 بدلا من DataSet11.

- ظهور أداة جديدة اسمها DataSet1 في أعلى صندوق الأدوات Toolbox تحت شريط خاص يحمل الاسم:

ProjectName Components

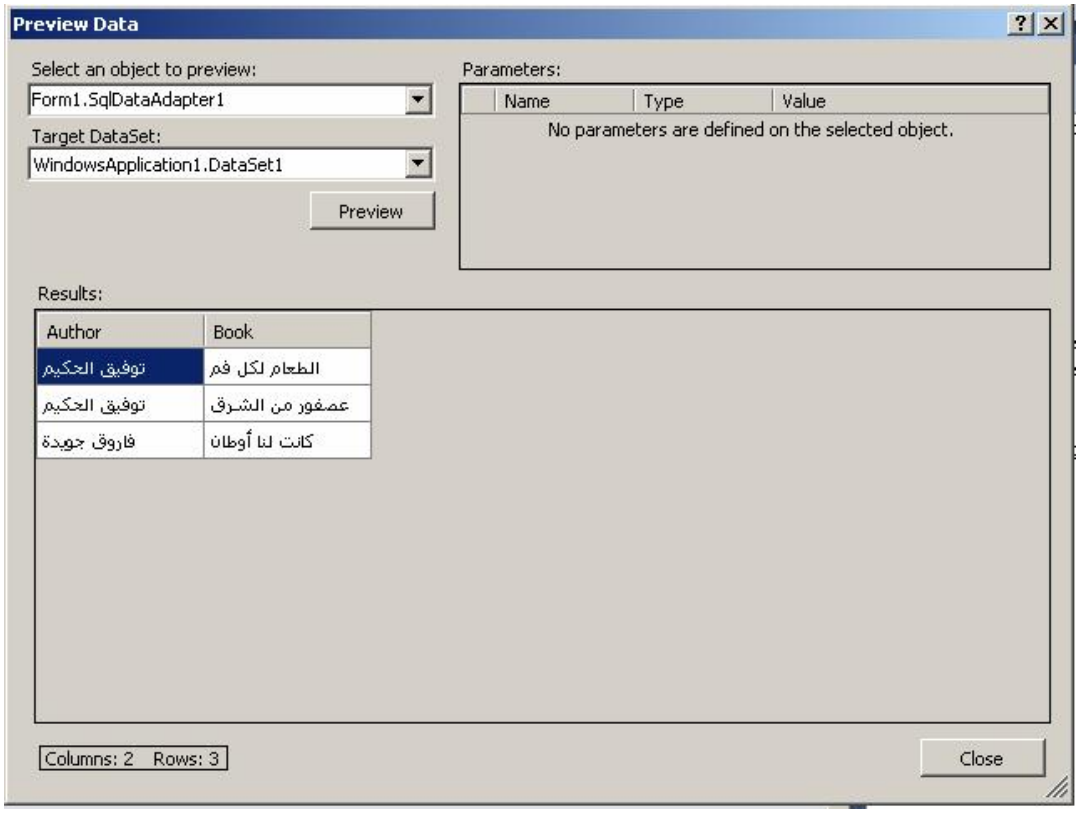
حيث ProjectName هو اسم المشروع.

وبهذا تستطيع إضافة نسخ منها إلى النموذج بطريقة مرئية.

والمشروع TypedDataSet يريك مثالا على هذا.

دعنا نرَ ماذا فعلنا حتى هذه اللحظة.. من القائمة الرئيسيّة Data، اضغط الأمر Preview Data (ستجد هذا الأمر أيضا في القائمة الموضعيّة عند ضغط مهيئ البيانات في صينيّة المكوّنات بزر الفأرة الأيمن).. هذا الأمر سيفتح نافذة استعراض البيانات التالية:

أعلى يسار هذه النافذة، توجد قائمتان منسدلتان، تتيحان لك تحديد مهيئ البيانات ومجموعة البيانات التي تريد استعراض بياناتهما، وأعلى اليمين ستجد جدولا يعرض المعاملات التي تم تعريفها في أوامر التحديد والتحديث إن وجدت.. أما الجزء السفلي من النافذة، فيعرض السجلات الناتجة من تنفيذ أمر التحديد، وهو سيكون فارغة مبدئيّا، إلى أن تضغط الزر Preview.


Preview Data
Select an object to preview:
Form1.SqlDataAdapter1
Target DataSet:
WindowsApplication1.DataSet1
Preview
Parameters:
Name
Type
Value
No parameters are defined on the selected object.
Results:
Author
Book
توفيق الحكيم
الطعام لكل فم
توفيق الحكيم
عصفور من الشرق
فاروق جويدة
كانت لنا أوطان
Columns: 2 Rows: 3
Close

مجموعة البيانات محددة النوع Typed DataSet:

رأينا كيف قامت دوت نت بإنشاء فئة اسمها DataSet1 آليا اعتمادا على المخطط DataSet1.xsd.. وتسمى هذه الفئة باسم مجموعة البيانات محددة النوع Typed DataSet، وذلك لأنها تقوم بتعريف أنواع خاصة لجداول وصفوف مجموعة البيانات، وتسمح لك بالتعامل مع الجداول والأعمدة بأسمائها مباشرة.. ولكي يحدث هذا، تقوم هذه الفئة بتعريف العديد من العناصر.. ولو فتحت الملف DataSet1.Designer.cs في المشروع TypedDataSet فستجد فيه تعريف الفئة DataSet1، وستجد فيها العناصر التالية:

١- فئة خاصة لكل صف في كل جدول في مجموعة البيانات.. هذه الفئات تحمل أسماء على الصيغة XRow، حيث X هو اسم الجدول.

وترث فئة الصف فئة صف البيانات الأم DataRow، وبداخل هذه الفئة يتم تعريف خاصية باسم كل عمود من أعمدة الجدول، تعيد قيمة الخانة الموجودة في هذا العمود في هذا الصف.. فمثلا، ستجد داخل الفئة DataSet1 فئة اسمها AuthorsRow تمثل صف البيانات في جدول المؤلفين، وستجد بداخلها خاصيتين هما: Author و Book تعيدان اسم المؤلف واسم الكتاب في الصف الحالي.

٢- فئة لكل جدول موجود في مجموعة البيانات.. هذه الفئات تحمل أسماء على الصيغة XDataTable، حيث X هو اسم الجدول.

وترث فئات الجداول الفئة عامة النوع TypedTableBase<T> والتي ترث بدورها فئة الجدول DataTable، حيث T هو نوع صفوف الجدول.

فمثلا، ستجد داخل الفئة DataSet1 فئة اسمها AuthorsDataTable تمثل جدول المؤلفين، وهي ترث الفئة TypedTableBase(Of AuthorsRow).

وبداخل فئة الجدول، يتم تعريف خصائص للتعامل مع كل عمود بالجدول، وهي تعيد كائنات من نوع فئة العمود DataColumn Class.. فمثلا، ستجد في الجدول AuthorsDataTable الخاصيتين AuthorColumn و BookColumn اللتين تتيحان لك التعامل مع عمودي المؤلفين والكتب.

كما يتم تعريف عدة أحداث لفئة الجدول إضافة إلى ما ترثه من الفئـــة DataTable، وهي:

- الصف يتغير XRowChanging.
- الصف تغير XRowChanged.
- الصف يُحذف XRowDeleting.
- الصف حُذف XRowDeleted.

حيث X هو اسم الجدول.. فمثلا: في جدول المؤلفين يتم تعريف الأحداث التالية:
AuthorsRowChanging, AuthorsRowChanged,
AuthorsRowDeleting, AuthorsRowDeleted.

٣- عدة خصائص على مستوى الفئة DataSet1 تحمل أسماء جداول مجموعة البيانات، لتتيح لك الحصول على كائن من نوع فئة هذا الجدول.. فمـــثلا، ســـتجد فـــي الفئـــة DataSet1 خاصية اسمها Authors، تعيد نسخة من الفئة AuthorsDataTable، ويمكنك من خلالها التعامل مع جدول المؤلفين.

لكن لماذا كل هذا؟.. وبم تفيدنا المجموعة محددة النوع يا ترى؟

انظر مثلا إلى الجملة التالية، التي تقرأ اسم المؤلف الموجود في الصفّ الثالث في جـــدول المؤلفين:

```
var A = DataSet11.Tables["Authors"].Rows[2]["Author"];
```

واضح طبعا أنها جملة طويلة تدفع إلى الاستياء.. فما رأيك إذن في الجملة التالية:

```
var A = DataSet11.Authors[2].Author;
```

إنّ الجملتين كلتيهما ــ ويا للعجب ــ متكافئتان، وإن كانت الأولى عامّة تستخدم خصـــائص فئة مجموعة البيانات الأم DataSet Class، بينما الثانيـــة خاصّـــة، تســـتخدم خصـــائص مجموعة البيانات DataSet1 محددة النوع.. لاحظ أن الجملة الثانيـــة تمنحـــك الميـــزات التالية:

١- مختصرة وواضحة ومفهومة.

٢- أقل عرضة للخطأ.. ففي الجملة الأولى (الطويلة) هناك احتمالان للخطـــأ، وذلـــك أثناء كتابتك لاسمي الجدول Authors والعمود Author، لأنـــك تكتبهمـــا يـــدويا

كنصوص، ولا يتم اكتشاف أي خطأ فيهما إلا أثناء تشغيل البرنامج.. أما في الجملة الثانية (القصيرة)، فإنك تتعامل مع خصائص معرفة سابقا في الفئة DataSet1، ولن يقبل محرر الكود أي خطأ في أسمائها، مما يعني انعدام أي فرصة للخطأ.

٣- لا تحتاج عند كتابتها إلى تذكر أسماء الجداول والأعمدة بنفسك، وهو أمر تتضح أهميته في قواعد البيانات الضخمة التي تحتوي على عشرات الجداول، التي يحتوي كل منها على عشرات الأعمدة، مما يعني أنك ستضيع الكثير من الوقت لو استخدمت مجموعة بيانات عادية، لأنك ستضطر إلى العودة إلى قاعدة البيانات كثيرا لتذكر أسماء عناصرها.. بينما مجموعة البيانات محددة النوع تجعل الحياة جنة، لأن الاستشعار الذكي IntilliSense سيعرض لك قائمة الأسماء بمجرد كتابة النقطة . لتختار منها اسم الجدول أو العمود الذي تريد التعامل معه.

كل هذا يوضح لك فوائد مجموعة البيانات محددة النوع، وكيف تختصر وتسهل كتابة الكود بشكل كبير.

والمشروع DataSetContents يريك كيف يمكن عرض كل جداول وعلاقات وبيانات مجموعة البيانات محددة النوع، بالطريقة الموضحة في الصورة:

ملحوظة:

يمكنك استعارة مخطط XML من مشروع آخر، وإنشاء مجموعة بيانات محددة النوع بناء عليه.. لفعل هذا، اتبع الخطوات التالية:

- أنشئ مشروعا جديدا.
- من القائمة العلوية Project، اضغط الأمر Add Existing Item.
- استخدم مربع حوار فتح ملف للوصل إلى مجلد المشروع DataSetContents، واختر الملف DsAuthorsBooks.xsd.. سيضاف هذا الملف إلى المشروع.
- اعرض النموذج، وافتح صندوق الأدوات، وأضف مجموعة بيانات إلى النموذج.. وفي مربع الحوار الذي سيظهر اختر Typed DataSet.. ستجد أن القائمة المنسدلة تعرض العنصر X.DsAuthorsBooks، حيث X هو اسم المشروع.. اضغط OK.
- ستضاف مجموعة بيانات اسمها DsAuthorsBooks1 إلى صينية المكونات.. يمكنك استخدام نافذة الخصائص لتغيير اسمها إلى أي اسم مناسب، وليكن Ds.
- اضغط هذه المجموعة بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر EditIin DataSet Designer.. سيؤدي هذا إلى فتح مخطط XML، وستجد فيه مخطط جدول المؤلفين، ومخطط جدول الكتب، والعلاقة بينهما.

إنشاء مجموعات بيانات خاصّة Custom DataSet:

في هذا المقطع سننشئ مجموعات بيانات بدون تحميل أية تفاصيل من قاعدة البيانات.. سننشئها باستخدام مخطط XML، وسنربطها بجدول عرض DataGridView بحيث يستطيع المستخدم إدخال البيانات بها، وسنسمح له بحفظ هذه البيانات في ملفّ XML، وإعادة تحميلها بعد ذلك كما يشاء.

ابدأ مشروعا جديدا اسمه CustomDataSet، ومن القائمة الرئيسية Project اضغط الأمر Add New Item لعرض نافذة إضافة عنصر.. من القائمة اليسرى اختر العنصر Data، ومن القائمة اليمنى اختر العنصر DataSet، وحدّد اسما لهذا العنصر الجديد وليكن MyDataSet، واضغط الزرّ OK.

سيضاف مخطّط XML إلى المشروع اسمه MyDataSet.xsd.. انقره مرتين بالفأرة لعرض مصمم المخطط.

لو فتحت صندوق الأدوات الآن، فستجد به أدوات تناسب مخطط XML، وستكون مبوبة تحت الشريط DataSet.. انقر مرّتين بالفأرة على العنصر DataTable لإضافة جدول جديد إلى المخطّط.. هذا الجدول سيظهر على المخطّط في صورة مستطيل فارغ، يحمل الاسم الافتراضيّ DataTable1.. لتغيّر هذا الاسم، اضغطه بالفأرة لإظهار مربع التحرير، واكتب الاسم الجديد Students، ثم اضغط Enter.. كما يمكنك استخدام نافذة الخصائص لتغيير اسم الجدول.

ولإضافة عمود إلى هذا الجدول، اضغطه بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط Add ثم Column.. حرر الاسم الافتراضي للعمود الجديد، واجعل اسمه ID.. اضغط F4 لعرض نافذة الخصائص، واستخدم الخاصية DataType لجعله من النوع Int32.. ويمكنك استخدام باقي الخصائص للتحكم في العمود بالطريقة التي تناسبك.. مثلا: اجعل للخاصية AutoIncrement القيمة True لجعل هذا الحقل ترقيما تلقائيا، ولا تنسَ أن تجعل للخاصيتين AutoIncrementSeed و AutoIncrementStep القيمة ١.

اضغط الهامش الأيسر للعمود ID بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Set Primary Key لجعله المفتاح الأساسي.

أضف إلى الجدول عمودا جديدا بنفس الطريقة واجعل اسمه Name.. سيكون نوع هذا العمود String بصور افتراضية، فاتركه كما هو.. يمكنك أن تحدد الخاصية Unique في نافذة الخصائص وتجعل قيمتها True، لتجعل اسم التلميذ متفردا غير قابل للتكرار.. ويمكنك أن تضع في الخاصية MaxLength القيمة ٣٠ لرفض أي اسم أطول من ٣٠ حرفا.

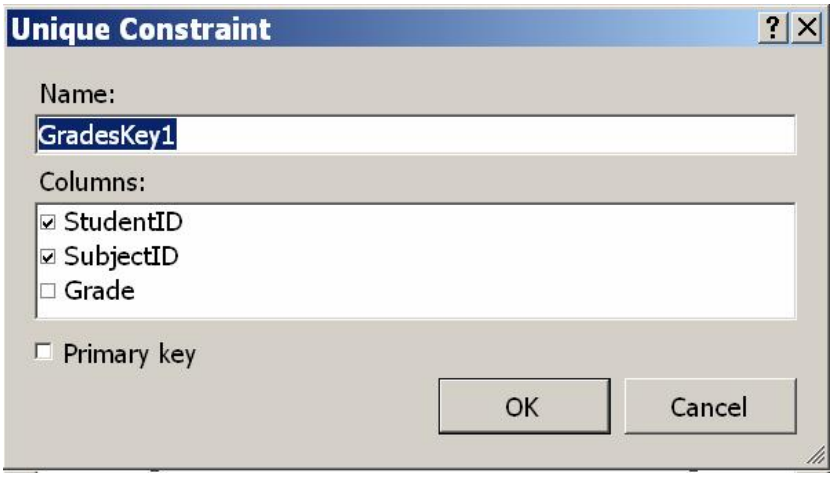
ولو أردت إدراج أي عمود قبل العمود Name، فاضغط هامشه الأيسر بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Insert Column.. وتستطيع حذف أي عمود في أي لحظة بتحديده وضغط الزر Delete.

بنفس الطريقة يمكنك إضافة جدول آخر اسمه Subjects، فيه العمودان: ID و Name.. لاحظ أنك تستطيع نسخ الجدول Students باستخدام الأمر Copy ولصق نسخة جديدة منه باستخدام الأمر Paste، حيث سيأخذ الجدول الجديد الاسم Students1، والذي يمكنك تغييره إلى Subjects.. هذا يسهل عليك إنشاء الجداول المتشابهة في تركيبها.

أضف جدولا ثالثا اسمه Grades، وأضف إليه الأعمدة StudentID و SubjectID و Grade، ولا تنس أن تغير نوع بياناتها جميعا إلى Int16.

واضح أننا سنسجل في الجدول Grades درجات كل طالب في كل المواد.. هذه علاقة متعدد بمتعدد Many-to-Many، فالطالب مرتبط بكل المواد، والمادة مرتبطة بكل الطلاب.. هذه فرصة لنجرب التعامل مع هذه العلاقة.

ويجب هنا أن نجعل الحقلين StudentID و SubjectID معا زوجا متفردا، حتى لا نكرر درجة نفس التلميذ في نفس المادة.. لفعل هذا، حدد هذين الحقلين (بضغطهما بزر الفأرة مع ضغط الزر Ctrl من لوحة المفاتيح)، ثم انقرهما بزر الفأرة الأيمن، ومن القائمة الفرعية Add اضغط Key.. ستظهر نافذة إضافة قيد التفرد Unique Constraint، كما هو موضح في الصورة:



في مربع النص العلوي اكتب اسم القيد، وفي القائمة السفلية تأكد أنك اخترت الأعمدة التي سيتم تطبيق القيد عليها (ستجد العمودين StudentID و SubjectID مختارين فعلا لأنك حددتهما قبل فتح النافذة).. ولو أردت جعل هذين العمودين مفتاحا أساسيا للجدول أيضا، فضع علامة الاختيار أمام الاختيار Primary Key أسفل النافذة.. لكننا لا نحتاج إلى هذا هنا.. اضغط OK لإنشاء قيد التفرد.

نريد الآن أن ننشئ العلاقات بين هذه الجداول.. يمكنك نقر العنصر Relation مرتين بالفأرة في صندوق الأدوات لعرض نافذة إنشاء العلاقة.. أو يمكنك أن تسحب العمود الأساسي من الجدول الرئيسي، وتسقطه على العمود الفرعي في الجدول التابع.. ولا تمر وأنت تسحب أي عمود على عمود آخر في نفس الجدول، وإلا فسيتم تحديده واعتباره جزءا من العلاقة.. لكنك تستطيع تصحيح ذلك في نافذة العلاقة على أي حال.. هذه النافذة مألوفة، ولن تجد فيها أي جديد لم نتعرف عليه سابقا.

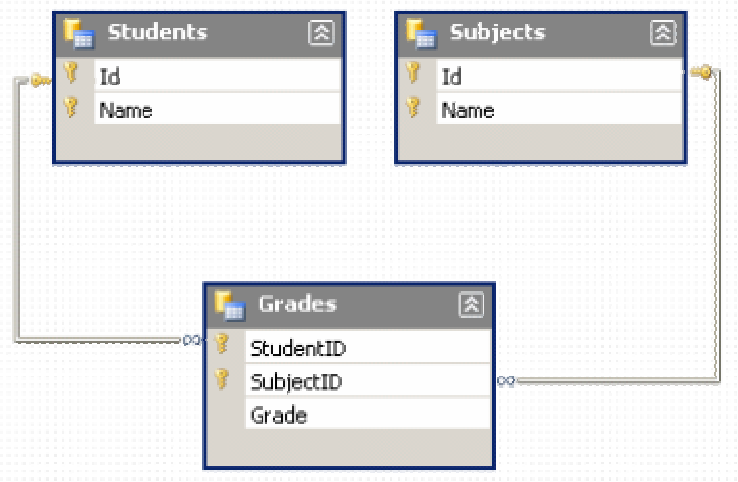
نريد هنا أن نربط بين العمودين Students.ID و Grades.StudentID، وكذلك بين العمودين Subjects.ID و Grades.SubjectID.. ولا تنس المحافظة على التكامل المرجعي في كل علاقة، وذلك باختيار:

Both Relation And Foreign Key Constraint

من القسم: Choose what to create، على أن تجعل اختيارات الحذف والتحديث والرفض Cascade.. هذا سيريحنا من المشاكل التي تحدث عند حذف اسم طالب، أو

تغيير اسم مادة، فالتكامل المرجعي سيحافظ على جدول الدرجات صحيحا دائما.

بعد إنجاز هذا، يجب أن يبدو المخطط كالتالي:



نريد الآن إنتاج مجموعة بيانات من هذا المخطط.. لفعل هذا انتقل إلى النموذج، وافتح صندوق الأدوات وانقر مرتين على العنصر DataSet.. سيظهر لك مربّع حوار إضافة مجموعة بيانات، وستجد في القائمة العلوية اسم المخطّط MyDataSet.. اضغط OK لإنتاج مجموعة بيانات محددة النوع من هذا المخطط، حيث ستضاف نسخة منها اسمها MyDataSet1 إلى صينية مكونات النموذج.. أقترح تغيير اسمها إلى DsStudents.

من هذه النقطة، يمكنك استخدام مجموعة البيانات بنفس الطريقة التي اعتدتها سابقا، وربطها بجدول العرض، وحفظ وتحميل البيانات بالطريقة المألوفة.. وستجد الكود الكامل الذي يفعل هذا في المشروع CustomDataSet.

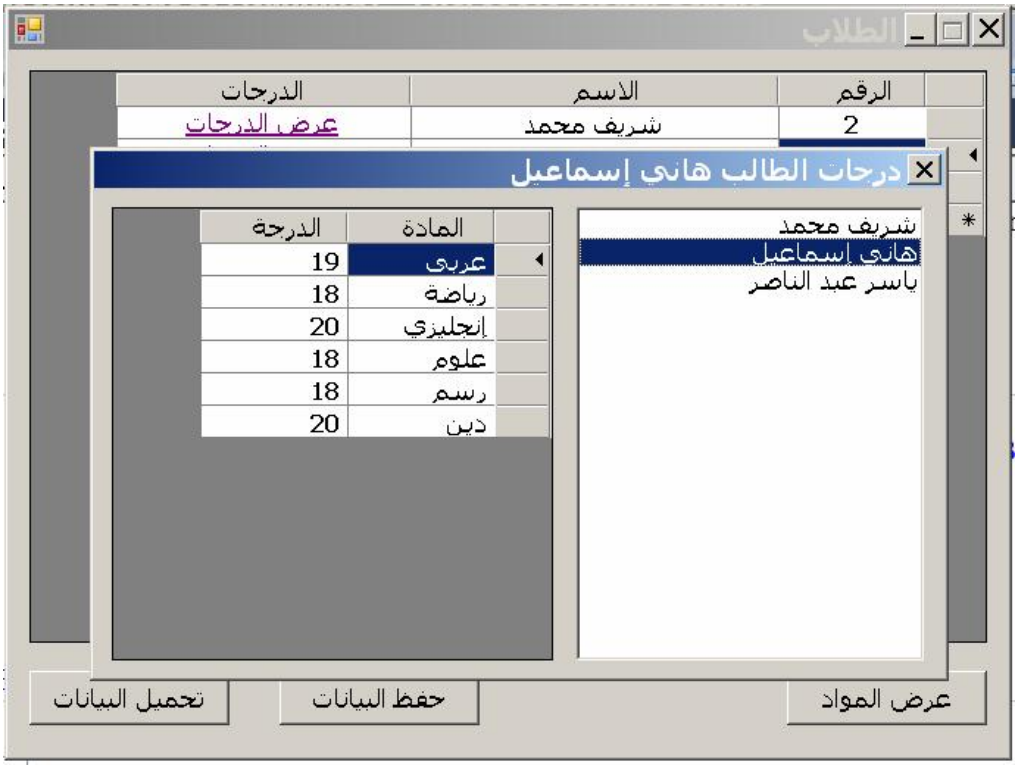
لاحظ أن إجراء أي تعديل على مخطط XML، ينعكس مباشرة على فئة مجموعة البيانات محددة النوع، لهذا لست في حاجة إلى حذفها ثم إعادة إنشائها، فكل شيء يتم تلقائيا بمنتهى البساطة.

جرب مثلا إضافة عمود محسوب Calculated Column إلى الجدول Grades.. لفعل هذا افتح المخطط، واضغط الجدول Grades بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Insert Column.. سمِّ العمود الجديد Subject، وفي نافذة الخصائص حدد

الخاصية Expression وضع فيها النص:

Parent(Subjects_Grades).Name

بمجرد أن تفعل هذا ستصير للخاصية ReadOnly القيمة True، وهذا معناه أن المستخدم لا يستطيع تعديل قيم هذا العمود، لأنه سيعرض ناتجا محسوبا بناء على قيمة عمود آخر.. وفي حالتنا هذه، جعلنا هذا العمود يعرض اسم المادة الدراسية، وذلك من خلال العلاقة Subjects_Grades التي تربط جدول المواد بجدول الدرجات، حيث سيستخدم العمود Subject قيمة الحقل الفرعي في هذه العلاقة (وهو الحقل SubjectID) ليحضر اسم المادة التي لها نفس الرقم من جدول المواد.. وسنتعرف على الأعمدة المحسوبة بتفصيل أكثر في فصل الجداول.



تلاحظ كما هو واضح في الصورة، إن إضافة اسم المادة إلى هذا الجدول سيجعلها تظهر في النافذة التي تعرض درجات الطالب، وهذا أفضل من إرباك المستخدم بعرض رقم المادة، والدرجة التي حصل عليها الطالب فيها.

لاحظ أننا استخدمنا قائمة ListBox لعرض أسماء الطلبة.. هذا يريح المستخدم أثناء إدخاله لدرجات الطلب، بسبب سرعة الانتقال من طالب إلى آخر.

ولعلك تتساءل: لمَ لا يدخل المستخدم أسماء الطلبة ودرجاتهم في نفس النموذج؟

هذا مجرد مثال مختصر، لكن في البرامج الحقيقية، ستحتاج إلى إدخال بيانات الطالب كاملة وليس اسمه فقط، مثل عمره، وفصله الدراسي، وعنوانه، وهاتفه... إلخ.. وكل هذا يحتاج إلى مساحة عرض كبيرة، والأفضل عمل نموذج مستقل له، وهو ما فعلناه في هذا المشروع، ليمكنك البناء عليه فيما بعد.

إلى الآن كل شيء رائع.. لكن تتبقى مشكلة في هذا البرنامج، وهي أن العمود المحسوب Subject سيتم حفظه في الملف عند حفظ مجموعة البيانات، وهو ما سيزيد من حجم الملف بلا ضرورة.. لهذا علينا حذف هذا العمود من مجموعة البيانات قبل حفظها.. لكن هذا سيؤدي إلى حدوث أخطاء في البرنامج!

ويمكن حل هذه المشكلة، باستخدام الوسيلة DataSet.Copy لنسخ مجموعة البيانات إلى مجموعة بيانات احتياطية، ثم حذف العمود Subject من هذه المجموعة الاحتياطية، وحفظها بياناتها في الملف.. وبهذا تظل مجموعة البيانات الأصلية كما هي، بينما نحصل على ملف أصغر حجما.. وعند تحميل هذا الملف، لن تحدث أية مشكلة في البرنامج بسبب غياب العمود Subject، فهو عمود محسوب، وسيستنتج البرنامج قيمته.. وستجد الكود الذي ينفذ هذا في الزر "حفظ البيانات".

لاحظ أن عيب هذه الطريقة هو أن نسخ مجموعة البيانات بكاملها قد يكون كارثة على الذاكرة إذا كان حجم بياناتها ضخما.. لهذا يمكن اللجوء إلى حل بديل، وهو حذف العمود من مجموعة البيانات قبل حفظها، ثم إنشائه مرة أخرى بعد الحفظ مباشرة.. أسهل طريقة لفعل هذا هي الاحتفاظ بمرجع للعمود في متغير من النوع DataCoulmn قبل حذفه من مجموعة أعمدة الجدول، ومن ثم حفظ البيانات في الملف، ثم إضافة العمود الذي نحتفظ بمرجعه إلى مجموعة الأعمدة مرة أخرى.. وستجد الكود الذي يفعل هذا في الزر "حفظ البيانات ٢".

حفظ بيانات الشجرة في مجموعة البيانات:

في المشروع CustomDataSet، رأينا مثالا على علاقة متعدد بمتعدد.. لعله يكون مناسبا الآن أن نرى مثالا على العلاقة الذاتية Self-Relation.. لفعل هذا، سننشئ مشروعا اسمه SaveTreeNodes، وهو يعرض شجرة ويتيح للمستخدم إضافة العناصر إليها، وتغيير مستوياتها، وهي وظائف تعلمنا كيف ننشئها في المشروع TreeViewSample في كتاب برمجة الويندوز، ولن نكرر شرحها هنا.. وستجد المشروع SaveTreeNodes ضمن أمثلة هذا الكتاب.

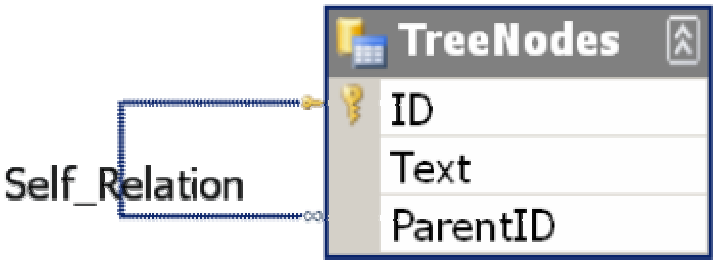
ما نريده الآن، هو أن نسمح للمستخدم بحفظ فروع الشجرة.. ونظرا، لأنه من غير العملي إنشاء قاعدة بيانات كاملة لحفظ بعض عناصر الشجرة، فسيكون من العملي هنا أن ننشئ مجموعة بيانات خاصة، ونستخدمها لحفظ العناصر في ملف XML.. وبهذا نكون قد استفدنا من قدرات مجموعة البيانات وعلاقاتها، وفي نفس الوقت سنحفظ البيانات في ملف مستقل.

لفعل هذا، أضفنا إلى المشروع SaveTreeNodes مخطط مجموعة بيانات بالطريقة المألوفة، وأسميناه TreeDataSet، وأضفنا إليه جدولا اسمه TreeNodes، وأضفنا إليه الأعمدة التالية:

اسم العمود	نوع بياناته	وظيفته
ID	Int16	المفتاح الأساسي، وهو يعمل أيضا كترقيم تلقائي.
Text	String	يحفظ نص فرع الشجرة.
ParentID	Int16	المفتاح الفرعي، وهو يشير إلى رقم الفرع الرئيسي للفرع الحالي.

وقد أضفنا علاقة إلى مخطط الجدول، لتربط بين الحقلين ID و ParentID.

هكذا يبدو شكل المخطط.. لاحظ كيف تخرج العلاقة من نفس الجدول وتعود إليه:



وقد أضفنا إلى النموذج نسخة من مجموعة البيانات وأسميناها TreeDs.. وحتى لا نعقد الأمور على أنفسنا، لن نملأ مجموعة البيانات بعناصر الشجرة إلا عند ضغط زر الحفظ، فهي مجرد وعاء وسيط يتيح لنا حفظ البيانات في ملف XML.. هذا هو كود هذا الزر:

```
TreeDs.Clear( );
foreach (TreeNode Node in TreeView1.Nodes) {
  var R = TreeDs.TreeNodes.AddTreeNodesRow(Node.Text, null);
  SaveChildren(Node, R);
}
TreeDs.WriteXml("C:\\TreeNodes.Xml");
```

في البداية أفرغنا مجموعة البيانات من محتوياتها، ثم مررنا عبر جذور الشجرة لإضافتها إلى الجدول TreeNodes الموجود في مجوعة البيانات.. لاحظ أن مجموعة البيانات محددة النوع قد أضافت الوسيلة AddTreeNodesRow إلى الجدول، لتتيح لنا إضافة صف جديد إليه.. هذه الوسيلة تستقبل معاملين:

- نصا يمثل قيمة الحقل Text في الصف الجديد.
- كائن صف من النوع TreeDataSet.TreeNodesRow، لترسل إليه الصف الرئيسي للصف الحالي.. هذا أسهل من أن تضع بنفسك رقم الصف الرئيسي في الحقل ParentID، وهذه إحدى التسهيلات التي منحتها لك العلاقة الذاتية.. ونظرا لأن جذور الشجرة ليست لها فروع رئيسية، فسنرسل القيمة Nothing إلى هذا المعامل.. هذا سيترك الحقل ParentID فارغا.

بعد هذا، يجب أن نضيف إلى مجموعة البيانات فروع كل جذر.. لفعل هذا استخدمنا إجراء اسمه SaveChildren، وهو يستقبل معاملين:

- كائن الفرع TreeNode الذي سنضيف عناصره الفرعية إلى مجموعة البيانات.

- كائن الصف TreeDataSet.TreeNodesRow الذي يعمل كصف رئيسي، للصفوف التي سنضيفها الجدول.

هذا هو كود هذا الإجراء، مع ملاحظة أنه إجراء ارتدادي Recursive يستدعي نفسه، لأن كل فرع قد يحتوي على عناصر فرعية، كل منها قد يحتوي على عناصر فرعية، وهكذا:

```
public void SaveChildren(TreeNode ParentNode,
            TreeDataSet.TreeNodesRow ParentRow) {
  foreach (TreeNode Node in ParentNode.Nodes)
  {
    var R = TreeDs.TreeNodes.AddTreeNodesRow(
        Node.Text, ParentRow);
    SaveChildren(Node, R);
  }
}
```

وبعد وضع جميع بيانات الفروع في مجموعة البيانات، سيتم تنفيذ آخر سطر في كود ضغط زر الحفظ، وهو يستدعي الوسيلة WriteXML لحفظ محتويات مجموعة البيانات في الملف.

وهكذا نكون قد حفظنا فروع الشجرة بالكامل.. بقي إذن أن نعيد قراءتها من الملف عند ضغط زر التحميل.. لفعل هذا سنفرغ كلا من مجموعة البيانات والشجرة من محتوياتهما، ثم نقرأ بيانات الملف باستخدام الوسيلة ReadXML:

```
TreeDs.Clear( );
TreeView1.Nodes.Clear( );
TreeDs.ReadXml("C:\TreeNodes.Xml");
```

بعد هذا سننقل البيانات من مجموعة البيانات إلى الشجرة.. لفعل هذا سنضيف الجذور إلى الشجرة أولا.. نحن نعرف أن الجذر ممثل في الجدول بصف توجد في الخانة ParentID الخاصة به القيمة DbNull.. لهذا سنمر على كل الصفوف، ونستخدم الوسيلة الجاهزة IsParentIDNull التي عرفتها لنا مجموعة البيانات محددة النوع، لنرى إن كانت هذه الخانة فارغة، فإن كانت كذلك، عرفنا فرعا جديدا، ووضعنا فيه النص الموجود في الخانة Text، وأضفناه إلى الشجرة كجذر، ثم نستدعي الإجراء LoadChildren لتحميل العناصر الفرعية في هذا الجذر.. هذا هو الكود الذي يفعل هذا:

```
foreach (var R in TreeDs.TreeNodes)
   if (R.IsParentIDNull) {
      var Node = TreeView1.Nodes.Add(R.Text);
      LoadChildren(Node, R);
   }
```

أيضا، يجب أن يكون الإجراء LoadChildren ارتداديا Recursive يستدعي نفسه، لتحميل العناصر الفرعية لكل فرع في جميع المستويات.

ولكن كيف نعرف العناصر الفرعية؟

هذا سهل جدا، بفضل العلاقة الذاتية المعرفة في الجدول، فنحن نستطيع استخدام الوسيلة GetChildRows لمعرفة الصفوف الفرعية التابعة لأي صف رئيسي.. هذا هو كود هذا الإجراء:

```
public void LoadChildren(TreeNode ParentNode,
        TreeDataSet.TreeNodesRow ParentRow)
{
   foreach (TreeDataSet.TreeNodesRow R in
            ParentRow.GetChildRows("Self_Relation"))
   {
      var Node = ParentNode.Nodes.Add(R.Text);
      LoadChildren(Node, R);
   }
}
```

هذا هو كل شيء.. يمكنك الآن تجربة البرنامج، وإضافة العناصر إلى الشجرة، وحفظها، ثم استرجاعها في أي وقت.

رائعة هي العلاقة الذاتية؟.. أليس كذلك؟

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

فئة مهيئ الجدول TableAdapter Class

فئة مهيئ الجدول ليست فئة من فئات إطار العمل، لكنها فئة يقوم مصمم مجموعة البيانات محددة النوع Typed DataSet Designer بإنشائها في برنامجك لتسهيل تعاملك مع قواعد البيانات.

ويشبه مهيئ الجدول TableAdapter مهيئ البيانات DataAdapter في كل شيء، فهو يسمح لك بالحصول على السجلات من قاعدة البيانات لملء أحد جداول مجموعة البيانات.. ولكنه يتفوق على مهيئ البيانات في قدرته على تنفيذ عدد كبير من استعلامات التحديد SELECT للحصول على سجلات الجدول بطرق مختلفة من قاعدة البيانات، بشرط أن يكون الناتج ملائما لتركيب الجدول الذي يتم ملؤه.. بينما مهيئ البيانات مهيأ للتعامل مع استعلام واحد فقط.

ولكي تنشئ مهيئ جدول، يجب أن يحتوي برنامجك على مجموعة بيانات محددة النوع أولا.. اتبع هذه الخطوات:

- أنشئ مشروعا جديدا اسمه TableAdapter.
- من القائمة العلوية Project اضغط الأمر Add New Item.
- من القائمة اليسرى اختر العنصر Data، ومن القائمة اليمنى اختر العنصر DataSet، وامنحها الاسم DsAuthorsBooks.xsd.
- في نافذة المخطط، افتح صندوق الأدوات، واسحب العنصر TableAdapter وأسقطه على مصمم المخطط.. سيؤدي هذا إلى بدء المعالج السحري لتهيئة مهيئ الجدول:

TableAdapter Configuration wizard.

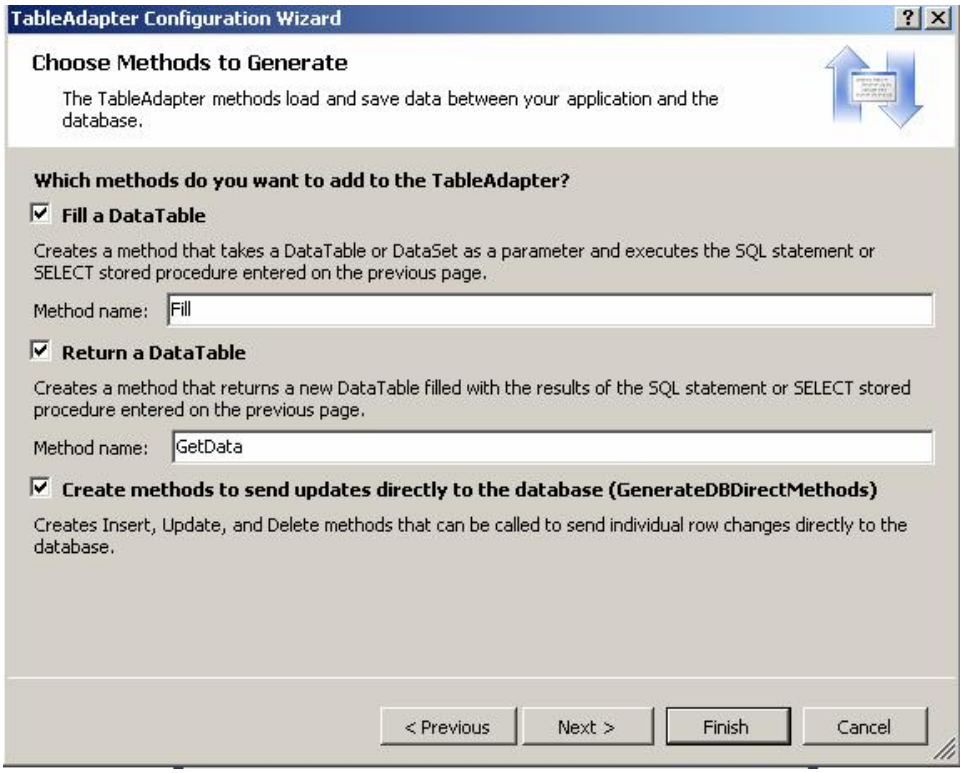
- أول نافذة في هذا المعالج، هي نافذة اختيار قاعدة البيانات المراد الاتصال بها، وقد تعرفنا عليها من قبل.. اختر قاعدة بيانات الكتب Books.mdf من الاتصالات المتاحة، أو أنشئ اتصالا جديدا بها، ثم اضغط الزر Next.

- النافذة التالية ستسألك إن كنت تريد حفظ نص الاتصال في إعـدادات البرنـامج.. وافق على هذا ودع الاسم الافتراضـي BooksConnectionString كمـا هـو، واضغط Next.
- النافذة التالية تتيح لك اختيار نوع أمر التحديـد.. اختـر Use SQL Statement واضغط Next.
- في النافذة التالية اكتب استعلام التحديد التالي:

SELECT * FROM Authors

واضغط Next.

- في النافذة التالية يمكنك اختيار الوسائل التي ستضاف إلى فئة مهيئ الجدول، كمـا هو موضح في الصورة:



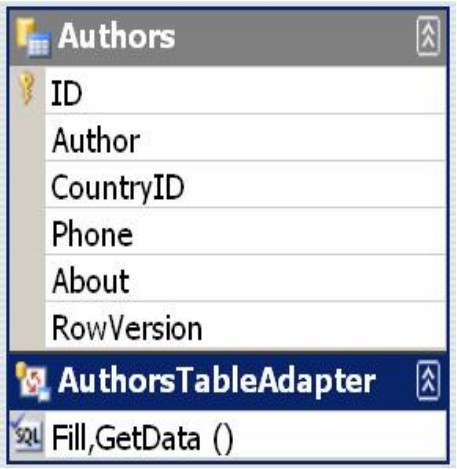
يمكنك أن تختار إنشاء وسيلة لملء جدول موجود في مجموعة البيانات، وسـيكون اسمها المبدئي Fill ويمكنك كتابة أي اسم آخر في مربع النص.

كما يمكنك أن تختار إنشاء وسيلة تعيد جدول بيانات DataTable مملوء بالنتائج، لتستخدمه أنت بالطريقة التي تناسبك، وسيكون اسم هذه الوسيلة مبدئيا GetData ويمكنك كتابة أي اسم آخر في مربع النص.

أما الاختيار الأخير، فيجعل مهيئ الجدول ينشئ الوسائل اللازمة لتحديث قاعدة البيانات.. هذه الوسائل ستحمل الأسماء Update و Insert و Delete.

بعد أن تحدد الاختيارات التي تناسبك، اضغط Next.

- ستظهر نافذة تلخص اختياراتك.. اضغط Finish لإنهاء المعالج السحري وإنشاء مهيئ الجدول.



ستجد في مصمم المخطط العنصر الموضح في الصورة.. هذا العنصر يمثل مخطط الجدول Authors، وفي الجزء السفلي منه مخطط مهيئ الجدول الذي سيستخدم للتعامل معه، واسمه الافتراضي AuthorsTableAdapter.

ويمكنك ضغط الشريط الذي يعرض اسم مهيئ الجدول بزر الفأرة الأيمن، واختيار الأمر Properties لعرض خصائص مهيئ الجدول في نافذة الخصائص.. وهذه الخصائص هي:

الاسم Name:

تحدد اسم فئة مهيئ الجدول.

المجال Modifier:

تحدد مجال فئة الجدول.. والقيمة الافتراضية هي Public ليمكن استخدام هذه الفئة حتى من خارج المشروع الحالي.

الفئة الأم Base Class:

تحدد الفئة الأم التي ترثها فئة مهيئ الجدول.. في الوضع الافتراضي تكون هذه الفئة هي فئة المكون System.ComponentModel.Component.. لكن لا مانع من أن تكتب بدلا منها أية فئة أخرى بشرط أن تكون مشتقة من الفئة Component.. يمكنك مثلا أن ترث فئة مهيئ البيانات، أو يمكنك أن ترث فئة مهيئ جدول آخر!

الاتصال Connection:

تحدد الاتصال بقاعدة البيانات.. ويمكنك اختيار اتصال من القائمة المنسدلة، أو ضغط العنصر الأخير فيها (New Connection) لإنشاء اتصال جديد.

مجال الاتصال ConnectionModifier:

تحدد مجال كائن الاتصال المعرف في مهيئ الجدول.. والقيمة الافتراضية هي Friend لجعله مرئيا من أي موضع في المشروع.

أمر التحديد SelectCommand:

أمر التحديد المستخدم لإحضار البيانات من قاعدة البيانات.. لاحظ أنك لو استخدمت أمر تحديد يعيد بيانات من أكثر من جدول، فسيعجز مهيئ الجدول عن إنتاج أوامر التحديث والإدراج والحذف آليا، لهذا يتوجب عليك في هذه الحالة استخدام الخصائص التالية لتعريف هذه الأوامر بنفسك.

أمر التحديث UpdateCommand:

أمر التحديث المستخدم لتحديث سجلات قاعدة البيانات.

أمر الإدراج InsertCommand:

أمر الإدراج المستخدم لإدراج السجلات في قاعدة البيانات.

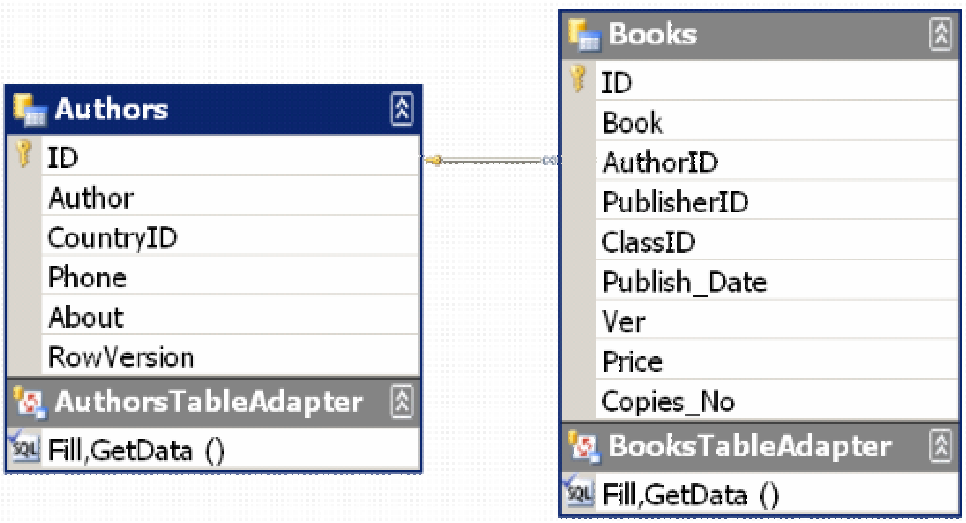
أمر الحذف DeleteCommand:

أمر الحذف المستخدم لحذف السجلات من قاعدة البيانات.

إنتاج وسائل قاعدة البيانات المباشرة GenerateDbDirectMethods:

إذا جعلت قيمة هذه الخاصية True، فستضاف إلى فئة مهيئ الجدول الوسائل Update و Insert و Delete لتتيح لك تحديث قاعدة البيانات باستدعائها مباشرة.. أما إذا جعلتها False، فسيكون عليك استخدام كائنات الأوامر بنفسك لإجراء عمليات التحديث والإدراج والحذف، وذلك من خلال الخصائص UpdateCommand، InsertCommand، DeleteCommand.

وبنفس الطريقة يمكنك إضافة مهيئ جدول الكتب.. لاحظ أن العلاقة بين الجدولين سيتم إنشاؤها تلقائيا بمجرد إضافة جدول الكتب.. هكذا سيكون المخطط:



ولكن، أين يتم إنشاء فئة مهيئ البيانات؟

يتم إنشاء هذه الفئة في نفس الملف الذي توجد فيه فئة مجموعة البيانات محددة النوع، وهو في مثالنا هذا الملف DsAuthorsBooks.Designer.cs.. لكن فئة مهيئ الجدول لا توضع داخل فئة مجموعة البيانات، بل توضع خارجها، ويتم تعريف نطاق اسم

Namespace خاص بها يكون على الصيغة XTableAdapters، حيث X هو اسم فئـــة عة البيانات.. هكذا مثلا سيكون الشكل العام لملف كود فئة البيانات ومهيئات الجداول فـــي مشروعنا هذا:

```
public partial class DsAuthorsBooks :System.Data.DataSet {
   // كود مجموعة البيانات محددة النوع
}

namespace DsAuthorsBooksTableAdapters {
   public partial class AuthorsTableAdapter : Component  {
      // كود مهيئ جدول المؤلفين
   }

   public partial class BooksTableAdapter : Component  {
      // كود مهيئ جدول الكتب
   }

   public partial class TableAdapterManager : Component {
      // كود مدير مهيئات الجداول
   }
}
```

وتمتلك فئة مهيئ الجدول الخصائص التالية:

الاتصال Connection:

تقرأ أو تغير كائن الاتصال الذي يستخدمه مهيئ الجدول للاتصال بقاعـــدة البيانـــات.. ويعتمد نوع هذه الخاصية على نوع قاعدة البيانات التي تتعامل معها، وفي مثالنا هـــذا ستكون من النوع SqlConnection.

ويستخدم مهيئ الجدول إجراءا خاصا Private Sub اسمه InitConnection لضبط خصائص كائن الاتصال.

الانتقالات Transaction:

تقرأ أو تغير كائن الانتقالات الذي يستخدمه مهيئ الجدول للتحكم في العمليات التي تتم على قاعدة البيانات.. وفي مثالنا هذا ستكون هذه الخاصية من النوع SqlTransaction.

محو قبل الملء ClearBeforeFill:

إذا جعلت قيمة هذه الخاصية True (وهي القيمة الافتراضية)، فسيتم محو السجلات من جدول مجموعة البيانات أولا، قبل ملئه بالنتائج الجديدة.. أما إذا جعلتها False، فسيتم تحديث السجلات الموجودة بالقيم الجديدة، وإضافة السجلات الجديدة إلى الجدول.

ملحوظة:
مهيئ الجدول يستخدم مهيئ البيانات داخليا لملء مجموعة البيانات، لهذا ستجد في فئة مهيئ الجدول خاصية محمية Protected Property اسمها Adapter، لن تراها من خارج الفئة، لكن تستطيع استخدامها في الفئات التي ترث فئة مهيئ الجدول، أو في أي كود إضافي تكتبه في مهيئ الجدول بنفسك.. ويستخدم مهيئ الجدول إجراء خاصا private void اسمه InitAdapter لوضع القيم في خصائص مهيئ البيانات. كما يحتوي مهيئ الجدول على خاصية محمية أخرى اسمها CommandCollection، وهي تعيد مصفوفة تحتوي على كائنات الأوامر التي يستخدمها مهيئ الجدول، وبهذا يستطيع مهيئ الجدول التعامل مع أكثر من استعلام كما سنرى لاحقا. ويستخدم مهيئ الجدول إجراء خاصا اسمه InitCommandCollection لوضع كائنات الأوامر في هذه المصفوفة وضبط خصائصها.

كما تمتلك فئة مهيئ الجدول الوسائل التالية:

ملء Fill:

تستقبل معاملا من نوع الجدول المراد ملؤه بالبيانات، وتعيد عددا صحيحا يخبرك بعدد السجلات التي أضيفت أو تم تحديثها في هذا الجدول.. وفي مهيئ جدول المؤلفين، يكون معامل هذه الوسيلة من نوع جدول المؤلفين المعرف في مجموعة البيانات DsAuthorsBooks.AuthorsDataTable، وبالمثل يكون هذا المعامل في مهيئ بيانات الكتب، من النوع DsAuthorsBooks.BooksDataTable.

لاحظ أنك تستطيع أن ترسل جدولا من جداول مجموعة البيانات إلى هذه الوسيلة، أو ترسل جدول حرا ليس مرتبطا بمجموعة بيانات، المهم أن يكون من النوع الصحيح.

قراءة البيانات GetData:

لا تستقبل أية معاملات، لكنها تعيد جدولا جديدا مملوءا بالبيانات.. هذا الجدول يكون من النوع AuthorsDataTable في مهيئ جدول المؤلفين، ومن النوع BooksDataTable في مهيئ بيانات الكتب.

تحديث Update:

تحفظ التغييرات في قاعدة البيانات.. لاحظ أن كل ما تفعله هذه الوسيلة، هو استدعاء الوسيلة Update الخاصة بمهيئ البيانات الداخلي.. ولهذه الوسيلة الصيغ التالية:

١- الصيغة الأولى تستقبل كائن الجدول المراد حفظ تغييراته.

٢- الصيغة الثانية تستقبل كائن مجموعة البيانات، حيث يقوم مهيئ الجدول بقراءة التغييرات من الجدول الخاص به في مجموعة البيانات، دون غيره من الجداول.. مثلا: تستخدم هذه الصيغة الكود التالي في مهيئ جدول المؤلفين:

```
return this.Adapter.Update(dataSet, "Authors");
```

٣- الصيغة الثالثة تستقبل كائن صف البيانات DataRow الذي تريد حفظ تغييراته في قاعدة البيانات.

٤- الصيغة الرابعة تستقبل مصفوفة تحتوي على صفوف البيانات التي تريد حفظ تغييراتها في قاعدة البيانات.

٥- الصيغة الخامسة تستقبل قيم الصف المراد حفظه في قاعدة البيانات.. ولهذه الصيغة عدة معاملات، كل منها يستقبل قيمة أحد الأعمدة الموجودة في الصف.. مثلا، ستحتوي هذه الوسيلة في مهيئ بيانات المؤلفين على هذه المعاملات

بالترتيب: Author، CountryID، Phone، About، Original_ID، Original_RowVersion.

٦- الصيغة السادسة تزيد على الصيغة السابقة بمعامل إضافي يستقبل المفتاح الأساسي للجدول (الحقل ID في مثالنا هذا).

وتعيد هذه الوسيلة عددا صحيحا يخبرك بعدد السجلات التي تم تحديثها في قاعدة البيانات، فإذا كان الناتج صفرا، فهذا معناه حدوث مشكلة تطابق Concurrency Violation.

إدراج Insert:

تدرج صفا جديدا في قاعدة البيانات.. ولهذه الوسيلة عددا من المعاملات بعدد أعمدة الجدول، لاستقبال قيم الصف المراد إضافته.

وتعيد هذه الوسيلة عددا صحيحا يخبرك بعدد السجلات التي أضيفت إلى قاعدة البيانات، فإذا كان الناتج صفرا، فهذا معناه أن قاعدة البيانات رفضت إدراج الصف بسبب وجود مشكلة في قيمة إحدى خاناته.

حذف Delete:

تحذف سجلا من قاعدة البيانات.. وتميز هذه الوسيلة السجل باستقبال مفتاحه الأساسي Original_ID وإصداره Original_RowVersion كمعاملين.. لاحظ أننا لا نستخدم إصدار السجل في جدول الكتب، لهذا تمتلك هذه الوسيلة في مهيئ جدول الكتب معاملات بعدد حقول الجدول، للبحث عن السجل الأصلي في قاعدة البيانات بدلالة كل قيمه.

وتعيد هذه الوسيلة عددا صحيحا يخبرك بعدد السجلات التي تم حذفها من قاعدة البيانات، فإذا كان الناتج صفرا، فهذا معناه حدوث مشكلة تطابق Concurrency Violation.

إضافة استعلامات جديدة إلى مهيئ الجدول:

إلى الآن، لا يبدو أن مهيئ الجدول يقدم شيئا جديدا يميزه عن مهيئ البيانات العادي.. فالحقيقة أن مزية مهيئ الجدول الرئيسية، هي قدرتك على إضافة أي عدد تريده من الاستعلامات إليه، ما دامت تلتزم بأحد الشرطين التاليين:

١- أن تعيد سجلات لها نفس تركيب الجدول الذي يتعامل معه مهيئ الجدول.. ليس من المنطقي مثلا أن تضيف إلى مهيئ جدول المؤلفين، استعلاما يعيد سجلات الكتب.

٢- أن تعيد قيمة منفردة Scalar Value.. يمكنك مثلا أن تضيف إلى مهيئ جدول المؤلفين استعلاما يعيد عدد المؤلفين، أو عدد كتب أحد المؤلفين.

ولإضافة استعلام جديد إلى مهيئ الجدول، اضغط اسم المهيئ في نافذة المصمم بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Add Query.. سيؤدي هذا إلى بدء المعالج السحري لتهيئة استعلام مهيئ الجدول:

TableAdapter Query Configuration Wizard

دعنا نستخدم هذا المعالج لإضافة استعلام إلى مهيئ جدول الكتب، يعيد كتب المؤلف الذي نريده:

- النافذة الأولى تسألك عن نوع الاستعلام الذي تريده (جملة SQL أم إجراء مخزن).. اختر Use SQL Statement واضغط Next.
- النافذة التالية تسألك عن نوع الاستعلام الذي تريده:

يمكنك الاختيار من بين الأنواع التالية:

أ. جملة استعلام تعيد صفوفا:

SELECT statement witch returns rows.

ب. جملة استعلام تعيد قيمة منفردة:

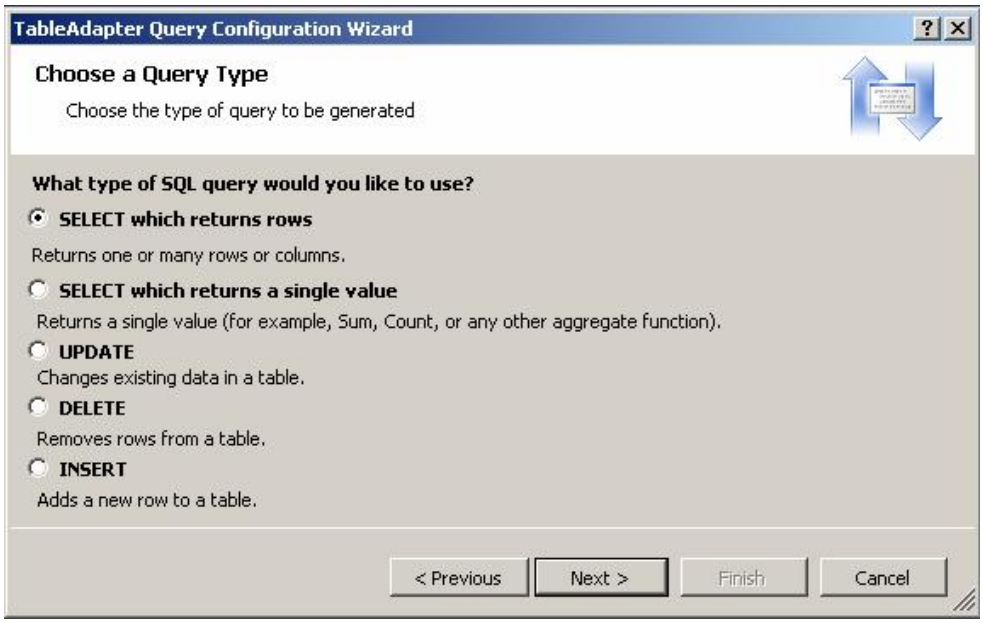
SELECT statement witch returns a single value.

ت. تحديث UPDATE.

ث. حذف DELETE.

ج. إدراج INSERT.

اختر أول اختيار، واضغط Next.



- في النافذة التالية اكتب جملة الاستعلام التالية:

```
SELECT     Books.*
FROM Books INNER JOIN Authors
ON Books.AuthorID = Authors.ID
AND Author = @Author
```

واضغط Next.

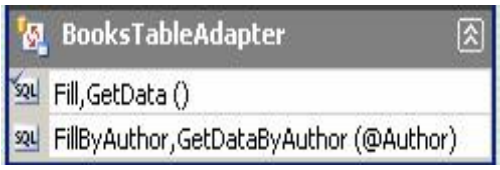
- النافذة التالية تتيح لك اختيار الوسائل التي ستضاف إلى مهيئ جدول الكتب لتنفيذ هذا الاستعلام.. ستجد وسيلتين هما:

أ. FillBy، وعليك تعديل اسمها إلى FillByAuthor، وهي تستقبل اسم المؤلف، وتملأ جدول الكتب في مجموعة البيانات بكتب هذا المؤلف.

ب. GetDataBy، وعليك تعديل اسمها إلى GetDataByAuthor، وهي تستقبل اسم المؤلف، وتعيد جدول كتب يحتوي على كتب هذا المؤلف.

- اضغط Next لعرض نافذة الملخص، ثم اضغط Finish لإنهاء المعالج السحري.

سيؤدي هذا إلى ظهور اسمي الوسيلتين الجديدتين في مخطط مهيئ جدول الكتب:



هذا معناه أن تعريف هاتين الوسيلتين قد أضيف إلى فئة مهيئ الجدول، وسيكون لكل منهما معامل نصي يستقبل اسم المؤلف.. وعموما، يقوم مهيئ جدول الكتب بتعريف المعاملات المناسبة لنوع الحقل الذي تستعلم عنه في قاعدة البيانات.

لاحظ أنك لو استخدمت استعلاما يعيد نتائج غير مرغوبة، فسيعرض لك مهيئ الجدول رسالة تحذرك من أن نتيجة الاستعلام لا تناسب مخطط الجدول.. ولو أردت تصحيح الاستعلام فاضغط بزر الفأرة الأيمن، فوق الصف الذي يعرض اسمي الوسيلتين الجديتين في مخطط مهيئ الجدول، ومن القائمة الموضعية اضغط الأمر Configure.. سيعرض هذا النافذة التي أدخلت فيها الاستعلام.. يمكنك تصحيحه كما تريد وضغط الزر Finish.

ولحذف الاستعلام، حدده في مخطط مهيئ الجدول، واضغط Delete.

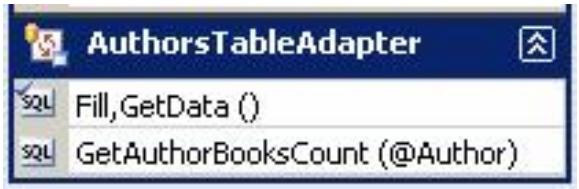
دعنا أيضا ننشئ استعلاما في مهيئ جدول الكتب يعيد لنا عدد كتب مؤلف معين.. دعنا نجرب طريقة أخرى هذه المرة.. من صندوق الأدوات اسحب العنصر Query وأسقطه فوق مهيئ جدول المؤلفين.. سيؤدي هذا إلى إطلاق المعالج السحري، حيث يمكنك اتباع نفس الخطوات السابقة، لكن مع اختيار:

SELECT Statement that returns a single value

وفي نافذة الاستعلام اكتب:

SELECT COUNT(BOOK) FROM Authors, Books
WHERE AuthorID = Authors.ID AND Author = @Author

واضغط Next.. ستظهر لك نافذة تتيح لك تسمية الدالة التي تنفذ هذا الاستعلام.. سيكون لهذه الدالة الاسم الافتراضي ScalarQuery.. غيره إلى GetAuthorBooksCount واضغط Finish.. سيظهر الاسم الجديد في مخطط مهيئ الجدول كما في الصورة، وستضاف هذه الوسيلة إلى فئة مهيئ الجدول، حيث ستستقبل نصا يمثل اسم المؤلف، وتعيد عددا صحيحا يمثل عدد كتبه.



ملحوظة ١:

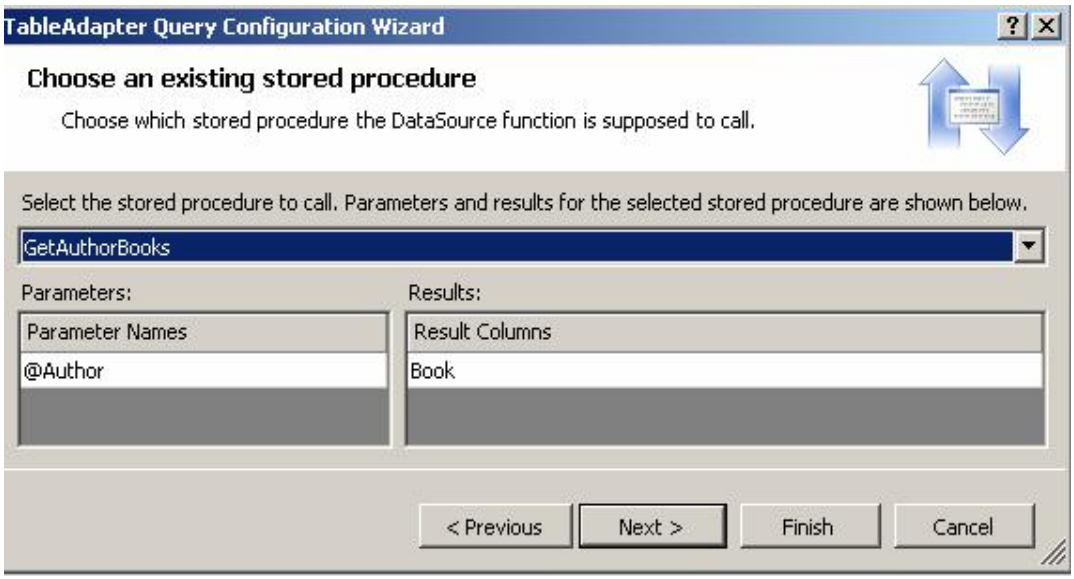
عند إنشاء الاستعلام عن حقل يمكن تركه فارغا (مثل الحقل Phone في جدول الكتب)، يقوم مهيئ البيانات بتعريف معامل الوسيلة FillBy بحيث يكون قابلا للانعدام Nullable (مثلا: سيكون معامل الوسيلة FillByPhone من نوع النص المنعدم String?).. هذا يتيح لك إرسال القيمة Nothing إلى هذه الوسيلة لتعيد إليك السجلات التي ما زال فيها هذا الحقل فارغا.

ملحوظة ٢:

إذا أردت إضافة وسيلة لتنفيذ إجراء مخزن، فاتبع نفس الخطوات المألوفة لإضافة استعلام، لكن هذه المرة اختر نوع الاستعلام:

Existing Stored Procedure

واضغط Next.. ستظهر نافذة تعرض قائمة منسدلة بها أسماء الإجراءات المخزنة في قاعدة البيانات.. اختر الإجراء GetAuthorBooks.. سيعرض النصف السفلي من النافذة بيانات هذا الإجراء المخزن: على اليسار ستظهر معاملات الإجراء، وعلى اليمين ستظهر الأعمدة الناتجة عن تنفيذه، كما هو موضح في الصورة:



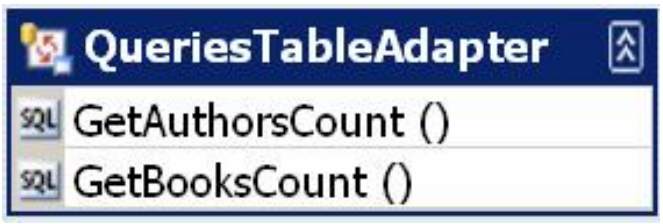
اضغط Next للانتقال إلى النافذة التالية، وهي تسألك عن القيمة العائدة من الوسيلة التي ستنفذ الإجراء المخزن.. هذه القيمة قد تكون:

- قيمة جدولية Tabular Value، حيث تعيد الوسيلة كائن جدول يحتوي على الصفوف الناتجة.
- قيمة منفردة Single Value، حيث تعيد الوسيلة قيمة أول خانة في أول عمود في النتيجة.
- ولا قيمة No Value، حيث ستكون الوسيلة بدون قيمة عائدة، وهذا مناسب للإجراءات المخزنة التي لا تعيد ناتجا.

اختر ما يناسبك واضغط Next.. باقي الخطوات لا جديد فيها.

إنشاء استعلامات عامة Global Queries:

يمكنك إنشاء مهيئ جدول لتنفيذ استعلامات عامة، كحساب دالة تجميع، أو تنفيذ استعلامات الحذف والإدراج دون أن يكون مرتبطا بجدول معين في مجموعة البيانات.. لفعل هذا، اضغط بزر الفأرة الأيمن في أي منطقة خالية من مصمم مجموعة البيانات، ومن القائمة الفرعية Add اضغط الأمر Query.. أو اسحب العنصر Query من صندوق الأدوات وألقه على أي منطقة فارغة من مصمم مجموعة البيانات.. سيؤدي هذا إلى إطلاق المعالج السحري لتهيئة الاستعلام، لكنه سيبدأ هذه المرة بنافذة اختيار الاتصال بقاعدة البيانات، ثم يستمر بنفس الخطوات السابقة، لكنك لن تستطيع إنشاء استعلام يعيد سجلات بهذا المعالج.. يمكنك فقط إنشاء استعلامات تعيد قيما منفردة، أو استعلامات التحديث والحذف والإدراج.. وبعد أن تنهي المعالج، ستجد مهيئ جدول جديد قد أضيف إلى مجموعة البيانات، وسيكون اسمه QueriesTableAdapter وهو اسم لا يمكنك تغييره.. وأية استعلامات عامة أخرى ستنشئها ستضاف إلى هذا المهيئ.، وقد أضفنا إليه في مشروعنا هذا الدالة GetAuthorsCount التي تعيد عدد المؤلفين، والدالة GetBooksCount التي تعيد عدد الكتب.

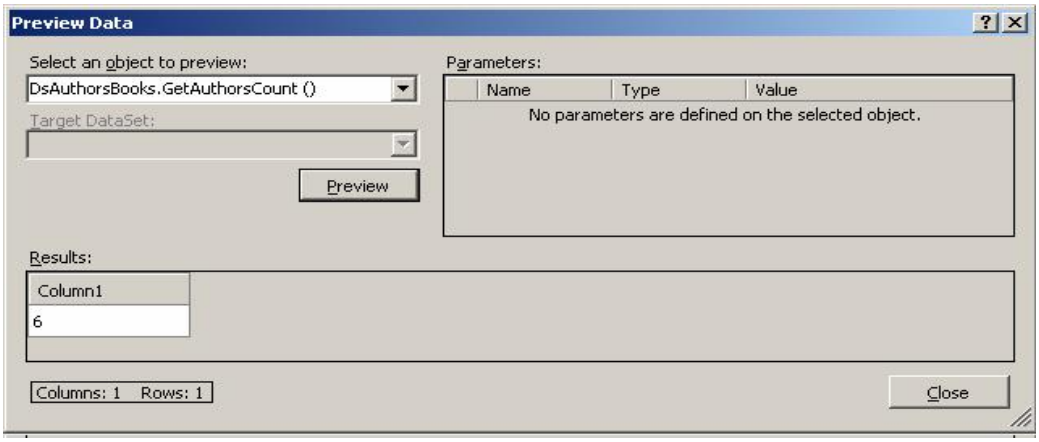


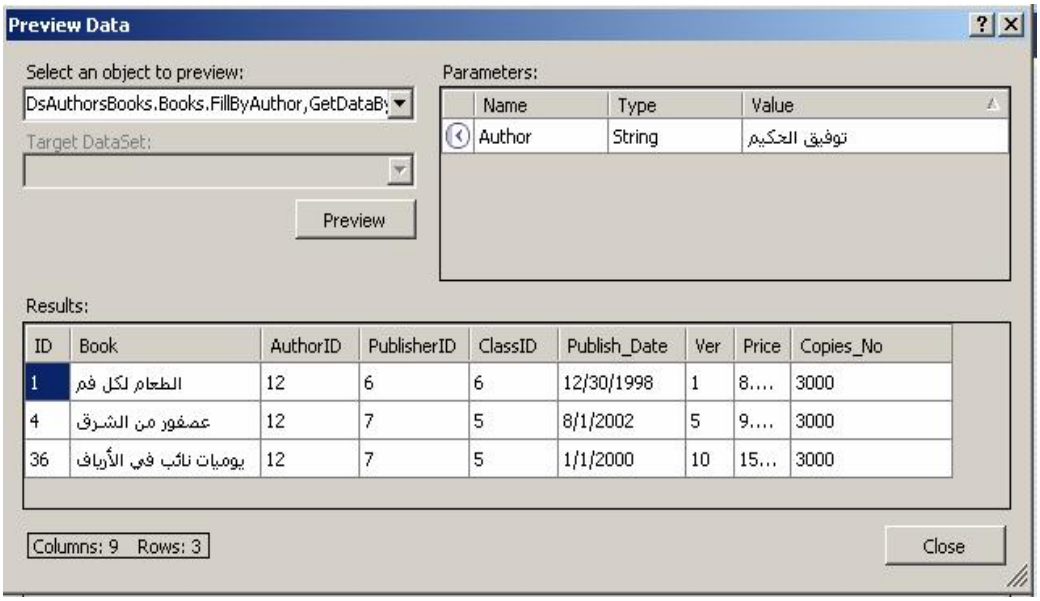
تذكر مرة أخرى، أن مهيئ جدول الاستعلامات لا يتعامل مع استعلامات أو إجراءات مخزنة تعيد سجلات.. هو فقط يتعامل مع استعلامات أو إجراءات مخزنة تعيد قيما مفردة، وإذا اخترت إجراءا مخزنا يعيد سجلات، فستعيد هذه الوسيلة قيمة أول خانة في أول صف في النتيجة!

معاينة بيانات مجموعة البيانات:

يمكنك معاينة نتيجة أي دالة في مهيئ الجدول، بالضغط بزر الفأرة الأيمن في أي موضع خال في مصمم مجموعة البيانات، أو فوق تصميم أحد الجداول أو أحد مهيئات الجداول، وضغط الأمر Preview Data من القائمة الموضعية.. وستجد نفس الأمر في القائمة الرئيسية Data.

ستظهر نافذة عرض النتائج، حيث يمكنك إسدال القائمة العلوية، واختيار مهيئ الجدول الذي تريده، والدالة التي تريد تنفيذها منه، ثم وضع المعاملات في الجدول الموجود على يمين النافذة إن كانت الدالة تحتاج معاملات، ثم ضغط الزر Preview لرؤية النتيجة.





فئة مدير مهيئات الجداول TableAdapterManager

ظهرت هذه الفئة في دوت نت ٢٠٠٨، ويتم إنتاجها آليا عند إنشاء مهيئات الجداول التي تربطها علاقات، لتسمح لك بإجراء التحديث المتراكب Hierarchical Update، وفيه يتم تحديث الجداول المترابطة معا، مع قدرتك على تحديد الترتيب الصحيح لإجراء عمليات التحديث، لمراعاة القيود المفروضة على الجداول.
لاحظ أنك تستطيع منع إنتاج هذه الفئة.. لفعل هذا اضغط بزر الفارة الأيمن في أي موضع فارغ من مخطط مجموعة البيانات، ومن القائمة الموضعية اضغط الأمر Properties، وفي نافذة الخصائص غير قيمة الخاصية Hierarchical Update إلى False.

وتمتلك هذه الفئة الخصائص التالية:

الاتصال Connection:

تقرأ أو تغير كائن الاتصال المستخدم في عملية التحديث.

مدير مهيئ الجدول... XTableAdapter:

الحرف X الذي وضعناه في بداية اسم هذه الخاصية هو بديل عن اسم أحد الجداول.. هذا معناه أن مدير مهيئات الجداول يمتلك خاصية لكل مهيئ جدول تم تعريفه في مجموعة البيانات.. وفي مشروعنا هذا ستحتوي فئة المدير على الخاصيتين AuthorsTableAdapter و BooksTableAdapter.
وتكون لهذه الخصائص القيمة Nothing إلى أن تضع في كل منها مهيئ الجدول الذي تريد أن يتحكم فيه مدير المهيئات.

عدد نسخ مهيئات الجداول TableAdapterInstanceCount:

تعيد عدد نسخ مهيئات الجداول التي لها قيمة غير Nothing.

نَسخ مجموعة البيانات احتياطيا قبل التحديث BackUpDataSetBeforeUpdate:

إذا جعلت قيمة هذه الخاصية True، فسيتم حفظ نسخة احتياطية من مجموعة البيانات قبل إجراء عملية التحديث.. يحدث هذا بتعريف مجموعة بيانات داخل إجراء التحديث، وحفظ سجلات مجموعة البيانات الأصلية فيها.. هذا مفيد إذا حدث خطأ

أثناء عملية التحديث، ففي هذه الحالة سيتم التراجع Rollback عن كل العمليات التي أجريت على قاعدة البيانات، وستستعيد مجموعة البيانات الأصلية حالتها السابقة قبل إجراء عملية التحديث، وذلك باستعادتها من مجموعة البيانات الاحتياطية.. لاحظ أن أخذ نسخة احتياطية من مجموعة بيانات ضخمة سيكون عبئا على الذاكرة وسيستهلك وقتا لتنفيذه، لهذا فالقيمة الافتراضية لهذه الخاصية هي False.

ترتيب التحديث UpdateOrder:

تحدد ترتيب تنفيذ أوامر التحديث والإدراج والحذف عند إجراء عملية التحديث، وهي تأخذ إحدى قيمتي المرقم UpdateOrderOption التاليتين:

InsertUpdateDelete	تنفيذ أوامر الإدراج ثم التحديث ثم الحذف.. هذه القيمة الافتراضية.
UpdateInsertDelete	تنفيذ أوامر التحديث ثم الإدراج ثم الحذف.

وتمتلك هذه الفئة الوسيلة الوحيدة التالية:

تحديث الكل UpdateAll:

أرسل إلى هذه الوسيلة مجموعة البيانات محددة النوع التي تريد نقل التغييرات منها إلى قاعدة البيانات.. في مشروعنا ستكون مجموعة البيانات من النوع DsAuthorsBooks.. ويتم إجراء عمليات التحديث بالترتيب الموضح في الخاصية UpdateOrder، وإذا حدث خطأ في أي مرحلة من مراحل التحديث، يتم التراجع Rollback عن تنفيذ جميع عمليات التحديث، أي أن قاعدة البيانات لا يحدث بها أي تغيير، وتظل كما كانت قبل استدعاء هذه الوسيلة.

إضافة أكواد خاصة بك إلى مجموعة البيانات والجداول ومهيئات الجداول:

يمكنك أن تضيف بعض الوسائل إلى فئة مجموعة البيانات، أو فئة الجدول، أو فئة مهيئ الجدول.. كما يمكنك كتابة إجراءات تستجيب لبعض أحداث فئة الجدول، سواء الأحداث المعرفة داخل فئة الجدول، أو تلك الموروثة من الفئة DataTable، والتي سنتعرف عليها في الفصل التالي.

لكن المشكلة أنك لو كتبت أي كود في الملف X.Designer.cs الذي فيه تعريف هذه الفئات (حيث X هو اسم مجموعة البيانات)، فسيكون هذا الكود عرضة للضياع عند قيامك بأي تعديلات في مصمم مجموعة البيانات، لأن هذه التعديلات ستعيد إنتاج ملف الكود من جديد، وستتخلص من أي كود خاص بك!

لحل هذه المشكلة، تم تعريف الفئات في هذا الملف باعتبارها جزئية Partial، ليمكنك إضافة الكود إليها في ملف آخر.. لفعل هذا، اضغط بزر الفأرة الأيمن فوق الجدول أو مهيئ الجدول، ومن القائمة الموضعية اختر الأمر View Code، لفتح تعريف جزئي مستقل لفئة الجدول أو فئة مهيئ الجدول.. هذا التعريف سيضاف في ملف جديد اسمه X.cs (في مثالنا هذا سيكون اسمه DsAuthorsBooks.cs).. وستجد هذا الملف ضمن الملفات الفرعية لمخطط مجموعة البيانات DsAuthorsBooks.xsd.

وعندما تفتح هذا الملف في محرر الكود، يمكنك اختيار الفئة من القائمة العلوية اليسرى، واختيار الحدث الذي تريد إضافته إليها من القائمة العلوية اليمنى كما هو مألوف.. كما يمكنك أن تضيف أية دالة تريدها إلى أية فئة، سواء كانت خاصة Private أو عامة Public، مع قدرتك على استخدام كل العناصر المعرفة على مستوى الفئة في كتابة كود هذه الدالة، سواء كانت هذه العناصر محمية Protected أو خاصة Private.

كما يقدم لك مصمم مجموعة البيانات الكثير من التسهيلات:

- فالنقر مرتين في أية منطقة خالية، يفتح فئة مجموعة البيانات.
- والنقر مرتين على عنوان الجدول يفتح فئة الجدول، ويضيف إليها مستجيبا للحدث XRowChanging، حيث X هو اسم الجدول.. وقد استخدمنا الحدث

AuthorsRowDeleting في المشروع TableAdapter لعرض رسالة تأكيد قبل حذف أي صف من الجدول.

- والنقر مرتين على أي صف في الجدول، يفتح فئة الجدول، ويضيف إليها تعريفا للحدث ColumnChanging، وشرطا يتأكد أن العمود الذي تغير هو العمود الذي نقرته بالفأرة.. وقد استخدمنا هذا الحدث في المشروع TableAdapter لمنع المستخدم من ترك اسم المؤلف فارغا.
- والنقر مرتين على عنوان مهيئ الجدول يفتح كود فئته.

استخدام مهيئ الجدول في الكود:

لاستخدام مهيئ الجدول يجب أن تعرف نسخة منه.. مثال:

```
var TaAuhtors = new
   DsAuthorsBooksTableAdapters.AuthorsTableAdapter( );
MessageBox.Show(TaAuhtors.GetAuthorBooksCount("
   توفيق الحكيم"));
```

ولاستخدام مدير المهيئات في الكود، يجب أن نعرف نسخة منه، وتضع نسخة من كل مهيئ جدول في الخاصية المناظرة له في مدير المهيئات، كالتالي:

```
var TaBooks = new
  DsAuthorsBooksTableAdapters.BooksTableAdapter( );
var TaM = new
  DsAuthorsBooksTableAdapters.TableAdapterManager( );
TaM.AuthorsTableAdapter = TaAuhtors;
TaM.BooksTableAdapter = TaBooks;
```

لكن الأسهل هو أن تتعامل مع هذه الكائنات بشكل مرئي، حيث تقدم لك دوت نت هذه التسهيلات:

١- عند سحب مجموعة البيانات من صندوق الأدوات وإسقاطها على النموذج، ستظهر نافذة تسألك إن كنت تريد إضافة مجموعة بيانات عادية أم محددة النوع، حيث تستطيع اختيار النوع X.DsAuthorsBooks من القائمة المنسدلة، حيث X هو اسم المشروع (وهو TableAdapter في حالتنا هذه).

٢- يمكنك التعامل مع عناصر مجموعة البيانات محددة النوع من صندوق الأدوات مباشرة، فهي يظهر في صندوق الأدوات تحت شريط جديد اسمه X Components، حيث X هو اسم المشروع.. وإذا لم تجد هذا الشريط، فأغلق صندوق الأدوات ثم أعد فتحه ليتم إنعاشه.. وستجد تحت شريط المشروع TableAdapter العناصر التالية:

أ. مجموعة البيانات محددة النوع DsAuthorsBooks.

ب. مهيئ جدول المؤلفين AuthorsTableAdapter.

ج. مهيئ جدول الكتب BooksTableAdapter.

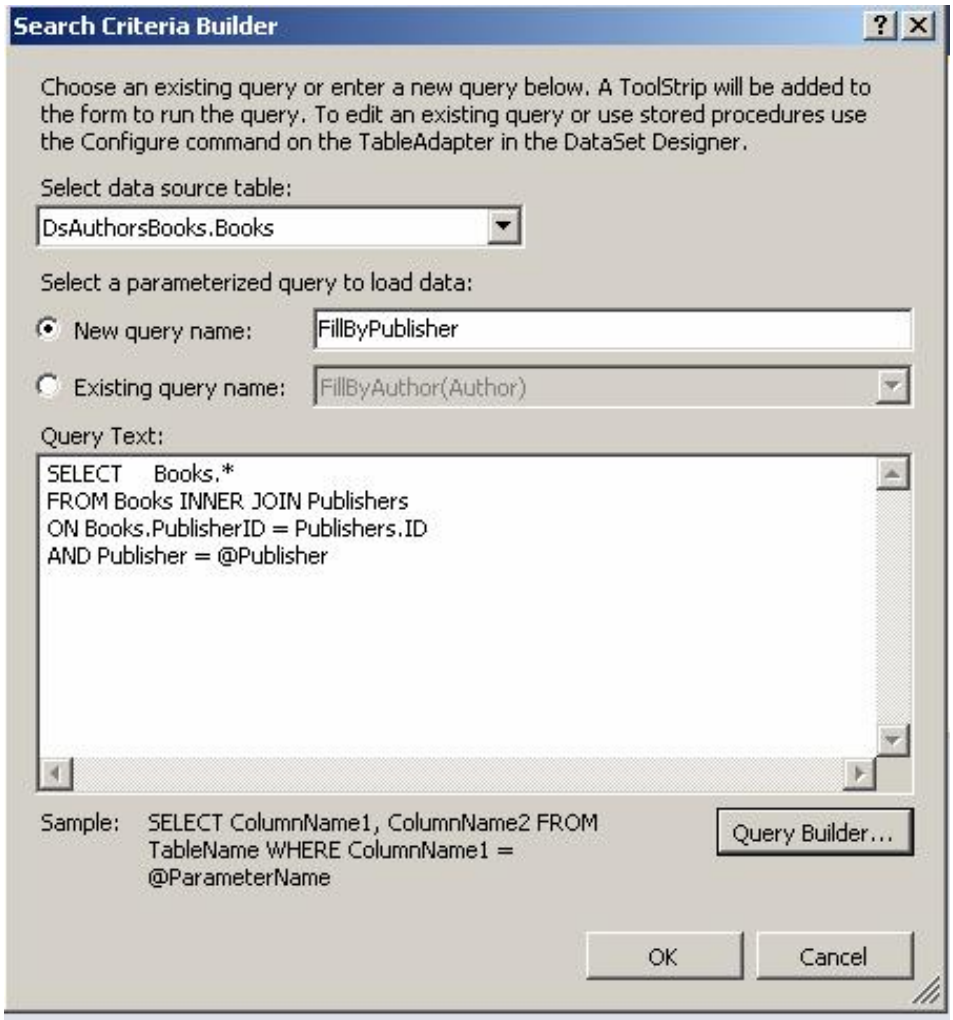
د. مهيئ جدول الاستعلامات QueriesTableAdapter.

هـ. مدير مهيئات الجداول TableAdapterManger.

هذا يتيح لك إضافة أي من هذه العناصر إلى النموذج، حيث ستظهر في صينية المكونات.. أضف نسخة من كل عنصر من هذه العناصر، وامنحها الأسماء التالية على الترتيب: Ds، TaAuthors، TaBooks، TaQueries، TaM.

٣- اضغط بزر الفأرة الأيمن، مدير المهيئات TaM في صينية المكونات، ومن القائمة الموضعية اضغط الأمر Properties لعرض خصائصه في نافذة الخصائص.. من القائمة المنسدلة للخاصية AuthorsTableAdapter اختر TaAuthors، ومن من القائمة المنسدلة للخاصية BoosTableAdapter اختر TaBooks.. ويمكنك تغيير قيمة باقي الخصائص كما يناسبك.

٤- اضغط بزر الفأرة الأيمن، أي مهيئ جدول في صينية المكونات، ومن القائمة الموضعية اضغط الأمر Add Query لإضافة استعلام جديد إلى مهيئ الجدول.. في هذه الحالة ستظهر نافذة مختصرة، تتيح لك تعريف الوسيلة FillBy فقط ولن يتم تعريف الوسيلة GetDataBy، كما هو موضح في الصورة:



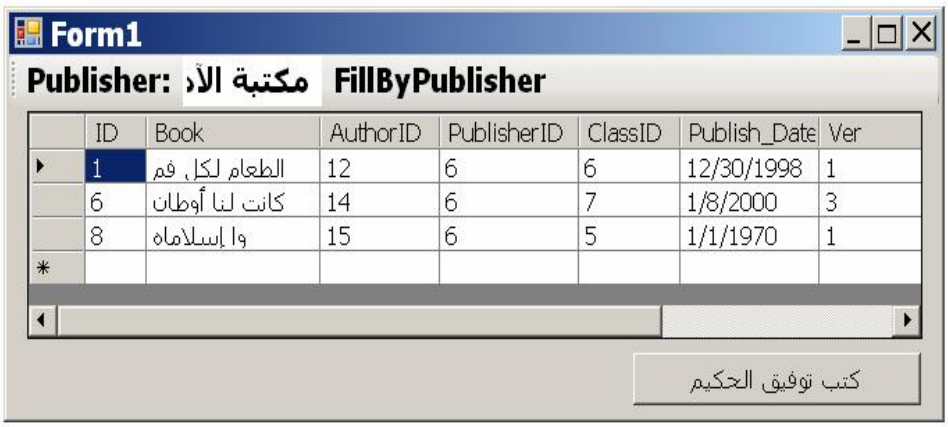
ويمكنك أن تغير مهيئ الجدول الذي تتعامل معه من القائمة المنسدلة، ويمكنك أن تغير اسم وسيلة الملء بتحريرها في مربع النص (وسنستخدم هنا الاسم FillByPublisher)، كما يمكنك أن تكتب الاستعلام في مربع النص السفلي.. وإذا أردت تعديل استعلام موجود سابقا، فاضغط الاختيار Existing Query Name واختر اسم الاستعلام من القائمة المنسدلة لعرضه في مربع النص السفلي.. في حالتنا هذه، سنستخدم استعلاما جديدا للحصول على الكتب التي نشرها ناشر معين.

اضغط Ok لإغلاق هذه النافذة.. سيؤدي هذا إضافة الوسيلة FillByPublisher إلى مهيئ بيانات الكتب.

وإذا أردت إضافة الوسيلة GetDataByPublisher، فاتبع الخطوات التالية:

- ضغط مهيئ الجدول بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Edit Query In DataSet لعرض مخطط مجموعة البيانات.
- اضغط الاستعلام FillByPublisher في مهيئ جدول الكتب بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Configure لعرض نافذة تحرير الاستعلام.
- اضغط Next لعرض نافذة وسائل الاستعلام، وضع علامة الاختيار أمام Return a DataTable، وغير اسم الوسيلة إلى GetDataByPublisher واضغط Finish.

أيضا، سيضاف رف أدوات ToolStrip إلى النموذج، عليه لافتة تحمل الاسم Publisher (وهو اسم المعامل المراد إدخاله لتنفيذ الاستعلام) ومربع نص ليكتب فيه المستخدم اسم الناشر، وزرا يحمل اسم الوسيلة FillByPublisher، وعند الضغط عليه سيتم ملء مجموعة البيانات بكتب هذا الناشر، فالكود الذي يفعل هذا تم إنتاجه آليا في حدث ضغط الزر.. لكن سيتبقى عليك أن تعرض محتويات مجموعة البيانات للمستخدم، وقد فعلنا هذا بعرضها في جدول عرض كما هو موضح في الصورة:



وتستطيع تغيير عنوان اللافتة والزر، وعرض رف الأدوات من اليمين إلى اليسار

والمشروع TableAdapter يريك أمثلة على استخدام مهيئات الجداول، مع استخدام مـــدير المهيئات في حدث ضغط الزر "حفظ التغييرات" لإرسال التغييرات التي أجراها المســـتخدم على السجلات إلى قاعدة البيانات.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

-١٢-

الجداول والعلاقات والقيود

سنتعرف في هذا الفصل على الكائنات الداخلة في تكـوين مجموعـة البيانـات DataSet، وهي:

- كائن الجدول DataTable والكائنات الداخلية المكونة لـه مثـل كـائن الصـف DataRow وكائن العمود DataCoulmn.
- كائن العلاقة DataRelation.
- كائنات القيود Constraints المختلفة.

إضافة إلى المجموعات التي تستخدمها هذه الفئات.

فئة أساس مجموعة البيانات الداخلية
InternalDataCollectionBase Class

هذه الفئة تمثل واجهة المجموعة ICollection، ولا تزيد على خصائصها ووسائلها بشيء جديد.

وهذه الفئة هي الفئة الأم لكل من المجموعات التالية:

١. DataTableCollection Class
٢. DataColumnCollection Class
٣. DataRowCollection Class
٤. DataRelationCollection Class
٥. ConstraintCollection Class

فئة مجموعة الجداول DataTableCollection Class

هذه المجموعة ترث الفئة InternalDataCollectionBase، وهي تحتوي على عناصر من النوع DataTable.. ويمكن الحصول على هذه المجموعة باستخدام الخاصيّة DataSet.Tables.

والكود التالي يعرض كلّ أسماء الجداول الموجودة في مجموعة البيانات Ds:

```
foreach (DataTable Tbl in Ds.Tables)
{
   MessageBox.Show(Tbl.TableName);
}
```

ولا تضيف هذه المجموعة جديدا إلى خصائص ووسائل واجهة المجموعة ICollection، ولكنها تضيف بعض الصيغ إلى بعض هذه العناصر، مثل:

المفهرس Indexer:

تعيد كائن البيانات DataTable الموجود في موضع معين في القائمة.. ولهذه الخاصية ثلاث صيغ:

١- الصيغة الأولى تستقبل رقم الجدول في المجموعة.

٢- والصيغة الثانية تستقبل نصا يمثل اسم الجدل.

٣- والصيغة الثالثة تزيد على الصيغة السابقة بمعامل ثان، يستقبل نصا يمثل اسم النطاق Name Space الذي يوجد تحته الجدول في مجموعة البيانات.

وتعيد هذه الخاصية null إذا لم تجد الجدول المطلوب في المجموعة.

إضافة Add:

تضيف جدولا إلى مجموعة البيانات، ولها أربع صيغ:

١- الصيغة الأولى بدون معاملات، وهي تنشئ جدولا باسم افتراضيّ (Table1 أو Table2 وهكذا...) وتضيفه إلى مجموعة البيانات.

٢- الصيغة الثانية تستقبل معاملا نصيّا، هو اسم الجدول الذي سيتمّ إنشاؤه وإضافته إلى المجموعة.. ولو أرسلت إلى هذا المعامل نصا فارغا ""، فسيسمى الجدول بالاسم الافتراضي Table1 أو Table2 وهكذا... لاحظ أن إضافة جدول بنفس اسم جدول موجود سابقا سيؤدي إلى حدوث خطأ.

٣- الصيغة الثالثة تزيد على الصيغة السابقة بمعامل ثان، يوضح نطاق الاسم الذي سيضاف إليه الجدول داخل مجموعة البيانات.. هذا يتيح لك إضافة أكثر من جدول بنفس الاسم إلى مجموعة الجداول، لكن كلا منها ينتمي إلى نطاق مختلف.. هذا مفيد عندما تملأ مجموعة البيانات بجداول متشابهة الأسماء من أكثر من قاعدة بيانات.. مثلا:

```
Ds.Tables.Add("MyTable", "Db1");
Ds.Tables.Add("MyTable", "Db2");
```

٤- والصيغة الرابعة تستقبل كائن جدول DataTable، لتتم إضافته إلى المجموعة.

لاحظ أن الصيغ الثلاث الأولى تعيد كائنا من النوع DataTable يمثل الجدول الذي تمّ إنشاؤه، بينما الصيغة الرابعة هي إجراء لا يعيد أية قيمة، وذلك لأنك أرسلت إليها كائن الجدول بالفعل، ولا تحتاج إلى مرجع آخر له.

يمكن حذفه CanRemove:

تعيد true إذا كان من الممكن حذف كائن الجدول DataTable المرسل كمعامل من مجموعة الجداول.. وتعيد هذه الوسيلة false إذا لم يكن الجدول موجودا في المجموعة، أو كان داخلا في علاقة.. والكود التالي سيعيد false لأن حذف جدول المؤلفين سيدمر تكامل العلاقة مع جدول الكتب:

```
MessageBox.Show(Ds.Tables.CanRemove(
        Ds.Tables["Authors"]).ToString( ))
```

حذف Remove:

تحذف الجدول المرسل إليها كمعامل، ولها نفس صيغ الوسيلة Add ما عدا الصيغة الأولى التي بدون معاملات.. لاحظ أنك لا تستطيع حذف جدول من مجموعة البيانات إذا كان داخلا في علاقة، فهذا سيؤدي إلى حدوث خطأ في البرنامج.. والمقصود بالعلاقة هنا، العلاقة المعرفة في مجموعة البيانات، فلو وضعت جدولي المؤلفين والكتب في مجموعة البيانات بدون إنشاء علاقة بينهما، فسيمكنك حذف أيهما بدون مشاكل رغم أن هناك علاقة بينهما فعلا في قاعدة البيانات.. أما لو أنشأت العلاقة بينهما في مجموعة البيانات، فعليك حذفها أولا قبل محاولة حذف أي من الجدولين.. والأفضل استخدام الوسيلة CanRemove أولا قبل محاولة حذف الجدول.. مثال:

```
var T = Ds.Tables["Authors"];
if (Ds.Tables.CanRemove(T))
   Ds.Tables.Remove(T);
```

تحتوي على Contains:

تعيد true إذا كان الجدول المرسل كمعامل موجودا في مجموعة البيانات.. ولهذه الوسيلة صيغتان، تماثلان الصيغتين الثانية والثالثة للوسيلة Add.

رقم العنصر IndexOf:

تعيد رقم الجدول المرسل إليها كمعامل إذا كان موجودا في المجموعة، وتعيد -١ إن لم يكن موجودا، ولها نفس صيغ الوسيلة Add ما عدا الصيغة الأولى التي بدون معاملات.

كما تمتلك مجموعة الجداول حدثين جديدين، هما:

المجموعة تتغير CollectionChanging:

ينطلق عندما توشك جداول المجموعة على التغير، نتيجة إضافة أو حذف جدول.. والمعامل الثاني e لهذا الحدث من النوع CollectionChangeEventArgs، وهو يمتلك الخاصيتين التاليتين:

Action	تعيد إحدى قيم المرقم CollectionChangeAction التي توضح نوع الفعل الذي سبب تغير المجموعة.. وهذه القيم هي: - Add: إضافة عنصر إلى المجموعة. - Remove: حذف عنصر من المجموعة. - Refresh: تغير عناصر المجموعة كلها، بسبب بعض الوسائل مثل Clear التي تمحو كل العناصر.	
Element	تعيد كائنا Object يحتوي على العنصر الذي تعرض للتغيير.. لاحظ أن قيمة هذا العنصر ستكون null إذا كانت للخاصية Action القيمة Refresh.	

المجموعة تغيرت CollectionChanged:

ينطلق بعد حدوث التغيير فعليا في عناصر المجموعة.. والمعامل الثاني e لهذا الحدث من النوع CollectionChangeEventArgs الذي تعرفنا عليه في الحدث السابق.

فئة جدول البيانات DataTable Class

تعمل هذه الفئة كوعاء لأحد الجداول بما فيه من أعمدة وصفوف، وهي تمثل الواجهة IListSource، كما أنها ترث الفئة MarshalByValueComponent، مما يتيح لك إضافتها إلى صينية مكونات النموذج، وإن كان عليك أن تضيفها أولا إلى صندوق الأدوات.. لكن لا داعي لهذا، فأنت تستطيع التعامل مع الجدول بطريقة مرئية في وقت التصميم، بعرض خصائص مجموعة البيانات غير محددة النوع Un-typed DataSet في نافذة الخصائص، واستخدام الخاصية Tables لإضافة الجداول وتغيير خصائصها بطريقة مرئية.. أما إذا كنت تتعامل مع مجموعة بيانات محددة النوع Typed DataSet، فيمكنك إضافة الجداول مباشرة إلى مخطط XML بالطرق التي تعرفنا عليها في الفصل السابق.

ولحدث إنشاء هذه الفئة أربع صيغ:

١- الصيغة الأولى بدون معاملات.

٢- والصيغة الثانية تستقبل معاملا نصيّا، هو اسم الجدول.

٣- والصيغة الثالثة تزيد على الصيغة السابقة بمعامل ثان، يوضح نطاق الاسم الذي سيضاف إليه الجدول داخل مجموعة البيانات.

٤- والصيغة الرابعة تستقبل معاملين من النوعين SerializationInfo و StreamingContext لاستخدامها في سلسلة الجدول Serialization.. هذا الموضوع خارج نطاق هذا الكتاب.

والمثال التالي يعرف كائن جدول ويضع فيه أول جدول في مجموعة جداول مجموعة البيانات:

```
DataTable T = Ds.Tables[0];
```

والمثال التالي يعرف كائن جدول جديد ويضيفه إلى مجموعة البيانات:

```
DataTable T = new DataTable("MyTable");
Ds.Tables.Add(T);
```

ويمتلك كائن الجدول الخصائص التالية:

اسم الجدول TableName:

تقرأ أو تغير اسم الجدول في مجموعة البيانات.

نطاق الاسم Namespace:

تحدد اسم النطاق الذي سيندرج تحته الجدول.. هذا يسمح بوجود أكثر من جدول بنفس الاسم في مجموعة البيانات، إذا كان كل منها في نطاق مختلف.

البادئة Prefix:

تحدد البادئة التي ستميز الجدول كاختصار لاسم نطاقه.

مجموعة البيانات DataSet:

تعيد كائن مجموعة البيانات DataSet التي ينتمي إليها هذا الجدول.. وتعيد null إذا لم يكن الجدول مضافا إلى مجموعة بيانات حاليا.

تنسيق التعامل عن بعد RemotingFormat:

تحدد التنسيق الذي سيتم به إرسال بيانات الجدول من جهاز إلى آخر، عندما التعامل مع برنامج يستخدم التحكم عن بعد Remoting، وهي تأخذ إحدى قيمتي المرقم SerializationFormat اللتين تعرفنا عليهما سابقا.

حساس لحالة الأحرف CaseSensitive:

لو جعلت هذه الخاصيّة true، فستتمّ مراعاة حالة الحروف (صغيرة Small أو كبيرة Capital) عند مقارنة النصوص في هذا الجدول.

المحل Locale:

تقرأ أو تغير كائن معلومات الثقافة CultureInfo، الذي يمثل اللغة التي تريد استخدامها لمقارنة وترتيب النصوص.

أقل سعة MinimumCapacity:

تحدد أقل عدد من السجلات يمكن أن يحتويه الجدول.. والقيمة الافتراضية لهذه الخاصية هي ٥٠ سجلا.. وتفيدك هذه الخاصية في حجز الذاكرة اللازمة عند التعامل مع جداول ضخمة الحجم، لتحسين كفاءة البرنامج، فأنت تحدد سعة مبدئية كبيرة لحجز الذاكرة المناسبة، بدلا من أن يتم حجز ذاكرة صغيرة، ثم يضطر البرنامج إلى زيادتها أكثر من مرة أثناء تشغيله.

الأعمدة Columns:

تعيد مجموعة الأعمدة DataColumnCollection الموجودة في هذا الجدول.. وسنتعرف على هذه المجموعة بالتفصيل لاحقا.

الصفوف Rows:

تعيد مجموعة الصفوف DataRowCollection الموجودة في هذا الجدول.. وسنتعرف على هذه المجموعة بالتفصيل لاحقا.

العلاقات الرئيسية ParentRelations:

تعيد نسخة من مجموعة العلاقات DataRelationCollection تحتوي على العلاقات الخارجة من هذا الجدول (العلاقات التي يدخل فيها كجدول رئيسيّ Master Table).. وسنتعرف على الفئة DataRelationCollection بالتفصيل لاحقا.

العلاقات الفرعية ChildRelations:

تعيد نسخة من مجموعة العلاقات DataRelationCollection، تحتوي على العلاقات القادمة إلى هذا الجدول (العلاقات التي يدخل فيها كجدول فرعي أو جدول التفاصيل Details Table).

المفتاح الأساسي PrimaryKey:

تستقبل هذه الخاصية مصفوفة أعمدة DataColumn Array، تحتوي على الأعمدة التي تريد استخدامها كمفتاح أساسي للجدول.. ويمكنك استخدام مصفوفة بها خانة واحدة إذا كان المفتاح الأساسي يتكون من عمود واحد، أو استخدام مصفوفة بها أكثر من خانة إذا كنت تستخدم عمودين أو أكثر معا كمفتاح رئيسيّ للجدول.. مثلا: لو كان لديك جدول به عمود للاسم الأول للشخص، وعمود آخر لاسمه الأوسط، وعمود ثالث لاسمه الأخير، فكل عمود من هذه الأعمدة لا يصلح بمفرده كمفتاح أساسي بسبب تكرر الأسماء به، بينما قد تصلح الأعمدة الثلاثة معا كمفتاح أساسي، لأن الاسم الثلاثي نادرا ما يتكرر.. كل ما عليك في هذه الحالة هو وضع كائنات هذه الأعمدة في مصفوفة ووضعها في هذه الخاصية، لتصير هذه الأعمدة المفتاح الأساسي.

القيود Constraints:

تعيد مجموعة القيود ConstraintCollection الموجودة في هذا الجدول.. وسنتعرف على هذه المجموعة بالتفصيل لاحقا.

العرض الافتراضي DefaultView:

تعيد كائن العرض DataView Object الذي يحمل مبدئيا كل بيانات الجدول الحالي، لكنك تستطيع ضبطه لعرض جزء فقط من سجلات الجدول تبعا لشرط معين.. وسنتعرف على فئة عرض البيانات DataView Class بالتفصيل في الفصل التالي.

تعبير العرض DisplayExpression:

تقرأ أو تغير النصّ الذي سيتمّ عرضه للمستخدم كعنوان للجدول في أدوات عرض البيانات كالأداة DataGridView.

به أخطاء HasErrors:

تعيد true إذا كانت هناك أيّة أخطاء في أي صفّ في هذا الجدول.

الخصائص الإضافية ExtendedProperties:

تعيد مجموعة الخصائص PropertyCollection التي تحتوي على الخصائص الإضافيّة للجدول.. والمثال التالي يضيف خاصيّة اسمها Password إلى جدول الكتب، ويضع فيها القيمة "كلمة المرور"، ثم يغيرها إلى "أحمد١٢٣":

```
DataTable T = Ds.Tables["Books"];
PropertyCollection EP = T.ExtendedProperties;
EP.Add("Password", "كلمة المرور");
EP["Password"] = "أحمد١٢٣";
MessageBox.Show(EP["Password"].ToString( ));
```

كما يمتلك كائن الجدول الوسائل التالية:

نسخ Clone:

تعيد كائن جدول DataTable جديدا، وتنسخ إليه مخطط الجدول الحالي Schema بكلّ ما فيه من أعمدة وقيود.. ولكنّ الجدول الناتج يكون فارغا من السجلات.

نسخ Copy:

تعيد كائن جدول DataTable جديدا، وتنسخ إليه الجدول الحالي بمخططه وسجلاته، ليكون مماثلا للجدول الأصلي تماما.

محو Clear:

تمحو كلّ السجلات الموجودة في الجدول.

تصفير Reset:

تفرغ الجدول تماما من كل أعمدته وسجلاته وقيوده.

صف جديد NewRow:

تنشئ سجلا جديدا له نفس مخطط الجدول (نفس الأعمدة بنفس أنواع بياناتها بنفس ترتيبها)، وتعيد إليك كائن الصف DataRow الذي يشير إلى هذا السجل، لكن دون إضافته إلى مجموعة صفوف الجدول Rows، لهذا عليك أن تضيفه إليها بنفسك..

مثال:

```
DataTable T = Ds.Tables["Authors"];
var R = T.NewRow( );
R["Author"] = "أحمد شوقي";
R["About"] = "أمير الشعراء";
R["CountryID"] = 21;
T.Rows.Add(R);
```

استعارة صف ImportRow:

أرسل إلى هذه الوسيلة كائن صف DaraRow، لتنسخه وتضيفه إلى الجدول بكلّ بياناته وخصائصه، بما في ذلك النسخة الأصلية Original Version والنسخة الحالية Current Version لقيم خاناته.

تحميل صف LoadDataRow:

استخدم هذه الوسيلة لتحديث أحد سجلات الجدول، أو إضافة سجل جديد إليه.. وتستقبل هذه الوسيلة معاملين:

- مصفوفة كائنات Object Array، تحتوي على قيم خانات السجل بنفس ترتيبها في الجدول.. لاحظ أن هذه الوسيلة ستبحث في الجدول، لترى إن كان المفتاح الأساسي لأي حقل له نفس القيمة الموجودة في الخانة المناظرة في المصفوفة.. فإذا كان المفتاح الأساسي موجودا في الجدول، يتم نسخ باقي القيم من المصفوفة إلى باقي حقول السجلّ لتحديثها.. وإذا لم يكن المفتاح الأساسي موجودا، يتم إنشاء سجلّ جديد وتوضع بحقوله قيم المصفوفة.

  لاحظ أن ترك إحدى خانات المصفوفة فارغة، سيؤدي إلى وضع القيمة الافتراضية في العمود المناظر لها إن كانت له قيمة افتراضية، أو سيتم توليد الترقيم التلقائي إذا كانت للخاصية AutoIncrement لهذا العمود القيمة true.. فإذا لم يكن هذا أو ذاك، وكانت الخانة لا تقبل أن تظل فارغة، فسيحدث خطأ في البرنامج.. ويحدث خطأ أيضا إذا كان عدد خانات المصفوفة أكبر من عدد أعمدة الجدول.

- معامل منطقي Boolean، لو جعلت قيمته true فسيتمّ استدعاء الوسيلة AcceptChanges بعد إضافة السجل إلى الجدول وبهذا يعتبر هذا السجل سجلا أصليا لم يحدث له أي تغيير.. أما إذا جعلت قيمة هذا المعامل false، فسيعتبر السجل الجديد سجلا مضافا Added ويعتبر السجل الذي تم تحديثه سجلا معدلا Modified.

وتعيد هذه الوسيلة كائن صفّ DataRow يحمل مرجعا إلى الصفّ الذي تمّ تحديثه أو إضافته.

ملحوظة:

تقوم فئة المجموعة المحددة النوع Typed DataSet Class، بتعريف عدة وسائل محددة النوع في كل جدول للتعامل مع صفوفه.. على سبيل المثال، لو كان في المجموعة محددة النوع فئة لجدول المؤلفين اسمها AuthorsDataTable، وتم تعريف فئة اسمها AuthorsRow تمثل نوع صفوف هذا الجدول، فإن هذا الجدول سيحتوي على الوسائل التالية:

صف مؤلفين جديد NewAuthorsRow:

تعيد كائنا من النوع AuthorsRow يمثل صفا جديدا من صفوف جدول المؤلفين، بحيث يمكنك إضافته إلى جدول المؤلفين.. مثال:

```
var R = Ds.Authors.NewAuthorsRow;
R.Author = "أحمد شوقي";
R.About = "أمير الشعراء";
R.CountryID = 21;
Ds.Authors.AddAuthorsRow(R);
```

إضافة صف المؤلفين AddAuthorsRow:

تستقبل هذه الوسيلة معاملا من النوع AuthorsRow يمثل صفا من صفوف جدول المؤلفين، لإضافته إلى جدول المؤلفين، كما رأينا في المثال السابق.

وتوجد صيغة أخرى لهذه الوسيلة تستقبل قيم صف المؤلفين لإضافته إلى الجدول في خطوة واحدة.. هكذا مثلا يمكن اختزال المثال السابق:

```
Ds.Authors.AddAuthorsRow("أحمد شوقي", 21, "",
                         "أمير الشعراء", null);
```

لاحظ أننا أرسلنا نصا فارغا إلى خانة رقم الهاتف، كما وضعنا القيمة null في خانة طابع الوقت لأن قاعدة البيانات تولدها تلقائيا.. أما المفتاح الرئيسي لهذا الصف (وهو الحقل ID) فلم تطالبنا به هذه الوسيلة أصلا لأنها تعرف أنه يولد تلقائيا.

حذف صف المؤلفين RemoveAuthorsRow:

تستقبل هذه الوسيلة معاملا من النوع AuthorsRow يمثل صفا من صفوف جدول المؤلفين، لحذفه من جدول المؤلفين.

تحديد Select:

تعيد مصفوفة صفوف DataRow Array، تحتوي على بعض أو كلّ صفوف الجدول.. ولهذه الوسيلة الصيغ التالية:

١- الصيغة الأولى بدون معاملات، وهي تعيد مصفوفة تحتوي على كل سجلات الجدول.

٢- الصيغة الثانية تستقبل معاملا نصّيا، يمثل الشرط الذي على أساسه سيتم اختيار السجلات من الجدول، ويمكنك صياغة هذا الشرط بنفس قواعد صياغة الفقرة WHERE في استعلامات SQL.. والجملة التالية تعيد كلّ الكتب التي تبدأ بحروف تسبق حرف الثاء في الترتيب الأبجديّ:

```
DataRow[ ] R = T.Select("Book < 'ث' ");
```

ويمكنك أن تستخدم في تكوين الشرط، الدوال والكلمات المستخدمة في تكوين شرط الخاصية DataRow.Expression التي سنتعرف عليها لاحقا.

٣- الصيغة الثالثة تزيد على الصيغة السابقة بمعامل نصي يحدد ترتيب الصفوف.. ويتكون هذا المعامل من شقين:

أ- اسم العمود الذي يتم الترتيب على أساسه (مثل Book)، أو أي تعبير يجمع بين أكثر من عمودين كناتج ضربهما (مثل Copies_No * Price).

ب-نوع الترتيب، وهو إحدى الكلمتين التاليتين:

- ASC: للترتيب التصاعديّ وهو الترتيب الافتراضيّ لهذا يمكن ألا تكتب هذه الكلمة.
- DESC: للترتيب التنازلي.

والمثال التالي يعرض أسماء الكتب التي تبدأ بحروف تسبق حرف الثاء في الترتيب الأبجديّ، مرتبة تنازليّا على حسب اسم الكتاب:

```
var R = T.Select ("Book < 'ث' ", "Book DESC");
```

٤- الصيغة الرابعة تزيد على الصيغة السابقة بمعامل ثالث من نوع المرقم DataViewRowState، مما يتيح لك تحديد حالة السجلات التي تريد تطبيق الشرط عليها.. هذا يمكنك من البحث في السجلات المضافة أو المعدّلة أو المحذوفة... إلخ.. وسنتعرّف على المرقم DataViewRowState بالتفصيل لاحقا.

حساب Compute:

تبحث في الجدول عن السجلات التي تحقق الشرط المرسل إلى المعامل الثاني، وتجري على هذه السجلات دالة التجميع Aggregate Function المرسلة إلى المعامل الأول.. وتعيد هذه الوسيلة كائنا Object يحمل ناتج عملية التجميع.. دعنا نأخذ مثالا: افترض أنك تريد حساب عدد الكتب التي تبدأ بحروف تسبق حرف النون في جدول الكتب.. يمكنك فعل هذا باستخدام الوسيلة Compute كالتالي:

```
C = TblBooks.Compute("Count(Book)", "Book < 'ن' ");
```

لاحظ أنك تستطيع إرسال نص فارغ إلى المعامل الثاني، وفي هذه الحالة سيتم تطبيق دالة الجميع على جميع سجلات الجدول.. والجملة التالية تعيد إليك مجموع نسخ الكتب في الجدول:

```
C = TblBooks.Compute("Sum(Copies_No)", "");
```

ويوجد عيب خطير في هذه الوسيلة، فهي لا تستطيع حساب دالة التجميع على أكثر من عمود مباشرة.. فإذا أردت مثلا أن تحسب مجموع أثمان كل نسخ الكتب الموجودة في الجدول، فإن الجملة التالية غير مقبولة:

```
C = TblBooks.Compute("Sum(Copies_No * Price)", "");
```

ولحلّ هذه المشكلة، عليك إنشاء عمود جديد في الجدول، واستخدام خاصيّة "الصيغة" Expression الخاصّة به لتكون قيم خاناته هي حاصل ضرب العمودين المطلوبين، ثم تجري على هذا العمود الحسابات التي تريدها.. وسنتعرف الأعمدة المحسوبة بالتفصيل عند التعرف على خصائص كائن العمود DataCoulmn.

وقد استخدمنا هذه الوسيلة في حدث ضغط زر تحميل البيانات في المشروع CustomDataSet لحساب مجموع درجات كل طالب وعرضه في عمود "المجموع" في جدول العرض.. السبب في هذا أن مجموع درجات كل طالب لا يتم حفظه في الملف، لان عمود "المجموع" مضاف إلى جدول العرض فقط وليس موجودا في مجموعة البيانات، لهذا علينا أن نحسب قيمته بأنفسنا.. هذا هو الكود الذي يفعل هذا:

```
SumCell.Value = DsStudents.Grades.Compute(
    "SUM(Grade)", "StudentID = " +
    StdId.ToString( ).Trim( ));
```

حيث StdId هو متغير يحمل رقم الطالب المراد حساب مجموع درجاته.

معرفة التغييرات GetChanges:

تعيد كائن جدول DataTable جديدا، يحتوي فقط على الصفوف التي تمّ تعديلها أو إضافتها أو حذفها من الجدول الحاليّ، منذ أن تمّ تحميله أو منذ آخر استدعاء للوسيلة AcceptChanges.

وهناك صيغة ثانية لهذه الوسيلة، تستقبل معاملا من نوع المرقم DataRowState الذي تعرفنا عليه من قبل، لتتمكن من اختيار السجلات التي وقع عليها نوع معين من التغيير دون غيره.

معرفة الأخطاء GetErrors:

تعيد مصفوفة صفوف DataRow Array، تحتوي على الصفوف التي حدثت بها أخطاء عند محاولة حفظ الجدول في قاعدة البيانات، ليمكنك تصحيح أخطائها.

قبول التغييرات AcceptChanges:

تقوم باستدعاء الوسيلة AcceptChanges الخاصة بكل صف في الجدول.

رفض التغييرات RejectChanges:

تقوم باستدعاء الوسيلة RejectChanges الخاصة بكل صف في الجدول.

بدء تحميل البيانات BeginLoadData:

يؤدي استدعاء هذه الوسيلة إلى:

- إيقاف إرسال التنبيهات Notifications إلى فئات ADO.NET التي تتعامل مع الجدول.. هذا معناه إيقاف انطلاق الأحداث Events الخاصة بهذه الفئات.
- إيقاف تحديث الفهارس Indexes.
- إيقاف التحقق من قواعد الصحّة Constraints.

وعليك استدعاء هذه الوسيلة قبل البدء في إضافة عدد كبير من السجلات إلى الجدول، لأن تكرار تنفيذ العمليات المذكورة سابقا بعد إضافة كل سجل إلى الجدول يستهلك وقتا ملموسا ويجعل البرنامج بطيئا، لهذا من الأذكى إيقافها مؤقتا، ثم إعادة تشغيلها بعد الانتهاء من ملء الجدول بالسجلات.

انتهاء تحميل البيانات EndLoadData:

هذه الوسيلة رديفة للوسيلة BeginLoadData، وعليك استدعاؤها بعد انتهاء تحميل السجلات في الجدول، لإعادة تشغيل العمليات الخاصة بالتنبيهات والأحداث والفهارس والقيود، وبهذا تضمن تنفيذها مرة واحدة فقط بعد عملية التحميل.

دمج Merge:

تضيف سجلات الجدول المرسل إليها كمعامل، إلى الجدول الحالي، وإن كانت السجلات المضافة موجودة سابقا، يتم تحديث السجلات الموجودة بقيم السجلات القادمة.

ولهذه الوسيلة نفس صيغ الوسيلة DataSet.Merge، مع اختلاف واحد، هو أن المعامل الأول في هذه الصيغ هو كائن جدول بيانات DataTable وليس DataSet.

إنشاء قارئ بيانات CreateDataReader:

تعيد قارئ بيانات الجدول DataTableReader، الذي يمكنك من خلاله المرور عبر كل سجلات الجدول الحالي.. وسنتعرف على الفئة DataTableReader لاحقا في هذا الفصل.

كتابة كود المخطط WriteXmlSchema:

تكتب كود XML الذي يعبر عن مخطط الجدول الحالي، في الملف المرسل كمعامل، سواء كان هذا المعامل في صورة نص يمثل اسم الملف، أو كائن يتعامل مع الملف من الأنواع Stream أو TextWriter أو XmlWriter.

وتوجد صيغة ثانية لهذه الوسيلة، تزيد عليها بمعامل منطقي، إذا جعلت قيمته true فسيتم حفظ مخطط الجداول الفرعية التابعة لهذا الجدول أيضا مع مخطط الجدول الحالي.

كتابة الكود WriteXml:

تكتب كود XML الذي يمثل سجلات الجدول، في الملف المرسل كمعامل، سواء كان هذا المعامل في صورة نص يمثل اسم الملف، أو كائن يتعامل مع الملف من الأنواع Stream أو TextWriter أو XmlWriter.

ولبعض صيغ هذه الوسيلة معامل منطقي، إذا جعلت قيمته true فسيتم حفظ سجلات الجداول الفرعية التابعة لهذا الجدول أيضا.

ولبعض صيغ هذه الوسيلة معامل من نوع المرقم XmlWriteMode الذي تعرفنا عليه سابقا، لتستطيع من خلاله اختيار طريقة حفظ البيانات.

قراءة كود المخطط ReadXmlSchema:

تقرأ مخطط جدول أو أكثر من الملف المرسل كمعامل، سواء كان هذا المعامل في صورة نص يمثل اسم الملف، أو كائن يتعامل مع الملف من الأنواع Stream أو

TextWriter أو XmlWriter.. وتضيف هذه الوسيلة هذه المخططات إلى الجدول الحالي والجداول الفرعية التابعة له.

قراءة الكود ReadXml:

تقرأ سجلات جدول أو أكثر، من الملف المرسل كمعامل، سواء كان هذا المعامل في صورة نص يمثل اسم الملف، أو كائن يتعامل مع الملف من الأنواع Stream أو TextWriter أو XmlWriter.. وتضيف هذه الوسيلة هذه السجلات إلى الجدول الحالي والجداول التابعة له.

كما يمتلك كائن الجدول الأحداث التالية:

العمود تغير ColumnChanged:

ينطلق بعد أن تتغيّر إحدى القيم في أحد أعمدة الجدول.. والمعامل الثاني e لهذا الحدث من النوع DataColumnChangeEventArgs، وهو يمتلك الخصائص التالية:

	Column	تعيد كائن العمود DataColumn الذي حدث به التغيير.
	Row	تعيد كائن الصفّ DataRow، الذي توجد به الخانة التي تغيرت.
	ProposedValue	تقرأ أو تغير القيمة المقترحة (التي تغيرت).. هذا يتيح لك ــ لو أردت ــ تعديل القيمة التي تغيرت.

العمود يتغير ColumnChanging:

مماثل للحدث السابق، ولكنه ينطلق عند محاولة إجراء التغيير في أحد أعمدة الجدول (أي قبل حدوث التغيير بالفعل).. ويمكنك استخدام الكود التالي لإلغاء تغيير قيمة الخانة:

```
e.ProposedValue = e.Row[e.Column];
```

لكن عليك استخدام هذه الجملة داخل شرط، فلو استخدمتها هكذا بمفردها فستمنع المستخدم من تغيير أي خانة في أي عمود في الجدول.. لهذا فالعملي أن تستخدمها لمنع بعض القيم الخاطئة، مثل ترك عمود اسم المؤلف فارغا لأن هذا غير مقبول في قاعدة البيانات.. وستجد هذا الكود في الفئة الجزئية لجدول المؤلفين في مجموعة البيانات محددة النوع في المشروع TableAdapter:

```
if (e.Column == AuthorColumn && e.ProposedValue == "")
   e.ProposedValue = e.Row[e.Column];
```

ويمكنك بنفس الطريقة، إضافة أي شروط أخرى للتأكد من صلاحية القيم التي يدخلها المستخدم في باقي الأعمدة.

الصف تغير RowChanged:

ينطلق بعد أن تتغيّر إحدى القيم في أحد سجلات الجدول.. والمعامل الثاني e لهذا الحدث من النوع DataRowChangeEventArgs، وهو يمتلك الخاصيتين التاليتين:

Row	تعيد كائن الصفّ DataRow، الذي توجد به الخانة التابعة للعمود الذي حدث به التغيير.	
Action	تخبرك بنوع التغيير الذي حدث للصفّ، وهي تعيد إحدى قيم المرقم DataRowAction التالية: - Add: تمت إضافة الصف. - Delete: تم حذف الصف. - Change: يتم تغيير إحدى قيم الصف.	

		- ChangeOriginal: تم تغيير النسخة الأصلية Original Version من السجل. - ChangeCurrentAndOriginal: تم تغيير النسخة الأصلية Original Version والنسخة الحالية Current Version من السجل. - Commit: تم نقل التغييرات التي حدثت على تعاملات الصف Transactions إلى قاعدة البيانات نهائيا. - Rollback: تم التراجع عن التغييرات التي حدثت على تعاملات الصف. - Nothing: لم يحدث أي تغيير على الصف.

الصف يتغير RowChanging:

مماثل للحدث السابق، ولكنّه ينطلق عند محاولة إجراء التغيير في أحد سجلات الجدول (قبل حدوث التغيير).

صف جديد للجدول TableNewRow:

ينطلق بعد استدعاء الوسيلة NewRow التي تعيد سجلا جديدا من سجلات الجدول.. والمعامل الثاني e لهذا الحدث من النوع DataTableNewRowEventArgs، وهو يمتلك الخاصية Row التي تعيد كائن الصف DataRow الذي يمثل الصف الجديد الذي تم إنشاؤه.. هذا يتيح لك وضع أية قيم افتراضية تريها في خانات الصف الجديد.

تم حذف الصف RowDeleted:

ينطلق بعد حذف أحد سجلات الجدول.. والمعامل الثاني e لهذا الحدث من النوع DataRowChangeEventArgs وقد تعرفنا عليه سابقا.

لاحظ أن هذا الحدث ينطلق والصف ما زال موجودا فعلا في مجموعة صفوف الجدول لكن حالته تكون DELETED، وهو ما سيسبب خطأ في البرنامج لو حاولت إضافة الصف مرة أخرى إلى الجدول!!

يتم حذف الصف RowDeleting:

مماثل للحدث السابق، ولكنه ينطلق عند محاولة حذف أحد سجلات الجدول (قبل إتمام الحذف).. والحقيقة أن هذا الحدث قليل الفائدة، لأنه لا يمتلك خاصية لإلغاء عملية الحذف قبل وقوعها، وكان المنتظر أن يمتلك هذا الحدث الخاصية e.Cancel لتجعله مفيدا!

وكل ما تستطيع فعله فيه، هو حفظ الصف الذي سيتم حذفه في متغير احتياطي، ليمكنك التراجع عن عملية الحذف بعد هذا لو أردت.

لاحظ أنك لا تستطيع حذف الصف بنفسك من داخل هذا الحدث، فلو حاولت استخدام الوسيلة DataTable.Rows.Remove(e.Row) لحذف الصف، فسيحدث خطأ في البرنامج!

ولو أردت منع عملية الحذف بأبسط طريقة، فعليك فعل هذا من أداة العرض.. مثلا: لو وضعت زرا على النموذج للقيام بعملية الحذف، فيمكنك أن تضيف إليه الكود الذي يسأل المستخدم إن كان يريد إتمام الحذف فعلا أم لا.. أما إن كنت تستخدم جدول عرض، فعليك استخدام الأحداث الخاصة به لفعل هذا، وإلغاء عملية الحذف إن قرر المستخدم هذا.

أما لو كنت مصرا على استخدام هذا الحدث العقيم، فسيتوجب عليك كتابة بعض الكود لفعل هذا.. وقد فعلنا هذا في المشروع التعريف الجزئي لفئة جدول المؤلفين في المشروع TableAdapter، حيث استخدمنا الحدث AuthorsRowDeleting المشتق من الحدث RowDeleting لسؤال المستخدم إن كان يريد حذف الصف، فإن قرر إلغاء العملية، قمنا بإنشاء نسخة احتياطية من جدول المؤلفين، واستخدام الوسيلة Merge لإضافة نسخة من سجلات الجدول الحالي إلى الجدول الاحتياطي:

```
TempTable = new AuthorsDataTable( );
TempTable.Merge(this);
```

ثم استخدام الحدث RowDeleted لدمج سجلات الجدول الاحتياطي بالجدول الأصلي مرة أخرى، وهذا سيعيد حالة السجلات كما كانت، بما في ذلك السجل المحذوف:

```
if (TempTable != null && TempTable.Count > 0) {
        this.Merge(TempTable);
        TempTable.Clear( );
}
```

لاحظ أن الشرط TempTable.Count > 0 هو المؤشر الذي يشعرنا بأن المستخدم رفض إتمام عملية الحذف.. ولو لم نستخدمه، فسيتم التراجع عن جميع عمليات الحذف بغض النظر عن رأي المستخدم!

والخاصية Count بالمناسبة، هي خاصية معرفة في فئة الجدول محدد النوع، وهي مجرد اختصار للكود التالي:

```
if (TempTable.Rows.Count > 0)
```

ورغم أنها تعمل بشكل صحيح، يظل بهذه الطريقة عيب خطير، وهو اضطرارنا إلى نسخ كل سجلات الجدول للمحافظة على حالة سجل واحد فقط، وهذه كارثة على الذاكرة وسرعة التنفيذ إذا كان عدد سجلات الجدول ضخما!.. وللأسف، الوسيلة Merge الخاصة بالجدول لا تقبل دمج سجل منفرد، ولا تقبل إلا كائن جدول كمعامل.. ولو أردت حل هذه المشكلة، فعليك استخدام مجموعة بيانات احتياطية، ثم استخدام الوسيلة Merge الخاصة بمجموعة البيانات، لأنها تقبل دمج مصفوفة من السجلات، ومن السهل وضع السجل المراد حذفه في مصفوفة وإرسالها إليها:

```
TempDs = new DsAuthorsBooks( );
TempDs.Merge(new[ ] {e.Row});
```

تحذير هام:

لا تعرف نسخة جديدة من المجموعة TempDs على مستوى فئة جدول المؤلفين..
هذا الكود غير صحيح:

DsAuthorsBooks TempDs = new DsAuthorsBooks ( );

السبب في هذا أنه سيؤدي إلى تعريف دائري يعطل البرنامج عن العمل إلى أن يدمر كل مساحة الرصة Stack المتاحة له في الذاكرة.. فعند تعريف نسخة جديدة من مجموعة البيانات DsAuthorsBooks، ستقوم بتعريف نسخة من جدول المؤلفين، التي ستقوم بتعريف نسخة احتياطية من المجموعة TempDs التي ستقوم بتعريف نسخة جديدة من جدول المؤلفين، التي ستقوم بتعريف نسخة احتياطية من المجموعة TempDs وهكذا إلى ما لا نهاية!

وهذا نفس ما سيحدث إن استخدمت جدول مؤلفين احتياطيا وعرفت نسخة جديدة منه على مستوى الفئة.

ولحل هذه المشكلة، عرف المتغير على مستوى الفئة بدون الكلمة New:

DsAuthorsBooks TempDs;

ثم ضع النسخة الجديدة في هذا المتغير في الحدث RowDeleting:

TempDs = new DsAuthorsBooks ( );

لكن.. لماذا لا نستخدم نرفض التغيير الذي حدث للصف المحذوف لنستعيده مباشرة بجملة كالتالية:

e.Row.RejectChanges( );

فكرة جيدة، لكن هذه الطريقة ستعمل فقط مع المؤلفين القادمين من قاعدة البيانات، أما إذا أضاف المستخدم مؤلفا، ثم قرر حذفه، ثم ضغط Cancel لعدم إتمام عملية الحذف، فستسبب الوسيلة RejectChanges خطأ، لأن السجل المضاف يفقد كل قيمه فعليا عند حذفه!!.. هذا رغم أن هذا الصف ما زال موجودا في الجدول، وتستطيع أن تحصل على رقمه بالكود التالي:

MessageBox.Show(this.Rows.IndexOf(e.Row).ToString( ));

لكن حتى لو لم يحدث هذا الخطأ، فسيؤدي إلغاء تعديلات هذا الصف المضاف إلى حذفه من الجدول، وهكذا لن تستعيد الصف في كل الأحوال!

أيضا، لا يمكنك رفض تغييرات الجدول كله:

this.RejectChanges( );

لأن هذا سيضيع كل التعديلات التي قام بها المستخدم ولم يحفظها في قاعدة البيانات، كما أنه سيعيد كل السجلات التي حذفها من قبل، وليس فقط آخر سجل محذوف!

لهذا تظل طريقة مجموعة البيانات الاحتياطية أفضل وأدق طريقة يمكن استخدامها.

واضح طبعا أن ميكروسوفت كانت ستحيل حياتنا إلى نعيم لو أضافت الخاصية e.Cancel في هذا الحدث كما هو مألوف!

تم محو الجدول TableCleared:

ينطلق مباشرة بعد نجاح الوسيلة Clear في محو كل سجلات الجدول، وقبل العودة لتنفيذ باقي الكود الذي استدعى الوسيلة Clear.. لاحظ أن هذا الحدث لن ينطلق إذا حدثت أية أخطاء أثناء حذف سجلات الجدول.

والمعامل الثاني e لهذا الحدث من النوع DataTableClearEventArgs، وهو يمتلك الخصائص التالية:

	Table	تعيد كائن الجدول DataTable الذي يتم محو سجلاته.
	TableName	تعيد اسم الجدول.
	TableNamespace	تعيد نطاق اسم الجدول.

يتم محو الجدول TableClearing:

مماثل للحدث السابق، ولكنه ينطلق عند محاولة محو سجلات الجدول (بعد استدعاء الوسيلة Clear لكن قبل تنفيذها).. لاحظ أن هذا الحدث ينطلق دائما، حتى لو كان الجدول فارغا من السجلات فعلا.

فئة مجموعة الصفوف DataRowCollection Class

هذه الفئة ترث المجموعة InternalDataCollectionBase، وهي تحتوي على عناصر من نوع فئة صف البيانات DataRow Class.
وبخلاف ما ترثه من الفئة الأم، تمتلك هذه المجموعة الوسائل الجديدة التالية:

إضافة Add:

تضيف صفا إلى مجموعة الصفوف، ولها صيغتان:

١- الصيغة الأولى تستقبل كائن الصف DataRow الذي تريد إضافته.

٢- الصيغة الثانية تستقبل مصفوفة كائنات Object Array تحتوي على القيم التي تريد وضعها في خانات السجل.. لاحظ أنك تتعامل مع مجموعة الصفوف من خلال كائن الجدول DataTable، لهذا فإن هذه المجموعة تعرف تركيب السجلات التي ستضيفها إليها، وعليك مراعاة ترتيب الأعمدة وأنواع بياناتها عند وضعها في المصفوفة حتى لا يحدث خطأ، وعليك كذلك ترك خانة المصفوفة المناظرة لخانة الترقيم التلقائي فارغة.

وتقوم هذه الصيغة بإنشاء كائن صف جديد ووضع القيم به وإضافته إلى مجموعة الصفوف، وتعيد إليك كائن صف DataRow يشير إلى الصف الذي تمت إضافته إلى المجموعة.

تحتوي على Contains:

تعيد true إذا كانت مجموعة الصفوف تحتوي على السجل الذي له المفتاح الأساسي Primary Key المرسل كمعامل.. ولهذا الوسيلة صيغتان:

١- الصيغة الأولى تستقبل كائنا Object يحتوي على قيمة المفتاح الأساسي للسجل الذي تريد البحث عنه.

٢- الصيغة الثانية تستقبل مصفوفة كائنات Objects، تحتوي على قيم المفتاح الأساسي، وذلك إذا كان المفتاح الأساسي للجدول يتكون من أكثر من عمود.

البحث عن Find:

مماثلة للوسيلة السابقة في صيغتيها، إلا أنها تعيد كائن الصف DataRow الذي يملك مفتاحا أساسيا مساويا للقيمة المرسلة كمعامل، وتعيد null إذا لم تعثر على الصف.

وتسبب هذه الوسيلة خطأ في البرنامج إذا لم يكن الجدول الذي تبحث فيه يحتوي على مفتاح أساسي.. يمكن أن يحدث هذا رغم ان الجدول الأصلي في قاعدة البيانات يحتوي على مفتاح أساسي، وذلك إذا استخدمت الوسيلة Fill لملء مجموعة البيانات، دون استخدام الوسيلة FillSchema أولا، فهي التي تنشيء المفتاح الأساسي في جداول مجموعة البيانات.

الإدراج في موضع InsertAt:

أرسل إلى هذه الوسيلة كائن الصف DataRow والموضع الذي تريد إدراجه فيه في مجموعة الصفوف.

فئة صفّ البيانات DataRow Class

تتعامل هذه الفئة مع أحد صفوف الجدول، وهي لا تمتلك حدث إنشاء عاما Public Constructor، لهذا لا تستطيع إنشاء نسخة جديدة منها مباشرة، وبدلا من هذا عليك استخدام الوسيلة DataTable.NewRow للحصول على كائن صف جديد.. الحكمة من هذا، هي أن الوسيلة NewRow تستخدم مخطط الجدول لإنشاء صف له نفس الأعمدة بنفس أنواع البيانات ونفس الترتيب.. والكود التالي يعرّف صفّا جديدا ويضيفه إلى جدول الكتب:

```
var TblBooks = Ds.Tables["Books"];
DataRow BooksRow = TblBooks.NewRow( );
TblBooks.Rows.Add(BooksRow);
```

وتمتلك هذه الفئة الخصائص التالية:

الجدول Table:

تعيد كائن الجدول DataTable الذي يحتوي على السجلّ الحالي.. تذكر أنك لا تستطيع إنشاء سجل جديد بدون استخدام الوسيلة NewRow من أحد الجداول، لهذا حتى لو لم يكن السجل مضافا إلى مجموعة صفوف الجدول Rows، فإن هذه الخاصية ستظل تشير دائما إلى الجدول الذي تم إنشاء السجل الحالي منه.

العنصر Item:

هذه هي الخاصية الافتراضية، وهي تقرأ أو تغير القيمة المحفوظة في إحدى خانات السجل الحالي.. لاحظ أن هذه الخاصية من النوع Object، ليمكنك التعامل مع مختلف أنواع الأعمدة.. ولهذه الخاصية الصيغ التالية:

١- بعض الصيغ لها معامل واحد، يستقبل اسم العمود أو رقمه، أو كائن العمود DataColumn الذي يمثله.. والمثال التالي يقرأ اسم المؤلف الموجود في الصفّ الثالث في جدول المؤلفين (الصفّ الأوّل هو الصف رقم صفر):

```
var R = Ds.Tables["Authors"].Rows[2];
var X = R["Author"];
```

وهو ما يمكنك فعله في سطر واحد كالتالي:

```
var X = Ds.Tables["Authors"].Rows[2]["Author"];
```

حيث يبدو أننا نتعامل مع مجموعة الصفوف Rows كأنها مصفوفة مصفوفات.

أما لو كنت تتعامل مع مجموعة بيانات محددة النوع، فسيختصر الكود السابق إلى:

```
var X = Ds.Authors[2]["Author"];
```

أو بصورة أفضل:

```
var X = Ds.Authors[2].Author;
```

٢- بعض الصيغ تزيد بمعامل ثان من نوع المرقم DataRowVersion، لتحدد من خلاله نسخة السجل التي تريد التعامل معها.. لكن هذه الصيغ للقراءة فقط، ولا يمكن استخدامها لتغيير قيم السجل.. ويمتلك المرقم DataRowVersion القيم التالية:

Original	التعامل مع نسخة السجل الأصلية Original Version.
Current	التعامل مع النسخة الحالية للسجل Current Version.
Proposed	التعامل مع القيمة المقترحة للسجل.. هذا مفيد إذا كان السجل قيد التحرير ولم يتم إنهاء عملية التحرير بعد، وتريد قراءة القيمة الجديدة قبل قبولها ووضعها في النسخة الحالية Current Version.
Default	التعامل مع النسخة الافتراضية للصف، وهي كالتالي: - النسخة الافتراضية للصفوف غير المعدلة هي النسخة الأصلية Original Version. - النسخة الافتراضية للصفوف المعدلة والمضافة والمحذوفة هي النسخة الحالية Current Version. - النسخة الافتراضية للصفوف غير المتصلة بأي جدول Detached، هي النسخة المقترحة Proposed.

والقيمة المستخدمة مع الصيغ التي لا تمتلك المعامل الثاني هي Default.

انظر المثال التالي:

```
// عرض النسخة الأصلية
MessageBox.Show(Row[0,
    DataRowVersion.Original].ToString ( ));
// عرض النسخة الحالية
MessageBox.Show(Row["Book",
    DataRowVersion.Current].ToString ( ));
// عرض النسخة الافتراضية
MessageBox.Show(Row["Book"].ToString( ));
```

مصفوفة العنصر ItemArray:

تعيد مصفوفة كائنات Object Array تحتوي على قيم كلّ خانات السجلّ الحالي بنفس ترتيب الأعمدة.. ويمكنك أن ترسل إليها مصفوفة بها القيم التي تريد وضعها في خانات السجل.. مثال:

```
var TblAuthors = Ds.Tables["Authors"];
var R = TblAuthors.NewRow( );
R.ItemArray = new object [] {null, "عنترة بن شداد", 5, "",
        "شاعر جاهلي", null};
TblAuthors.Rows.Add(R);
```

به أخطاء HasErrors:

تعيد true إذا كانت هناك أخطاء متعلقة بالسجل الحالي.. ويمكنك معرفة سبب الخطأ باستخدام الخاصية RowError أو الوسيلة GetColumnsInError، مع ملاحظة أن عليك فحص كلتيهما، لأنهما لا تحتويان على نفس البيانات!

خطأ الصف RowError:

تقرأ أو تغيّر النصّ الذي يصف الخطأ الذي حدث في هذا الصفّ.. لاحظ أن وضع أي نص في هذه الخاصية يغير قيمة الخاصية HasErrors إلى true، ويجعل جدول العرض يضع أيقونة الخطأ بجوار هذا الصف.

حالة الصف RowState:

تعيد إحدى قيم المرقم DataRowState التي توضح حالة السجلّ من حيث كونه مضافا أو محذوفا أو معدّلا.. وقد تعرفنا على قيم هذا المرقم سابقا.

كما تمتلك هذه الفئة الوسائل التالية:

له نسخة HasVersion:

تعيد true، إذا كان السجل الحالي يمتلك نسخة البيانات الموضحة في المعامل المرسل إلي هذه الوسيلة، وهو من نوع المرقم DataRowVersion الذي تعرفنا عليه من قبل.

والمثال التالي يعرض قيمة الخانة الأولى في نسخة السجل المقترحة إن وجدت:

```
if (Row.HasVersion(DataRowVersion.Proposed))
   MessageBox.Show(Row[0,
            DataRowVersion.Proposed].ToString( ));
```

تغيير الحالة إلى مضاف SetAdded:

تغير قيمة الخاصية RowState إلى Addded.. وتسبب هذه الوسيلة خطأ في البرنامج إذا كان كانت حالة السجل تشير إلى أنه معدل.. ولحل هذه المشكلة، عليك استدعاء الوسيلة AcceptChanges أولا.

تغيير الحالة إلى معدل SetModified:

تغير قيمة الخاصية RowState إلى Modified.. وتسبب هذه الوسيلة خطأ في البرنامج إذا كان كانت حالة السجل تشير إلى أنه معدل.. ولحل هذه المشكلة، عليك استدعاء الوسيلة AcceptChanges أولا.

قبول التغييرات AcceptChanges:

يؤدّي استدعاء هذه الوسيلة إلى قبول التغييرات التي حدثت على الصف الحالي منذ أن تمّ تحميله من قاعدة البيانات، أو منذ آخر مرّة تمّ فيها استدعاء الوسيلة AcceptChanges.. ليس معنى هذا أنّ هذه التغييرات سيتمّ حفظها في قاعدة البيانات، ولكن سيتمّ النظر إليها على أنها البيانات الأصليّة للسجل، ولن يمكنك التراجع عنها.. لاحظ أن هذه الوسيلة تفعل ما يلي:

- تضع القيمة Unchanged في الخاصية RowState للسجل إذا كانت قيمتها تشير إلى أنه معدل أو مضاف.
- تزيل السجل من الجدول نهائيا إذا كانت حالته تشير إلى أنه محذوف Deleted.
- تنقل القيم من النسخة الحالية Current Version إلى النسخة الأصلية Original Version للسجل.

وعليك أن تستخدم هذه الوسيلة بحذر، حتى لا تضيع التغييرات التي حدث للسجل الحالي دون حفظها في الجدول الأصلي في قاعدة البيانات.

وتسبب هذه الوسيلة خطأ في البرنامج، إذا حاولت استدعائها لقبول تغييرات صف ليس مضافا إلى أي جدول!

رفض التغييرات RejectChanges:

يؤدّي استدعاء هذه الوسيلة إلى رفض كلّ التغييرات التي أجريت على السجل الحالي، بحيث يعود إلى الحالة التي كان عليها عند تحميله من قاعدة البيانات أو عند آخر استخدام للوسيلة AcceptChanges.. لاحظ أن هذه الوسيلة تفعل التالي:

- تضع القيمة Unchanged في الخاصية RowState للسجل إذا كانت حالته تشير إلى أنه معدل أو محذوف.
- تزيل السجل من الجدول نهائيا إذا كانت حالته تشير إلى أنه مضاف Added.
- تنقل القيم من النسخة الأصلية Original Version إلى النسخة الحالية Current Version للسجل.

وتتيح لك هذه الوسيلة استعادة القيم الأصلية للسجل.. هذا مفيد في بعض الحالات، مثل التخلص من القيم التي سببت خطأ في السجل.

بدأ التحرير BeginEdit:

تبدأ عمليّة تحرير الصف الحالي، وتعطل أحداث الجدول التي تنطلق عند حدوث تغيرات في السجلات، كما توقف عمليّات التحقق من صحّة البيانات المدخلة في كل خانة من خانات السجل، مما يتيح للمستخدم تحرير كل خانات السجل بدون أي اعتراض.

ويتم استدعاء هذه الوسيلة تلقائيا عندما يحاول المستخدم تحرير بيانات السجل المعروض في أدوات ربط البيانات Data-bound Controls مثل مربعات النصوص وجدول عرض البيانات DataGridView.

لاحظ أن السجل يحتفظ بالبيانات التي يتم إدخالها أثناء عملية التحرير في النسخة المقترحة Proposed Version.

إلغاء التحرير CancelEdit:

تلغي عمليّة التحرير التي بدأت باستدعاء الوسيلة Begin Edit، وتتخلص من نسخة السجل المقترحة Proposed Version، وتحتفظ بالنسخة الحالية Current Version كما هي.. هذا معناه إلغاء التغييرات التي حدثت على السجل أثناء عملية التحرير.

إنهاء التحرير EndEdit:

تنهي عمليّة التحرير التي بدأت باستدعاء الوسيلة Begin Edit، وتفحص القيم التي تم إدخالها في السجل أثناء وضع التحرير، فإن كانت صحيحة تقوم بحفظ نسخة السجل المقترحة Proposed Version في النسخة الحالية Current Version.. هذا معناه حفظ التغييرات التي حدثت على السجل أثناء عملية التحرير.

ويتم استدعاء هذه الوسيلة تلقائيا عند استدعاء الوسيلة AcceptChanges.

معرفة خطأ العمود GetColumnError:

تعيد نصّا يصف الخطأ الذي حدث في إحدى خانات السجلّ الحالي.. وتستقبل هذه الوسيلة معاملا يوضح العمود الذي توجد فيه هذه الخانة، سواء في صورة رقم العمود أو اسمه أو كائن العمود DataColumn الذي يمثله.

تغيير خطأ العمود SetColumnError:

تسمح لك بوضع نصّ يصف الخطأ الذي حدث في إحدى خانات السجلّ الحالي.. ولهذه الوسيلة معاملان:

- المعامل الأول يوضح العمود الذي توجد فيه هذه الخانة، سواء في صورة رقم العمود أو اسمه أو كائن العمود DataColumn الذي يمثله.
- المعامل الثاني يستقبل النص الذي يشرح سبب الخطأ.

لاحظ أن هذه الوسيلة أكثر تفصيلا من الخاصية RowError، لأنها تحدد الخطأ الذي حدث في كل خانة على حدة.. وتؤدي الوسيلة SetColumnError إلى وضع القيمة true في الخاصية HasErrors، وإلى ظهور أيقونة الخطأ في جدول العرض في الخانة الناتجة من تقاطع الصف الحالي مع العمود الذي أرسلته كمعامل.. لكن هذه الوسيلة لا تؤثر على قيمة الخاصية RowError.

معرفة الأعمدة التي بها أخطاء GetColumnsInError:

تعيد مصفوفة أعمدة DataColumn Array تحتوي على الأعمدة التي بها أخطاء في السجلّ الحالي.

محو الأخطاء ClearErrors:

تمحو كلّ النصوص التي تشير إلى حدوث أخطاء في السجلّ.. هذا سيجعل الخاصية RowError والوسيلة GetColumnError تعيدان نصوصا فارغة.

حذف Delete:

تضع القيمة Deleted في الخاصية RowState الخاصة بالسجل الحالي.. هذا يتيح لك التراجع عن حذف هذا السجل باستدعاء الوسيلة RejectChanges أو حذفه فعلا عند استدعاء الوسيلة AcceptChanges.

لاحظ أن استخدام الوسيلة Delete مع سجل مضاف (RowState = Added) سيؤدي إلى حذف هذا السجل في الحال.

معرفة الصفوف التابعة GetChildRows:

تعيد مصفوفة صفوف DataRow Array بها كلّ السجلات المرتبطة بعلاقة بهذا السجلّ في جداول أخرى.. ولهذه الوسيلة الصيغ التالية:

١. الصيغة الأولى تستقبل كائن العلاقة DataRelation الذي تريد استخدامه.. هذا ضروري، لأن السجل الحالي قد يكون له سجلات فرعية في أكثر من جدول، كما هو الحال في جدول الدول Countries، الذي له سجلات فرعية في جدولي المؤلفين والناشرين.

٢. الصيغة الثانية تستقبل اسم العلاقة، لتبحث عنها في مجموعة العلاقات الفرعية ChildRelations الخاصة بالجدول الذي يوجد به السجل الحالي.

٣. الصيغتان الثالثة والرابعة مماثلتان للصيغتين السابقتين، ولكنهما تزيدان بمعامل ثان من نوع المرقم DataRowVersion، ليمكنك من خلاله اختيار نسخة السجلات Version التي تريد قراءتها من الجدول الفرعي.

تغيير الصف الرئيسي SetParentRow:

أرسل إلى هذه الوسيلة كائن السجلّ DataRow الذي تريد جعله السجل الرئيسي Master للسجل الحالي.. لاحظ أن السجل الرئيسي يمكن أن يكون في جدول آخر (الجدول الرئيسي)، أو أن يكون في الجدول الحالي (علاقة ذاتية Self Relation).

وتوجد صيغة أخرى لهذه الوسيلة، تزيد على الصيغة السابقة بمعامل ثانٍ، يستقبل كائن العلاقة DataRelation الذي يربط بين السجلين.

ويمكنك استخدام هذه الوسيلة إذا أردت تصحيح خطا في الجدول الفرعي، كأن تغير مؤلف أحد الكتب بعد نسبته خطأ إلى مؤلف آخر.. وفي هذه الحالة كل ما ستفعله هذه الوسيلة، هي وضع قيمة المفتاح الرئيسي ID للمؤلف، في خانة المفتاح الفرعي AuthorID للكتاب.

معرفة الصف الرئيسي GetParentRow:

تعيد السجلّ الرئيسي الذي يرتبط به السجلّ الحالي بعلاقة.. ولهذه الوسيلة عدة صيغ:

١. بعض الصيغ تستقبل معاملا واحدا، هو العلاقة التي يشترك فيها السجل الحالي، سواء في صورة اسم العلاقة، أو كائن العلاقة DataRelation.

٢. بعض الصيغ تستقبل معاملا ثانيا من نوع المرقم DataRowVersion، ليمكنك من خلاله تحديد النسخة Version التي تريد قراءتها من السجل الرئيسي.

معرفة الصفوف الرئيسية GetParentRows:

مماثلة للوسيلة السابقة في صيغها، لكنها تعيد مصفوفة صفوف DataRow Array، تحتوي على كلّ السجلات الرئيسيّة التي تشير إلى السجلّ الحالي.. في الحقيقة لا تبدو لهذه الوسيلة أية أهمية حاليا، فهي دائنا تعيد سجلا رئيسيا واحدا، وهذا يجعل استخدام الوسيلة GetParentRow أكثر منطقية!

هل هي عدم IsNull:

تعيد true إذا كانت الخانة الموجودة في السجلّ الحالي والعمود المرسل كمعامل فارغة DbNull.. ولهذه الوسيلة الصيغ التالية:

١- بعض الصيغ لها معامل واحد، يستقبل اسم العمود أو رقمه أو كائن العمود DataCoulmn الذي يمثله.

٢- وهناك صيغ لها معامل ثان من نوع المرقم DataRowVersion، ليمكنك من خلاله تحديد النسخة Version التي تريد فحص قيمها.

فئة مجموعة الأعمدة DataColumnCollection Class

هذه الفئة ترث المجموعة InternalDataCollectionBase، وهي تحتوي على عناصر من نوع فئة عمود البيانات DataColumn Class.
ولا تملك هذه المجموعة أية خصائص أو وسائل جديدة غير ما ترثه من الفئة الأم، ولكن بعض هذه العناصر يحتاج منا إلى وقفة:

العنصر Item:

تعيد كائن عمود البيانات DataColumn الموجود في المجموعة بناء على المعامل المرسل إليها.. ولهذه الوسيلة صيغتان:

١- الصيغة الأولى تستقبل رقم الخانة التي يوجد بها العمود في المجموعة.. لاحظ أن خطأ سيحدث لو أرسلت رقم خانة غير موجودة في المجموعة.

٢- والصيغة الثانية تستقبل نصا يمثل اسم العمود في الجدول، ليتم البحث عنه في المجموعة، فإن كان موجودا تعيد هذه الوسيلة كائن العمود الذي يمثله، وإن لم يكن موجودا فإنها تعيد null ولا يحدث خطأ.

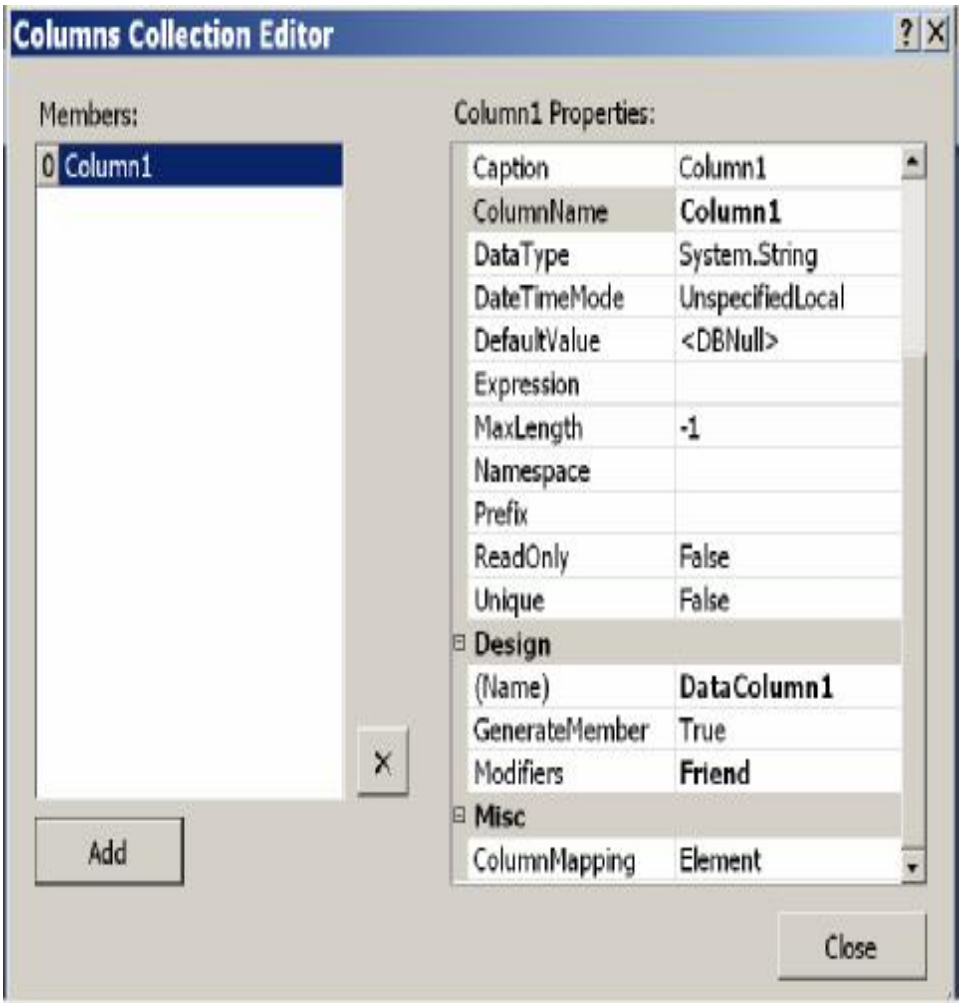
إضافة Add:

تضيف عمودا إلى مجموعة الأعمدة، مع ملاحظة أن الأعمدة التي تضيفها إلى مجموعة البيانات هي أعمدة مؤقتة خاصة بالبرنامج فقط، ولا تظهر في قاعدة البيانات، حتى بعد إجراء عملية التحديث Update.. لكن لو كنت تحتاج إلى إنشائها في قاعدة البيانات، فعليك باستخدام أوامر SQL الخاصة بإنشاء أعمدة تناظر الأعمدة الجديدة التي أضفتها في مجموعة البيانات.
وتمتلك هذه الوسيلة الصيغ التالية:

١- الصيغة الأولى بدون معاملات، وهي تنشئ عمودا جديدا بالاسم الافتراضي (Column1 أو Column2 ... إلخ).. لاحظ أن هذا العمود سيتعامل مع بيانات نصية String.
٢- الصيغة الثانية تستقبل كائن العمود DataColumn وتضيفه إلى المجموعة.
٣- الصيغة الثالثة تستقبل اسم العمود، وتقوم بإنشائه وإضافته إلى المجموعة.. وتستطيع إرسال نص فارغ إلى هذه الوسيلة، لإنشاء عمود له الاسم الافتراضي (Column1 أو Column2 ... إلخ).
٤- الصيغة الرابعة تزيد على الصيغة السابقة بمعامل ثان من نوع فئة النوع Type، ليمكنك من خلاله تحديد نوع بيانات العمود.. لاحظ أن نوع العمود يعتبر نصيا String في الصيغ التي لا تستقبل هذا المعامل.
٥- الصيغة الخامسة تزيد على الصيغة السابقة بمعامل ثالث، يستقبل نصا يمثل الصيغة التي ستوضع في الخاصية Expression الخاصة بالعمود، مما يتيح لك إنشاء عمود محسوب Calculated Column.. وسنتعرف على هذه الخاصية عند التعرف على فئة عمود البيانات DataColumn.

لاحظ أن هذه الصيغ تعيد كائن العمود DataColumn الذي أضيف إلى مجموعة الأعمدة، ما عدا الصيغة الثانية فأنت ترسل إليها كائن العمود بالفعل لهذا ليست لها قيمة عائدة.

ويمكنك إضافة الأعمدة إلى هذه المجموعة بطريقة مرئية في وقت التصميم، وذلك من خلال نافذة خصائص الجدول.. لفعل هذا يجب أن يكون لديك كائن جدول في صينية مكونات النموذج (وهذا غير شائع)، أو يمكنك استخدام مجموعة بيانات عادية Un-Typed DataSet موضوعة في صينية المكونات، فلو عرضت خصائصها في نافذة الخصائص، فسيمكنك استخدام الخاصية Tables لعرض محرر مجموعة الجداول، ولو حددت أي جدول في هذه المجموعة، فستظهر خصائصه في القسم الأيمن من النافذة، وستجد بينها الخاصية Columns.. ولو ضغطت زر الانتقال الموجود في خانة هذه الخاصية، فستظهر نافذة محرر مجموعة الأعمدة، كما في الصورة:



في هذه النافذة يمكنك ضغط الزر Add لإضافة عمود جديد، حيث ستظهر خصـــائص هذا العمود في القسم الأيمن، ويمكنك تغييرها كما تشاء.

موضع العمود IndexOf:

تبحث عن العمود المرسل إليها كمعامل في مجموعة الأعمدة، وتعيد رقم الخانة التـــي يوجد بها في المجموعة، أو تعيد -١ إن لم يكن موجودا.. ولهذه الوسيلة صيغتان:

١- الصيغة الأولى تستقبل اسم العمود.

٢- والصيغة الثانية تستقبل كائن العمود DataColumn.

يمكن حذفه CanRemove:

تعيد true إذا كان من الممكن حذف كائن العمود DataColumn المرسل كمعامل من مجموعة الأعمدة.. وتعيد هذه الوسيلة false إذا لم يكن العمود موجودا في المجموعة، أو إذا كان داخلا في علاقة.

حذف Remove:

تحذف العمود المرسل إليها كمعامل من مجموعة الأعمدة.. ولهذه الوسيلة صيغتان:

١- الصيغة الأولى تستقبل اسم العمود.

٢- والصيغة الثانية تستقبل كائن العمود DataColumn.

كما تمتلك مجموعة الأعمدة الحدث التالي:

المجموعة تغيرت CollectionChanged:

ينطلق عندما يتغير عدد الأعمدة، سواء بالحذف أو الإضافة.

والمعامل الثاني e لهذا الحدث من النوع CollectionChangeEventArgs الذي تعرفنا عليه من قبل في مجموعة الجداول DataTableCollection.

فئة عمود البيانات DataColumn Class

تمثل هذه الفئة مخطط أحد أعمدة الجدول، وهي ترث الفئة MarshalByValueComponent، مما يتيح لك إضافتها إلى صينية مكونات النموذج، وإن كان عليك أن تضيفها أولا إلى صندوق الأدوات.

ولحدث إنشاء هذه الفئة نفس صيغ الوسيلة Add الخاصة بمجموعة الأعمدة DataColumnCollection، ما عدا الصيغة الثانية.. كما توجد صيغة إضافية لحدث الإنشاء، تستقبل المعاملات التالية بالترتيب:

- اسم العمود.
- كائن نوع Type Object، يحتوي على نوع بيانات العمود.
- صيغة العمود التي ستوضع في الخاصية Expression التي سنتعرف عليها بعد قليل.
- إحدى قيم المرقم MappingType، لتوضع في الخاصية ColumnMapping الخاصة بالعمود، وسنتعرف عليها بعد قليل.

والمثال التالي يعرف عمودا نصيا اسمه Temp ويضيفه إلى جدول الكتب في مجموعة البيانات التي تحمل الاسم Ds:

```
var Clmn = new DataColumn("Temp", typeof(string));
Ds.Tables["Books"].Columns.Add(Clmn);
```

وتمتلك فئة العمود الخصائص التالية:

الجدول Table:

تعيد كائن الجدول DataTable الذي ينتمي إليه هذا العمود.

اسم العمود ColumnName:

تقرأ أو تغير اسم العمود.

نطاق الاسم Namespace:

تقرأ أو تغير نطاق الاسم الخاص بالجدول الذي ينتمي إليه العمود.

البادئة Prefix:

تقرأ أو تغير البادئة التي تمثل نطاق اسم الجدول الذي ينتمي إليه العمود.

الرتبة Ordinal:

تعيد عددا صحيحا يمثل ترتيب العمود في مجموعة الأعمدة.

العنوان Caption:

تقرأ أو تغير عنوان العمود.. من المفروض أن يتم عرض هذا العنوان بدلا من اسم العمود في أدوات ربط البيانات Data-Bound Controls، لكنك لو جربت هذا مع جدول العرض مثلا، فلن تجد له تأثيرا!.. يبدو أن عليك ربط أداة العرض حصريا بهذه الخاصية، لكي ترى تأثيرها، كما سنرى لاحقا.

نوع البيانات DataType:

تقرأ أو تغير كائن النوع Type Object الذي يمثل نوع بيانات العمود.

القيمة الافتراضية DefaultValue:

تقرأ أو تغير القيمة الافتراضيّة لخانات العمود.. يمكنك مثلا أن تضع الرقم صفر في هذه الخاصية إذا كنت تتعامل مع عمودي رقمي.

السماح بالعدم AllowDBNull:

إذا جعلت قيمة هذه الخاصيّة true، فسيُسمح بترك بعض خانات هذا العمود فارغة DbNull.

أقصى طول MaxLength:

تقرأ أو تغير أقصى عدد من الحروف يمكنك كتابته في العمود الذي يتعامل مع بيانات نصية.. والقيمة الافتراضية لهذه الخاصية -١ مما يعني عدم وجود قيود على عدد الحروف.. لاحظ أن قيمة هذه الخاصية سيتم تجاهلها إذا كان العمود يتعامل مع بيانات من نوع آخر غير النصوص.

للقراءة فقط ReadOnly:

إذا جعلت قيمة هذه الخاصيّة true، فلن يمكنك تغيير قيمة أي خانة في هذا العمود بعد إضافة الصف الذي توجد به إلى الجدول.. لاحظ أن جعل قيمة هذه الخاصية true مع عمود يحمل ناتج عملية حسابية موضوعة في الخاصية Expression، سيؤدي إلى حدوث خطأ في البرنامج، لأن تغيير قيمة أي خانة في عمود داخل في العملية الحسابية سيؤدي تلقائيا إلى إعادة حساب قيمة الخانة المناظرة في عمود الناتج.

متفرد Unique:

إذا جعلت قيمة هذه الخاصيّة true، فلن يسمح بتكرار قيم خانات هذا العمود.

ترقيم تلقائي AutoIncrement:

إذا جعلت قيمة هذه الخاصيّة true، فستزيد قيمة خانات هذا العمود تلقائيّا كلما تمت إضافة صف جديد إلى الجدول.. لاحظ أن عمود الترقيم التلقائي يكون للقراءة فقط ReadOnly، ولا يمكنك تغيير قيمته بنفسك.. والقيمة الافتراضية لهذه الخاصية هي false.

بذرة الترقيم التلقائي AutoIncrementSeed:

تحدد رقم بدء الترقيم في عمود الترقيم التلقائيّ.. والقيمة الافتراضية هي ١.

خطوة الترقيم التلقائي AutoIncrementStep:

تحدّد مقدار الزيادة في عمود الترقيم التلقائيّ.. والقيمة الافتراضية هي ١.

الخصائص الإضافية ExtendedProperties:

تعيد مجموعة الخصائص PropertyCollection التي تحتوي على الخصائص الإضافيّة للعمود، وهي مماثلة لتلك الخاصّة بكائن الجدول.

خريطة العمود ColumnMapping:

تحدد كيف يتم تمثيل العمود في كود XML عند حفظ الجدول، وهي تأخذ إحدى قيم المرقم MappingType التالية:

Element	يتم تمثيل العمود كعنصر XML.. هكذا مثلا سيتم حفظ العمودين ID و Author في جدول المؤلفين: <Authors> <ID>1</ID> <Author> = 'توفيق الحكيم </Authors>
Attribute	يتم تمثيل العمود كسمة XML.. هكذا مثلا سيتم حفظ العمود ID: <Authors ID=1 Author = 'توفيق الحكيم'> <Book ID=1 Book = 'شهرزاد'/> </Authors>
SimpleContent	يتم تمثيل العمود كفرع من نوع الفئة XmlText.. هذا الموضوع خارج نطاق هذا الكتاب.
Hidden	هذا العمود خفي، ويستخدم في البناء الداخلي للجدول، لكن لا يظهر للمستخدم عند ربط الجدول بأدوات ربط البيانات.

نظام التاريخ والوقت DateTimeMode:

تحدد نظام التوقيت المستخدم مع هذا الجدول، إذا كان نوع بياناته تاريخ أو وقت.. وتأخذ هذه الخاصية إحدى قيم المرقم DataSetDateTime التالية:

Local	التعامل بالتاريخ المحلي للجهاز الذي يوجد عليه البرنامج.
Utc	التعامل بالتاريخ العالمي.
Unspecified	غير محدد.
UnspecifiedLocal	توقيت محلي غير محدد.. هذه هي القيمة الافتراضية.

لاحظ أنك لا تستطيع تغيير قيمة هذه الخاصية بعد إضافة الخانات إلى العمود، إلا في حالة واحدة: إذا كنت تحول من القيمة Unspecified إلى UnspecifiedLocal أو العكس.

التعبير Expression:

تقرأ أو تغير الصيغة النصية للعمود، والتي يمكن استخدامها فيما يلي:

- حساب قيم خانات العمود الحالي، من ناتج عملية حسابية على أعمدة أخرى، وفي هذه الحالة يسمى بالعمود المحسوب Calculated Column.
- حساب ناتج شرط معين على كل سجل، لاستخدام الوسيلة DataTable.Select بعد ذلك للحصول على السجلات التي حققت هذا الشرط، وسنرى مثالا على هذا بعد قليل.

والجدول التالي يلخص لك المعاملات والدوالّ التي يمكنك استخدامها لتكوين صيغة العمود:

المعاملات الحسابية	يمكنك استخدام المعاملات الحسابية التالية على الأعمدة: الجمع: +، الطرح: -، الضرب *، القسمة /، باقي القسمة %.. مثال: **Col.Expression = "Price * Copies_No";**
المعاملات المنطقية	يمنكن استخدام المعاملات المنطقية التالية بين الأعمدة: AND, OR, NOT والمثال التالي يرى إن كان اسم الكتاب يبدأ بحرف يسبق الميم

	أبجديا، وأن الكتاب للمؤلف رقم ٤: **Col.Expression =** **"Book < 'م' And AuthorID = 4";** لاحظ أن التعبير السابق سيضع في خانات العمود true أو false، مما يسمح لك باستخدام الوسيلة Select الخاصة بكائن الجدول لاختيار الصفوف التي تحقق أو لا تحقق شرطا معينا.. اعتبر أن العمود في المثال السابق اسمه Col.. هذا المثال يحصل على الصفوف التي تحقق الشرط السابق: **DataRow[] R = T.Select ("Col = true");**
معاملات المقارنة	يمكنك استخدام معاملات المقارنة التالية: = <> > < >= <= IN LIKE بنفس الطريقة التي تعرّفنا عليها عند شرح جمل SQL، مع ملاحظ أن علامات التعويض (* أو %) غير مسموح بها في منتصف التعبير المستخدم مع الدالة LIKE.. مثلا، التعبير التالي غير مقبول: **" 'أحمد*توفيق' Like "Author** لكن يمكن استخدام علامات التعويض في بداية النص أو نهايته.. مثال: **" 'هيكل*' Like "Author** **" '*هيكل' Like "Author**
CONVERT	تقوم بالتحويل بين أنواع البيانات.. مثال: **Col.Expression =** **"Convert(ID, 'System.Int32')";**
LEN	تعيد طول النصّ الموجود في خانات العمود.. والمثال التالي سيجعل العمود Col يعرض طول أسماء الكتب الموجودة في العمود Book: **Col.Expression = "Len(Book)";**

لاحظ أنّك غير مجبر على قصر معامل هـذه الدالـة (والـدوال التالية أيضا) على اسم أحد حقول الجدول.. إنّ تعبيرا كهذا متاح أيضا: **Col.Expression="Len(Book + Author)";**	
تستقبل هذه الدالة معاملين: اسم العمـود، وقيمـة افتراضـية.. وتفحص هذه الدالة العمود، فإن وجدت فيه قيمـة أعادتهـا، وإن وجدته فارغا أعادت القيمة الافتراضية.. المثال التالي يعيد قيمـة العمود AuthorID أو -١ إذا كان فارغا: **Col.Expression = "IsNull(AuthorID, -1)";**	ISNULL
تزيل المسافات من بداية ونهاية خانات الحقـل المرسـل إليهـا كمعامل.. مثال: **Col.Expression= "TRIM(AuthorID)";**	TRIM
تعيد جزءا من النصّ الموجود في خانات العمود المرسل إليهـا كمعامل، بدءا من الحرف المذكور رقمه في المعامـل الثـاني، وبـالطول المحـدّد فـي المعامـل الثالـث.. (مماثلـة للدالـة String.SubString).. مثال: **Col.Expression =** **"SUBSTRING(Book, 2, 8)";**	SUBSTRING
تأخذ هذه الدالة ثلاثة معاملات: - الأوّل شرط سيتمّ التحقق منه. - والثاني هو القيمة المعادة إذا كان الشرط صحيحا. - والثالث هو القيمة المعادة إذا كان الشرط خاطئا. مثال: **Col.Expression =** **"IIF(LEN(Price)>10), 'غال', 'رخيص')";**	IIF

Parent	تشير إلى الجدول الرئيسي الذي يدخل في علاقة مع الجدول الحالي.. افترض أن المتغير Col يتعامل مع عمود في جدول الكتب.. المثال التالي يضع في هذا العمود طول اسم كل مؤلف: **Col.Expression = "LEN(Parent.Author)";** وإذا كان الجدول مشتركا في أكثر من علاقة مع جداول رئيسية اخرى، فيمكنك أن ترسل اسم العلاقة بين قوسين بعد الكلمة Parent كالتالي: **Col.Expression =** **"LEN(Parent(AuthorsBooks).Author)";**
Child	تشير إلى الجدول الثانويّ المرتبط بالجدول الحاليّ بعلاقة، حيث تستخدم المفتاح الأساسي Primary Key لكل سجل في الجدول الرئيسي، للحصول على السجلات الفرعية المرتبطة بهذا المفتاح في الجدول الفرعي.. ونظرا لأن ناتج هذه الدالة قد يكون أكثر من سجل، فلا يمكن استخدامها بمفردها، وإنما تستخدم مع إحدى دوال التجميع. والمثال التالي يضيف عمودا إلى جدول المؤلفين، يعرض عدد الكتب التي ألفها كلّ مؤلف: **DataColumn Col = new** **DataColumn("NoOfBooks", typeof(int));** **Col.Expression =** **"Count(Child.AuthorID)";** **Ds.Tables["Authors"].Columns.Add(Col);** ولو كان الجدول داخلا في أكثر من علاقة، فيمكنك إرسال اسم العلاقة كمعامل إلى التعبير Child، مثل: **Col.Expression =** **"Count(Child(AuthorsBooks).AuthorID)";**
دوال التجميع	يمكنك استخدام دوال التجميع التالية: Sum, Avg, Min, Max, Count, StDev, Var لاحظ أن استخدام هذه الدوال على العمود الحالي سيجعلها تتعامل

	مع كل الصفوف بدون تقسيمها إلى مجموعات Grouping.. مثلا، لو استخدمت التعبير: "Count (AuthorID)" في جدول الكتب فسيعيد عدد صفوف جدول الكتب.. ولو أردت تجميع الصفوف التي تتشابه في قيمة عمود معين، فعليك استخدام التعبير Child للاستفادة من العلاقة بين جدولين في تجميع الصفوف التي تتشابه في المفتاح الرئيسي.. ولو استخدمت التعبير: "Count (Child.AuthorID)" في جدول المؤلفين، فستحصل على مجموع كتب كل مؤلف.
\r	حرف نهاية السطر.
\n	حرف بداية السطر.
\t	علامة جدولة Tab.

لاحظ أن عليك وضع النصوص والتواريخ بين العلامتين ' ' مثل:

'طويل'
'1/1/2009'

كما يمكن وضع التواريخ بين العلامتين # #، مثل:

#1/1/2009#

ويمتلك كائن العمود الوسيلة التالية:

تغيير الرتبة SetOrdinal:

أرسل إلى هذه الوسيلة عددا صحيحا، يمثل الموضع الجديد الذي تريد أن يصير العمود فيه في مجموعة الأعمدة.. هذا هو الحل الوحيد لتغيير موضع العمود، لأن مجموعة الأعمدة لا تحتوي على الوسيلة Insert.. وقد استخدمنا هذه الوسيلة في الزر "حفظ البيانات ٢" في المشروع CustomDataSet لجعل العمود Subject في الموضع رقم ٢ (العمود الثالث) بعد إعادة إضافته إلى الجدول.. لاحظ أن هذا سيغير

ترتيب العمود في الجدول، لكنه سيظهر في جدول العرض كـــآخر عمـــود!.. هـــذا لا يؤثر في عمل البرنامج، لكنه يزعج المستخدم، لهذا عليك تغيير ترتيب عرض العمود في جدول العرض أيضا.. هذا هو سبب استخدامنا للجملة التالية في نهايـــة الإجـــراء ShowGrades:

GradesCols[2].DisplayIndex = 0;

هذه الجملة تجعلنا واثقين أن العمود الذي يعرض اسم المادة يظهر دائما قبـــل العمـــود الذي يعرض درجات الطالب.

فئة قارئ جدول البيانات DataTableReader Class

هذه الفئة ترث فئة قارئ البيانات الأم DbDataReader Class، وهي تشبه قارئ البيانات العادي في طريقة عملها، لكنها لا تستخدم كائن أمر للحصول على السجلات من قاعدة البيانات، فهي تقرأ السجلات من جداول مجموعة البيانات مباشرة.

ولإنشاء قارئ بيانات يقرأ سجلات أحد الجداول، عليك باستخدام الوسيلة CreateDataReader الخاصة بهذا الجدول كالتالي:

```
var Tr = Ds.Tables["Authors"].CreateDataReader( );
while (Tr.Read( ))
{
        MessageBox.Show(Tr["ID"].ToString( ));
        MessageBox.Show(Tr["Author"].ToString( ));
}
```

ولإنشاء قارئ بيانات يقرأ سجلات كل الجداول، عليك باستخدام الوسيلة CreateDataReader الخاصة بمجموعة البيانات DataSet كالتالي:

```
DataTableReader Tr = Ds.CreateDataReader( );
do
{
  while (Tr.Read( ))
  {
        string RowTxt = "";
        for (var I = 0; I < Tr.FieldCount; I++)
             RowTxt += Tr.GetName(I) + " = " +
                   Tr[I].ToString( ) + "\r\n";
         MessageBox.Show(RowTxt);
  }
} while (Tr.NextResult( ));
```

لاحظ أننا لم نستخدم أسماء الأعمدة عند قراءة خانات كل سجل، وذلك لأن السجلات ستختلف من جدول إلى آخر في عدد الأعمدة وأسمائها.. وبدلا من هذا استخدمنا الخاصية FieldCount لإنشاء حلقة تكرار تمر عبر كل الأعمدة، لقراءة كل خانة باستخدام رقم العمود بدلا من اسمه.

ويمكنك تجربة هذا الكود في المشروع DataTableReaderSample.

ولحدث إنشاء الفئة DataTableReader الصيغتان التاليتين:

١- الأولى تستقبل كائن الجدول DataTable الذي ستقرأ سجلاته.

٢- والثانية تستقبل مصفوفة جداول DataTable Array لتقرأ سجلاتها.

ولا تمتلك هذه الفئة أية خصائص أو وسائل جديدة غير ما ترثه من الفئة الأم.

فئة مجموعة العلاقات
DataRelationCollection Class

هذه الفئة ترث الفئة InternalDataCollectionBase، وهي تحتوي على كائنات من نوع فئة علاقة البيانات DataRelation Class.
وتمتلك هذه الفئة الخصائص والوسائل الشهيرة للمجموعات Collections، ومعظمها يستخدم اسم العلاقة كمعامل، أو يستخدم كائن العلاقة DataRelation الذي يمثلها.. لهذا نحتاج هنا إلى التركيز على العناصر التالية فقط:

العنصر Item:

هذه هي الخاصية الافتراضية، وهي تستقبل اسم العلاقة كمعامل أو رقم العلاقة في المجموعة، وتعيد إليك كائن العلاقة DataRelation الذي يمثلها.. وقد استخدمنا هذه الخاصية في المشروع DataSetContents، للحصول على كائن العلاقة التي يضغط المستخدم اسمها في قائمة العلاقات، لنعرض خصائصها في مربع رسالة.

إضافة Add:

تضيف علاقة إلى المجموعة، ولها الصيغ التالية:

١- الصيغة الأولى تستقبل كائن العلاقة DataRelation المراد إضافته.

٢- الصيغة الثانية تستقبل معاملين من النوع DataColumn، يمثلان الحقل الرئيسي والحقل الفرعي على الترتيب، حيث سيتم إنشاء علاقة بينهما، وإضافتها إلى المجموعة.

٣- الصيغة الثالثة مماثلة للصيغة السابقة، إلا أنها تستقبل مصفوفتين من النوع DataColumn، وذلك لمراعاة الحالة التي يتكون فيها كل من المفتاح الأساسي والفرعي من أكثر من عمود.

٤- الصيغة الرابعة تستقبل ثلاثة معاملات: اسم العلاقة، وكائن العمود الرئيسي، وكائن العمود الفرعي.
٥- الصيغة الخامسة تزيد بمعامل منطقي على الصيغة السابقة، إذا جعلت قيمته false فلن يتم إنشاء قيود عند إنشاء العلاقة.. والقيمة الافتراضية في الصيغ التي لا تحتوي هذا المعامل هي true، لهذا يتم إنشاء قيد التفرد UniqueConstraint على الحقل الأساسي إن لم يكن موجودا، وإنشاء قيد المفتاح الفرعي ForeignKeyConstraint على الحقل الفرعي إن لم يكن موجودا، ويتم إضافتهما إلى قيود الجدول.
٦- الصيغة السادسة مماثلة للصيغة السابقة، إلا أن معامليها الثاني والثالث يستقبلان مصفوفة حقول DataColumn Array لمراعاة الحالة التي يتكون فيها كل من المفتاحين الأساسي والفرعي من أكثر من حقل.

لاحظ أن جميع الصيغ ــ ما عدا الأولى ــ تعيد كائن العلاقة DataRelation الذي تم إنشاؤه وإضافته إلى المجموعة.

المجموعة تغيرت CollectionChanged:

ينطلق هذا الحدث عندما يتغير عدد عناصر مجموعة العلاقات، سواء بالحذف أو الإضافة.. والمعامل الثاني e لهذا الحدث من النوع CollectionChangeEventArgs الذي تعرفنا عليه من قبل في مجموعة الجداول DataTableCollection.

فئة العلاقة DataRelation Class

تحتوي هذه الفئة على تفاصيل العلاقة المنشأة بين جدولين.

ولحدث إنشاء هذه الفئة نفس صيغ الوسيلة DataRelationCollection.Add ما عدا الصيغة الأولى التي تستقبل كائن علاقة.. وإضافة إلى هذه الصيغ، يمتلك حدث الإنشاء الصيغتين الجديدتين التاليتين:

١- الصيغة الأولى تستقبل المعاملات التالية:

- نصا يمثل اسم العلاقة.
- نصا يمثل اسم الجدول الرئيسي.
- نصا يمثل اسم الجدول الفرعي.
- مصفوفة نصية تحتوي على أسماء أعمدة المفتاح الأساسي.
- مصفوفة نصية تحتوي على أسماء أعمدة المفتاح الفرعي.
- معاملا منطقيا Boolean، لتوضع قيمته في الخاصية Nested الخاصة بكائن العلاقة.

٢- الصيغة الثانية تستقبل المعاملات التالية:

- نصا يمثل اسم العلاقة.
- نصا يمثل اسم الجدول الرئيسي.
- نصا يمثل نطاق اسم Namespace الجدول الرئيسي.
- نصا يمثل اسم الجدول الفرعي.
- نصا يمثل نطاق اسم الجدول الفرعي.
- مصفوفة نصية تحتوي على أسماء أعمدة المفتاح الأساسي.
- مصفوفة نصية تحتوي على أسماء أعمدة المفتاح الفرعي.
- معاملا منطقيا Boolean، لتوضع قيمته في الخاصية Nested الخاصة بكائن العلاقة.

والمثال التالي يريك كيف تنشئ علاقة بين الحقل ID في جدول المؤلّفين، والحقل AuthorID في جدول الكتب:

```
DataColumn ID = Ds.Tables["Authors"].Columns["ID"];
var AuthorID = Ds.Tables["Books"].Columns["AuthorID"];
var R = new DataRelation("AuthorsBooks", ID, AuthorID);
```

لاحظ أنّ هذه العلاقة لم توضع في مجموعة البيانات DataSet إلى الآن، لهذا عليك إضافتها إلى مجموعة العلاقات بنفسك كالتالي:

```
Ds.Relations.Add(R);
```

وبعد تنفيذ الجملة الأخيرة، ستضاف هذه العلاقة تلقائيا إلى مجموعة العلاقات الرئيسية ParentRelations لجدول المؤلفين، ومجموعة العلاقات الفرعية ChildRelations لجدول الكتب.

وتمتلك فئة العلاقة الخصائص التالية:

مجموعة البيانات DataSet:

تعيد كائن مجموعة البيانات DataSet، الذي تنتمي إليه هذه العلاقة.

اسم العلاقة RelationName:

تقرأ أو تغيّر اسم العَلاقة.

الجدول الرئيسي ParentTable:

تعيد كائن جدول DataTable يمثل الجدول الرئيسيّ في هذه العلاقة.

الجدول الفرعي ChildTable:

تعيد كائن جدول DataTable يمثل الجدول الثانويّ في هذه العلاقة.

الأعمدة الرئيسية ParentColumns:

تعيد مصفوفة من النوع DataColumn تحتوي على الأعمدة الرئيسيّة في هذه العلاقة.

الأعمدة الفرعية ChildColumns:

تعيد مصفوفة من النوع DataColumn تحتوي على الأعمدة الثانويّة في هذه العلاقة.

قيد المفتاح الرئيسي ParentKeyConstraint:

تعيد كائن قيد التفرد UniqueConstraint، المفروض على المفتاح الرئيسيّ في هذه العلاقة.

قيد المفتاح الفرعي ChildKeyConstraint:

تعيد كائن قيد المفتاح الفرعي ForeignKeyConstraint، المفروض على المفتاح الثانويّ في هذه العلاقة.

متداخلة Nested:

تفيد عند حفظ سجلات الجدول الرئيسي في ملف XML باستخدام الوسيلة WriteXml، فلو جعلت قيمة هذه الخاصية true، فسيتم حفظ السجلات الفرعية مع السجل الأصلي الذي تربطها به العلاقة الحالية.

الخصائص الإضافية ExtendedProperties:

تعيد مجموعة الخصائص PropertyCollection التي تحتوي على الخصائص الإضافيّة للعمود، وهي مماثلة لتلك الخاصّة بكائن الجدول.

فئة مجموعة القيود ConstraintCollection Class

هذه الفئة ترث المجموعة InternalDataCollectionBase، وكل عنصر من عناصرها من نوع فئة القيد Constraint Class التي سنتعرف عليها لاحقا.
وتمتلك هذه الفئة الخصائص والوسائل الشهيرة للمجموعات Collections، ومعظمها يستخدم اسم القيد كمعامل، أو يستخدم كائن القيد Constraint الذي يمثله.. لهذا نحتاج هنا إلى التركيز على العناصر التالية فقط:

العنصر Item:

أرسل إلى هذه الخاصية رقم القيد في المجموعة، أو نصا يحمل اسم القيد، لتعيد إليك كائن القيد Constraint الذي يمثله.

إضافة Add:

تضيف قيدا إلى المجموعة، ولها الصيغ التالية:

١- الصيغة الأولى تستقبل كائن القيد Constraint المراد إضافته.

٢- الصيغة الثانية تنشئ قيد تفرد UniqueConstraint وتضيفه إلى المجموعة.. وهي تستقبل ثلاثة معاملات:

- اسم القيد.
- كائن العمود DataColumn الذي يجب أن يكون متفردا.
- قيمة منطقية إذا جعلتها true فسيتم جعل العمود المرسل إلى المعامل الثاني مفتاحا أساسيا للجدول.

٣- الصيغة الثالثة مماثلة للصيغة السابقة، إلا أن معاملها الثاني يستقبل مصفوفة أعمدة DataColumn Array، وذلك إذا كان المفتاح المطلوب تفرده في الجدول يتكون من أكثر من عمود.

٤- الصيغة الرابعة تنشئ قيد مفتاح فرعي ForeignKeyConstraint وتضيفه إلى المجموعة.. وهي تستقبل المعاملات التالية:

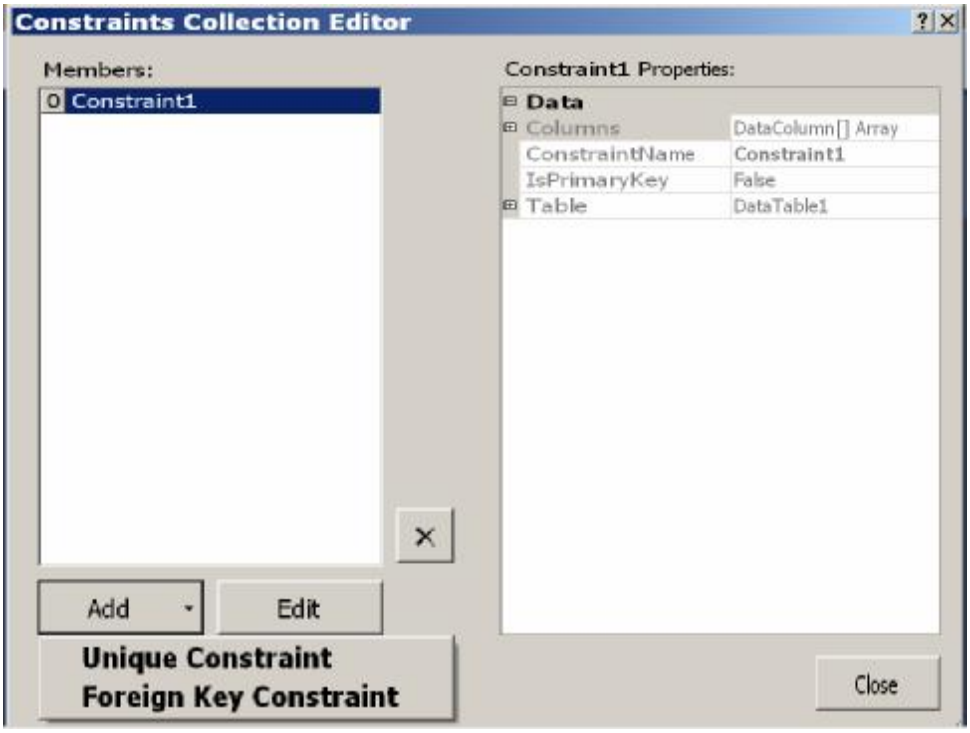
- اسم القيد.

- كائن العمود DataColumn الأساسي.
- كائن العمود DataColumn الفرعي.

٥- الصيغة الخامسة مماثلة للصيغة السابقة، إلا أن معامليها الثاني والثالث يستقبلان مصفوفة أعمدة DataColumn Array، وذلك إذا كان المفتاح الأساسي والمفتاح الفرعي يتكونان من أكثر من عمود.

لاحظ أن جميع الصيغ ما عدا الأولى، تعيد كائن القيد Constraint الذي تم إنشاؤه وإضافته إلى المجموعة.

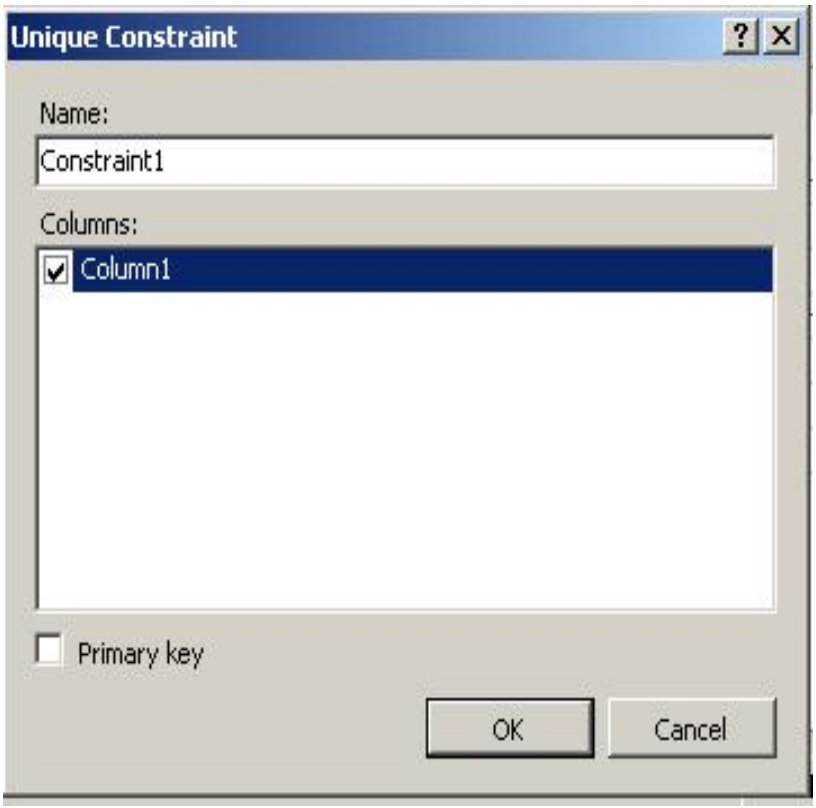
ويمكنك إضافة القيود إلى هذه المجموعة بطريقة مرئية في وقت التصميم، وذلك من خلال نافذة خصائص الجدول.. لفعل هذا يجب أن يكون لديك كائن جدول في صينية مكونات النموذج (وهذا غير شائع)، أو يمكنك استخدام مجموعة بيانات عادية Un-Typed DataSet موضوعة في صينية المكونات، فعند عرض خصائصها في نافذة الخصائص، سيمكنك استخدام الخاصية Tables لعرض محرر مجموعة الجداول، ولو حددت أي جدول في هذه المجموعة، فستظهر خصائصه في القسم الأيمن من النافذة، وستجد بينها الخاصية Constraints.. ولو ضغطت زر الانتقال الموجود في خانة هذه الخاصية، فستظهر نافذة محرر مجموعة القيود، كما في الصورة:



اضغط الزر Add لإضافة قيد جديد.. ستظهر لك قائمة موضعية لتتيح لـــك اختيـــار نوع القيد، من بين النوعين التاليين:

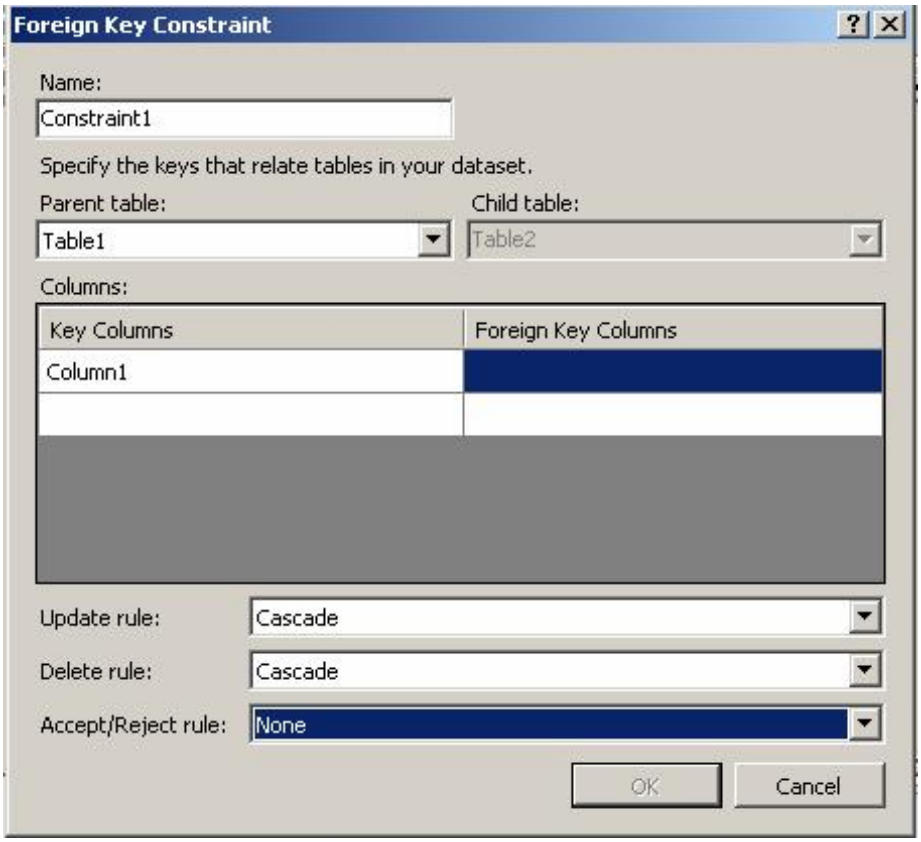
١- قيد التفرد Unique Constraint:

عند ضغط هذا الاختيار، ستظهر لك نافذة إنشاء قيد التفرد، وقد تعرفنا عليها فـــي مخطط مجموعة البيانات، وهي تبدو كما في الصورة:



٢- قيد المفتاح الفرعي Foreign Key Constraint:

عند ضغط هذا الاختيار، ستظهر لك نافذة إنشاء قيد المفتاح الفرعي، وهي تشـــبه نافذة إنشاء علاقة، ولا جديد فيها، وتبدو كما في الصورة:



وبعد أن تضيف القيد سيظهر في القائمة اليسرى، وستظهر خصائصه في القائمة اليمنى.

يمكن حذفه CanRemove:

تعيد true إذا كان من الممكن أن تحذف من المجموعة، كائن القيد Constraint المرسل إليها كمعامل، بدون حدوث خطأ في البرنامج.. مثلا: محاولة حذف قيد التفرد UniqueConstraint قبل حذف قيد المفتاح الفرعي ForeignKeyConstraint المرتبط به، تؤدي إلى حدوث خطأ في البرنامج، لهذا عليك استخدام هذه الوسيلة قبل استخدام الوسيلة Remove أو RemoveAt.

المجموعة تغيرت CollectionChanged:

ينطلق هذا الحدث عندما يتغير عدد عناصر مجموعة القيود، سواء بالحذف أو الإضافة.. والمعامل الثاني e لهذا الحدث من النوع CollectionChangeEventArgs الذي تعرفنا عليه من قبل في مجموعة الجداول DataTableCollection.

فئة القيد Constraint Class

هذه الفئة أساسة مجردة Abstract Base Class وتجب وراثتها، وهي تعمل كفئة أم لكل من فئة قيد التفرد UniqueConstraint Class وقيد المفتاح الفرعي ForeignKeyConstraint Class.

وتمتلك هذه الفئة الخصائص التالية:

اسم القيد ConstraintName:

تقرأ أو تغير اسم القيد.

الجدول Table:

تعيد كائن الجدول DataTable الذي ينطبق عليه القيد.

الخصائص الإضافية ExtendedProperties:

تعيد مجموعة الخصائص PropertyCollection التي تحتوي على الخصائص الإضافيّة للقيد، وهي مماثلة لتلك الخاصّة بكائن الجدول.

فئة قيد التفرّد UniqueConstraint Class

هذه الفئة ترث الفئة Constraint، وهي تحتوي على تفاصيل قيد التفرد الذي يضمن عدم تكرار قيم حقل أو مجموعة من الحقول.

ولحدث إنشاء هذه الفئة الصيغ التالية:

١- الأولى تستقبل كائن العمود DataColumn الذي سيفرض عليه القيد.

٢- الصيغة الثانية تزيد على الصيغة السابقة بمعامل منطقي، إذا جعلت قيمته true يتم جعل العمود المرسل إلى المعامل الأول مفتاحا أساسيا للجدول.

٣- الصيغة الثالثة تستقبل مصفوفة أعمدة DataColumn Array تحتوي على الأعمدة التي سيفرض عليها القيد.

٤- الصيغة الرابعة تزيد على الصيغة السابقة بمعامل منطقي، إذا جعلته true فسيتم جعل الأعمدة المرسلة إلى المعامل الأول مفتاحا أساسيا للجدول.

٥- هناك صيغ مماثلة للصيغ السابقة، لكنها تزيد بمعامل أول يستقبل اسم القيد.

٦- الصيغة الأخيرة تستقبل ثلاثة معاملات:

- اسم القيد.
- مصفوفة نصية تستقبل أسماء الأعمدة.
- معامل منطقي إذا جعلت قيمته true يتم جعل الأعمدة المرسلة إلى المعامل الثاني مفتاحا أساسيا للجدول.

والكود التالي يعرف قيد التفرد على الحقل ID لجدول المؤلفين:

```
var ID = Ds.Tables("Authors").Columns("ID");
var Uc = new UniqueConstraint("IDUnique", ID, true);
```

إلى الآن لم يوضع هذا القيد في جدول المؤلفين، لذا عليك أن تضيفه كالتالي:

```
Ds.Tables("Authors").Constraints.Add(Uc);
```

لكن هذا القيد لن يعمل، إلا إذا جعلت للخاصيّة "فرض القيود" EnforceConstraints الخاصّة بمجموعة البيانات القيمة true:

```
Ds.EnforceConstraints = true;
```

وإضافة إلى ما ترثه من الفئة الأم، تمتلك فئة قيد التفرد الخاصيتين التاليتين:

الأعمدة Columns:

تعيد مصفوفة تحتوي على الأعمدة التي يؤثّر عليها هذا القيد.

هل هو مفتاح أساسي IsPrimaryKey:

تعيد true إذا كان هذا القيد مفروضا على المفتاح الأساسيّ للجدول.

فئة قيد المفتاح الثانوي ForeignKeyConstraint Class

هذه الفئة ترث الفئة Constraint، وهي تحتوي على تفاصيل قيد المفتاح الثانوي، المفروض على السجلات في الجدول الثانويّ، والذي يضمن صحة العلاقة بين الجدولين، ويمنع حذف أحد السجلات الأساسية في الجدول الرئيسيّ إذا كانت له سجلات فرعية في الجدول الثانوي، كما يمنع تعديل قيمة المفتاح الأساسي في أي سجل إذا كانت هذه القيمة مستخدمة في المفتاح الفرعي لسجلات الجدول الثانوي.

ولحدث إنشاء هذه الفئة الصيغ التالية:

١- الصيغة الأولى تستقبل كائن العمود DataColumn الذي يمثل المفتاح الرئيسي في العلاقة، وكائن العمود DataColumn الذي يمثل المفتاح الفرعي في العلاقة، لفرض القيد عليهما.

٢- الصيغة الثانية مماثلة للصيغة السابقة، إلا أنها تتعامل مع مصفوفتين من النوع DataColumn، لمراعاة الحالة التي يتكون فيها المفتاح الأساسي والمفتاح الفرعي من أكثر من عمود.

٣- الصيغة الثالثة تزيد على كل من الصيغتين السابقتين بمعامل أول، يستقبل اسم القيد.

٤- الصيغة الرابعة تستقبل المعاملات التالية بالترتيب:

- نصا يمثل اسم القيد.
- نصا يمثل اسم الجدول الرئيسي.
- نصا يمثل نطاق اسم الجدول.
- مصفوفة نصية تحتوي على أسماء الأعمدة التي تعمل كمفتاح أساسي.
- مصفوفة نصية تحتوي على أسماء الأعمدة التي تعمل كمفتاح فرعي.
- إحدى قيم المرقم AcceptRejectRule لوضعها في الخاصية AcceptRejectRule التي سنتعرف عليها بعد قليل.
- إحدى قيم المرقم Rule لوضعها في الخاصية DeleteRule التي سنتعرف عليها بعد قليل.

- إحدى قيم المرقم Rule لوضعها في الخاصية UpdateRule التي سنتعرف عليها بعد قليل.

٥- الصيغة الخامسة مماثلة للصيغة السابقة، لكن ينقصها المعامل الثالث الذي يستقبل نطاق اسم الجدول.

والمثال التالي يعرّف قيدا من هذا النوع:

```
var ID = Ds.Tables("Authors").Columns("ID");
var AuthorID = Ds.Tables(Books").Columns("AuthorID");
var Fkc = new ForeignKeyConstraint("AuthorIDCnst",
    ID, AuthorID);
```

ويجب أن تضيف هذا القيد إلى الجدول الثانوي، مع ملاحظة أن خطأ سيحدث لو حاولت إضافته إلى الجدول الرئيسي.. والكود التالي يضيف القيد الذي أنشأناه إلى مجموعة قيود جدول الكتب:

```
Ds.Tables("Books").Constraints.Add(Fkc);
```

لاحظ أن تنفيذ هذه الجملة سيؤدي إلى إضافة قيد التفرد تلقائيا على العمود ID في جدول المؤلفين، وذلك إذا لم يكن هذا القيد موجودا مسبقا.

وإضافة إلى ما ترثه من الفئة الأم، تمتلك هذه الفئة الخصائص التالية:

الجدول المرتبط RelatedTable:

تعيد كائن جدول DataTable يمثل الجدول الرئيسيّ في العلاقة.

الأعمدة المرتبطة RelatedColumns:

تعيد مصفوفة أعمدة DataColumn Array، تحتوي على الأعمدة التي تعمل كمفتاح أساسي في هذه العلاقة.

الأعمدة Columns:

تعيد مصفوفة أعمدة Array DataColumn، تحتوي على الأعمدة التي تعمل كمفتاح ثانوي في هذه العلاقة.

قاعدة القبول والرفض AcceptRejectRule:

توضّح الفعل الذي سيتمّ اتخاذه عند استدعاء الوسيلة DataRow.AcceptChanges أو الوسيلة DataRow.RejectChanges.. وتأخذ هذه الخاصيّة إحدى قيمتي المرقم AcceptRejectRule التاليتين:

None	عند قيام أحد السجلات بقبول التغييرات أو رفضها، لا يحدث أي شيء للسجلات المرتبطة به في العلاقة.. هذه هي القيمة الافتراضية.
Cascade	عند قيام أحد السجلات بقبول التغييرات أو رفضها، يتم قبول التغييرات أو رفضها في السجلات المرتبطة به في العلاقة.. مثلا: لو تم قبول تغييرات سجل في جدول المؤلفين، يتم قبول التغييرات على التوالي لكل السجلات التي تحتوي على كتب هذا المؤلف في جدول الكتب.

قاعدة الحذف DeleteRule:

توضّح الفعل الذي سيتمّ اتخاذه مع السجلات الفرعية، عند حذف السجل الرئيسي الذي تنتمي إليه، وهي تأخذ إحدى قيم المرقم Rule التالية:

None	تترك السجلات الفرعية كما هي بدون تغيير، لكن هذا قد يؤدي إلى حدوث خطأ في البرنامج، لأن حذف أو تعديل السجل الرئيسي سيكسر قيد المفتاح الثانوي، ويترك بعض السجلات في الجدول الثانوي تشير إلى سجل غير موجود في الجدول الرئيسي.
Cascade	يؤدّي حذف أو تعديل السجلّ الرئيسي إلى حذف أو تعديل كلّ السجلات الثانويّة المرتبطة به.. فمثلا، سيؤدّي حذف أحد المؤلفين من جدول المؤلفين، إلى حذف كلّ كتبه من جدول الكتب.

SetDefault	يؤدّي حذف أو تعديل السجلّ الرئيسي إلى وضع القيم الافتراضيّة في حقول المفتاح الثانوي المرتبطة به.
SetNull	يؤدّي حذف أو تعديل السجلّ الرئيسي إلى إفراغ حقول المفتاح الثانوي المرتبطة لتصير بها القيمة DbNull.

قاعدة التحديث UpdateRule:

توضّح الفعل الذي يتمّ اتخاذه مع السجلات الفرعية، عند تغيير قيم السجل الرئيسي الذي تنتمي إليه، وهي تأخذ إحدى قيم المرقم Rule التي تعرفنا عليها الخاصية السابقة.

-١٣-

عــروض البيــانــات
Data Views

تتيح لك العروض View عرض بعض أو كل سجلات الجدول بالترتيب الذي تريــده، دون التأثير على سجلات الجدول الأصلي.. هذا يمنحك مرونة عاليــة عنــد عــرض البيانــات للمستخدم، دون الحاجة إلى إعادة إرسال استعلامات مختلفة إلى قاعدة البيانات.

وفكرة العرض بسيطة، فكائن العرض يحتوي على فهرس Index يشــير إلــى ســجلات الجدول، ولا يحتوي على السجلات نفسها.. هذا يجعل كــائن العــرض ســريعا فــي أداء عمليات الترشيح Filtering والترتيب Sorting والبحــث Searching، دون أن يســتهلك مساحة كبيرة في الذاكرة!

لاحظ أن فهرس السجلات يتم إنشاؤه عند إنشاء كائن العرض، ويعاد إنشاؤه مرة أخرى إذا تم تغيير طريقة الترتيب أو الترشيح.. لذا مــن الأفضــل أن تحــدد مواصــفات الترتيــب والترشيح عند إنشاء كائن العرض لتوفر على نفسك الوقت الضــائع فــي إعــادة إنشــاء الفهرس.

وفي هذا الفصل سنتعرف على الفئات التي تتيح لنا إنشاء العروض والتعامل معها.

واجهة قائمة الربط
IBindingList Interface

هذه الواجهة ترث واجهة القائمة IList، وهي تقدم الوسائل اللازمة للتعامل مع مصدر البيانات Data Source من خلال أدوات ربط البيانات Data-Bound Controls.

وتمتلك هذه الواجهة الخصائص التالية:

السماح بالتحرير AllowEdit:

تعيد true إذا كان من الممكن تغيير قيمة أي عنصر في القائمة.

السماح بالجديد AllowNew:

تعيد true إذا كان من الممكن إضافة عنصر جديد إلى القائمة باستخدام الوسيلة AddNew.

السماح بالحذف AllowRemove:

تعيد true إذا كان من الممكن حذف عنصر من القائمة باستخدام الوسيلتين Remove و RemoveAt.

تدعم الترتيب SupportsSorting:

تعيد true إذا كانت القائمة تسمح بترتيب عناصرها.. وإذا كانت قيمة هذه الخاصية false، فستؤدي محاولة استخدام أي من خصائص الترتيب إلى حدوث خطأ من النوع NotSupportedException.

تدعم البحث SupportsSearching:

تعيد true إذا كانت القائمة تتيح البحث في عناصرها، وتتيح ترتيبها.

هل هي مرتبة IsSorted:

تعيد true إذا كانت عناصر القائمة مرتبة.

اتجاه الترتيب SortDirection:

توضح اتجاه ترتيب القائمة، وهي تعيد إحدى قيمتي المرقم ListSortDirection التاليتين:

Ascending	ترتيب تصاعدي.
Descending	ترتيب تنازلي.

خاصية الترتيب SortProperty:

تعيد كائنا من نوع فئة واصف الخاصية PropertyDescriptor Class، وهو يحتوي المعلومات اللازمة لمعرفة الخاصية المستخدمة في ترتيب عناصر القائمة، وذلك إذا كانت القائمة تحتوي على كائنات Objects تمتلك خصائص مختلفة.. مثلا عند التعامل مع كلئن عرض فيه سجلات أحد الجداول، يمكنك أن تحدد أحد القول لترتيب السجلات تبعا له.

تدعم التنبيه عن التغيير SupportsChangeNotification:

تعيد true إذا كانت القائمة تطلق الحدث ListChanged عند حدوث تغير في عناصرها.

كما تمتلك هذه الواجهة الوسائل التالية:

إضافة فهرس AddIndex:

أرسل إلى هذه الخاصية كائنا من النوع PropertyDescriptor، لتقوم بإضافة الخاصية التي يشير إليها، إلى مجموعة الفهارس المستخدمة في البحث في عناصر القائمة.

حذف فهرس RemoveIndex:

أرسل إلى هذه الخاصية كائنا من النوع PropertyDescriptor، لتقوم بحذف الخاصية التي يشير إليها، من مجموعة الفهارس المستخدمة في البحث في عناصر القائمة.

إضافة عنصر جديد AddNew:

لا تستقبل هذه الوسيلة أية معاملات، ولكنها تنشئ عنصرا جديدا من نفس نوع عناصر القائمة، وتضيفه إليها، وتعيد إليك كائنا Object يحمل مرجعا إلى هذا العنصر المضاف.

تنفيذ الترتيب ApplySort:

تقوم بترتيب عناصر القائمة، وهي تستقبل معاملين:

- كائن من النوع PropertyDescriptor، يوضح الخاصية التي سيتم ترتيب العناصر على أساسها.
- إحدى قيمتي المرقم ListSortDirection توضح اتجاه الترتيب.

حذف الترتيب RemoveSort:

تعيد عناصر القائمة إلى ترتيبها الأصلي الذي كانت عليه قبل استدعاء الوسيلة ApplySort.

بحث Find:

تقوم بالبحث في عناصر القائمة، وهي تستقبل معاملين:

- كائن من النوع PropertyDescriptor، يوضح الخاصية التي سيتم البحث عن قيمتها.
- كائن Object يحمل القيمة المراد البحث عنها.

كما تمتلك هذه الواجهة الحدث التالي:

القائمة تغيرت ListChanged:

ينطلق عند حدوث تغير في عناصر القائمة.. والمعامل الثاني e لهذا الحدث من النوع ListChangedEventArgs، وهو يمتلك الخصائص التالية:

	ListChangedType	تعيد إحدى قيم المرقم ListChangedType التي توضح نوع التغيير الذي حدث، وهي: - Reset: حدث تغير في عدد كبير من العناصر. - ItemAdded: إضافة عنصر. - ItemDeleted: حذف عنصر. - ItemMoved: تغير موضع عنصر. - ItemChanged: تغيرت قيمة عنصر. - PropertyDescriptorAdded: إضافة واصف خاصية. - PropertyDescriptorDeleted: حذف واصف خاصية. - PropertyDescriptorChanged: تغيير واصف خاصية.

NewIndex	تعيد رقم العنصر الذي تغير في القائمة.	
OldIndex	تعيد رقم العنصر قبل تغيير موضعه في القائمة.	
PropertyDescriptor	تعيد كائنا من النوع PropertyDescriptor، يحتوي على واصف الخاصية، وذلك إذا كان التغيير قد حدث لواصف إحدى الخصائص.	

واجهة القائمة محددة النوع
ITypedList Interface

تحصل هذه الواجهة على خصائص العنصر الذي سيتم الارتباط Binding به، وهي تمتلك الوسيلتين التاليتين:

معرفة اسم القائمة GetListName:

تعيد اسم القائمة التي سيتم الارتباط بها.

معرفة خصائص العنصر GetItemProperties:

أرسل إلى هذه الخاصية مصفوفة من النوع PropertyDescriptor بها الكائنات التي سيتم الارتباط بها في القائمة، لتعيد إليك مجموعة من النوع PropertyDescriptorCollection، بها خصائص هذه الكائنات.. لاحظ أنك لو أرسلت null كمعامل، فستحصل على مجموعة بها عنصر واحد فقط، وهو واصف الخاصية للقائمة نفسها.

فئـــة مديـــر العـــرض
DataViewManager Class

هذه الفئة تمثـــل الـــواجهتين IBindingList و ITypedList، كمـــا أنهـــا تـــرث الفئـــة MarshalByValueComponent، مما يتيح لك إضافتها إلى صينية مكونات النمـــوذج، وإن كان عليك أن تضيفها أولا إلى صندوق الأدوات.

وتتيح لك هذه الفئة لك التحكم في كيفية عرض سجلات كل جداول في مجموعة البيانات.

ويمكن الحصول على مـــدير العـــرض مـــن مجموعـــة البيانـــات، باســـتخدام الخاصـــيّة DefaultViewManager، كالتالي:

```
var DVM = Ds.DefaultViewManager;
```

ولحدث إنشاء هذه الفئة صيغتان:

١- الأولى بدون معاملات.

٢- والثانية تستقبل مجموعة البيانات التي يتعامل معها مدير العرض.. مثال:

```
var DVM =  new DataViewManager (Ds);
```

ويمتلك مدير العرض الخاصيتين التاليتين:

مجموعة البيانات DataSet:

تقرأ أو تغيّر مجموعة البيانات التي يتعامل معها مدير العرض.

إعدادات عرض البيانات DataViewSettings:

تعيد مجموعة إعدادات العرض DataViewSettingCollection، وهي مجموعـــة تمثل الواجهة ICollection، وتحتوي على إعـــدادات العـــرض الخاصـــة بجـــداول مجموعة البيانات، وكل عنصر من عناصرها من نـــوع الفئـــة DataViewSetting التي سنتعرف عليها لاحقا.

لاحظ أن هذه المجموعة للقراءة فقط، لهذا لا يمكنك إضافة أية عناصر إليها.. لكن كل جدول يضاف إلى مجموعة البيانات، يضيف عنصر إعدادات العرض الخاص به إلى هذه المجموعة تلقائيا.

ويمكنك الحصول على كائن إعدادات الجدول من هذه المجموعة، إما باستخدام رقم الجدول أو اسمه أو كائن الجدول DataTable الذي يشير إليه.. مثال:

```
DataViewSetting Vs =
    Ds.DefaultViewManager.DataViewSettings[0];
```

ويمتلك مدير العرض الوسيلة التالية:

إنشاء عرض بيانات CreateDataView:

أرسل إلى هذه الوسيلة كائن الجدول DataTable لتنشئ عرضا View لسجلاته، تبعا للإعدادات الخاصة بهذا الجدول في مجموعة الإعدادات DataViewSettings..

وتعيد هذه الوسيلة كائن العرض DataView الذي تم إنشاؤه.. مثال:

```
DataTable TblAuthors = Ds.Tables["Authors"];
DataView DV =
    Ds.DefaultViewManager.CreateDataView(TblAuthors);
```

فئة إعدادات العرض DataViewSetting Class

تحتوي هذه الفئة على إعدادات العرض الافتراضيّة التي يتمّ استخدامها مع الجداول المعروضة.

وليس لهذه الفئة حدث إنشاء، ولكن يمكنك الحصول على نسخة منها خاصة بأحد الجداول من خلال مجموعة إعدادات العرض DataViewSettings كالتالي:

```
DataViewSetting Vs =
    Ds.DefaultViewManager.DataViewSettings["Authors"];
```

وتمتلك هذه الفئة الخصائص:

Table:

تعيد كائن الجدول DataTable الذي ينتمي إليه كائن الإعدادات الحالي.

مدير العرض DataViewManager:

تعيد كائن مدير العرض DataViewManager الذي يحتوي كائن الإعدادات الحالي.

مرشح حالة الصفوف RowStateFilter:

تحدد حالة الصفوف التي تريد عرضها، وهي تأخذ إحدى قيم المرقم DataViewRowState التالية:

None	أي حالة.
Unchanged	عرض الصفوف التي لم تتغير.
Added	عرض الصفوف المضافة.
Deleted	عرض الصفوف المحذوفة.
OriginalRows	النسخة الأصلية للصفوف Original Version.

CurrentRows	النسخة الحالية للصفوف Current Version.
ModifiedOriginal	النسخة الأصلية للسجلات التي تم تعديلها.
ModifiedCurrent	النسخة الحالية للسجلات التي تم تعديلها.

مرشح الصفوف RowFilter:

تقرأ أو تغير الشرط الذي يتم على أساسه اختيار السجلات التي يعرضها كائن العرض.. ويتم تكوين الشرط في هذه الخاصية، بنفس الطريقة المستخدمة في تكوين شرط الخاصية DataRow.Expression.

والمرشح التالي يعرض الكتب التي يقل ثمنها عن ٥ جنيهات:

```
Vs.RowFilter = "Price < 5";
```

الترتيب Sort:

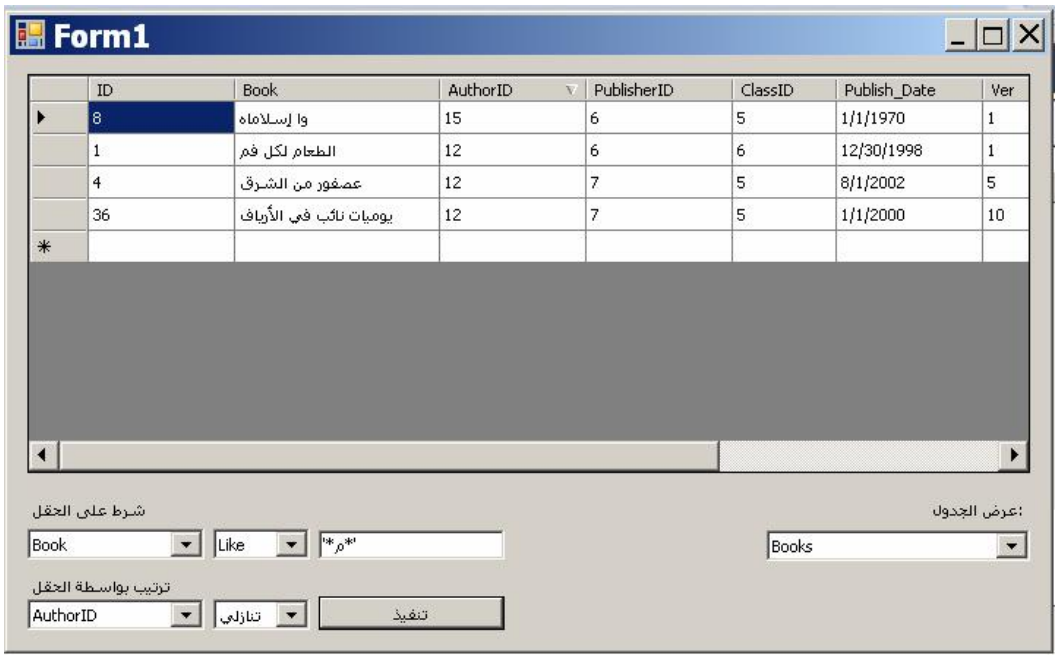
تحدد النص المستخدم في ترتيب السجلات المعروضة، وهو يتكون من اسم الحقل المستخدم في الترتيب، متبوعا باتجاه الترتيب.. مثلا: لترتيب صفوف جدول الكتب تنازليا على حسيب اسم الكتاب، استخدم القيمة التالية لهذه الخاصية:

```
Vs.Sort = "Book DESC";
```

تطبيق الترتيب الافتراضي ApplyDefaultSort:

إذا جعلت قيمة هذه الخاصية true، فسيتم تطبيق الترتيب الافتراضي للصفوف (كما في الجدول الأصلي).. والقيمة الافتراضية لهذه الخاصية هي false، وفي هذه الحالة يتم تطبيق الترتيب الموضح في الخاصية Sort.

ويعتبر المشروع Views تدريبا جيدا على استخدام مدير العروض وإعدادات العرض.. في هذا المشروع نتيح للمستخدم عرض جداول قاعدة الكتب، باختيار الجدول الذي يريده من القائمة المنسدلة "عرض الجدول"، كما نتيح له اختيار مرشح الصفوف وطريقة الترتيب كما هو موضح في الصورة:



دعنا نفهم كيف يعمل هذا المشروع:

- في حدث تحميل النموذج Load، نستخدم مهيئات الجداول لملء مجموعة البيانات، ثم نضيف الجداول إلى القائمة المركبة LstTables.
- في حدث تغير العنصر المحدد في قائمة الجداول SelectedIndexChanged، نعرض سجلات الجدول الذي اختاره المستخدم في جدول العرض، ونفرغ قائمة الحقول LstFields وقائمة الترتيب LstSort من محتوياتهما، ونضيف إلى كل منهما أسماء أعمدة الجدول الحالي.. ونظرا لأن المستخدم قد يريد عرض كل السجلات بدون ترتيب، فسنضيف إلى كل مجموعة عنصرا إضافيا اسمه (None) وسنجعله أول عنصر في كل منهما.

وللتحكم في طريقة عرض الجدول، سنستخدم إعدادات العرض الخاصة به.. ولتسهيل الكود في باقي البرنامج، سنضع كائن إعدادات الجدول الحالي في متغير معرف على مستوى النموذج اسمه ViewSettings:

```
DataTable T = LstTables.SelectedItem;
ViewSettings = Ds.DefaultViewManager.DataViewSettings[T];
```

- في حدث تغير العنصر المحدد SelectedIndex في قائمة الحقول LstFields، سنعطل قائمة معاملات المقارنة LstOp ومربع نص القيمة TxtValue، وذلك إذا اختار المستخدم العنصر (None) وهو العنصر رقم صفر في القائمة، أما إذا اختار المستخدم أحد أعمدة الجدول، فسنفعّل قائمة المعاملات وجدول القيمة حتى يمكن استخدامهما في تكوين مرشح السجلات.. كل هذا يمكن فعله بالسطرين التاليين فحسب:

```
LstOp.Enabled = (LstField.SelectedIndex != 0);
TxtValue.Enabled = LstOp.Enabled;
```

- في حدث تغير العنصر المحدد SelectedIndex في قائمة حقل الترتيب LstSort: إذا اختار المستخدم العنصر (None) وهو العنصر رقم صفر في القائمة، فسنعطل قائمة اتجاه الترتيب LstSortOrder ونجعل للخاصية ApplyDefaultSort الخاصة بإعدادات العرض القيمة true لعرض السجلات بترتيبها الأصلي.. أما إذا اختار المستخدم أحد أعمدة الجدول، فسنفعّل قائمة اتجاه الترتيب ونضع false في الخاصية ApplyDefaultSort لنعرض السجلات بالترتيب الذي يريده المستخدم.. كل هذا يمكن فعله بالسطرين التاليين فحسب:

```
LstSortOrder.Enabled = (LstSort.SelectedIndex != 0);
ViewSettings.ApplyDefaultSort = ! LstSortOrder.Enabled;
```

- أخيرا، في حدث ضغط الزر "تنفيذ"، سنقوم بتنفيذ خيارات العرض التي اختارها المستخدم، حيث سنكون مرشح السجلات كالتالي:

```
if (LstField.SelectedIndex == 0)
    ViewSettings.RowFilter = "";
else
    ViewSettings.RowFilter = LstField.Text + " " +
      LstOp.Text + " " + TxtValue.Text.Trim( );
```

لاحظ أن المستخدم هو المسئول عن كتابة القيمة المناسبة بالشكل الصحيح في مربع النص.. فعليه مثلا أن يكتب رقما في مربع النص إذا كان الحقل الذي اختاره

رقميا، وأن يكتب نصا ويضعه بين علامتي التنصيص ' ' إذا كان الحقل يتعامل مع نصوص أو تواريخ.. وإذا اختار المعامل IN فعليه أن يفصل بين القيم بالعلامة , ويضع كل ما كتبه بين قوسين ( ).. وهكذا.

لاحظ أيضا أن وضع شرط خاطئ في الخاصية RowFilter لا يسبب خطأ في البرنامج، لكنه يجعل مدير الإعدادات يتجاهل هذه الإعدادات ولا ينفذها.

بعد هذا سنكون قيمة الخاصية Sort بالكود التالي:

```
if (LstSort.SelectedIndex == 0)
  ViewSettings.Sort = "";
else
  ViewSettings.Sort = LstSort.Text + " " +
  ((LstSortOrder.SelectedIndex == 0) ? "ASC" : "DESC");
```

أخيرا، سننشئ كائن عرض ونعرضه في جدول العرض.. لفعل هذا سنحصل على مدير العروض من كائن إعدادات العرض، ونستخدم الوسيلة CreateDataView الخاصة به كالتالي:

```
var Dv = ViewSettings.DataViewManager.
         CreateDataView(ViewSettings.Table);
Dgv.DataSource = Dv;
```

وهكذا، وبهذا البرنامج البسيط، الذي لم نكتب به الكثير من الكود، صار باستطاعة المستخدم عرض جميع بيانات قاعدة الكتب، وبأي طريقة عرض يحبها!

واجهة ربط قائمة العرض
IBindingListView Interface

هذه الواجهة ترث الواجهة IBindingList، وهي تقدم إمكانيات متقدمة في ترشيح السجلات Filtering وترتيبها Sorting.

وتمتلك هذه الواجهة الخصائص التالية:

المرشح Filter:

تحدد الشرط الذي يتم على أساسه اختيار السجلات لعرضها.. مثل:

"Price > 5"

واصفات الترتيب SortDescriptions:

تعيد مجموعة للقراءة فقط من النوع ListSortDescriptionCollection، وهي تمثل واجهة القائمة IList، وكل عنصر من عناصرها من نوع الفئة ListSortDescription، التي سنتعرف عليها لاحقا.

تدعم الترتيب المتقدم SupportsAdvancedSorting:

تعيد true إذا كان مصدر البيانات يستطع ترتيب السجلات تبعا لقيم أكثر من عمود.

تدعم الترشيح SupportsFiltering:

تعيد true إذا كان مصدر البيانات يسمح بانتقاء بعض السجلات تبعا لشرط معين.

وتمتلك هذه الواجهة الوسيلة التالية:

إزالة المرشح RemoveFilter:

تلغي عملية الترشيح، وتعيد عرض جميع السجلات بدون مراعاة شرط الترشيح.

فئة واصف ترتيب القائمة
ListSortDescription Class

تحتوي هذه الفئة على المعلومات اللازمة لترتيب عناصر قائمة البيانات.
ويستقبل حدث إنشاء هذه الفئة معاملين:

- واصف الخاصية PropertyDescriptor التي سيتم الترتيب على أساسها.
- إحدى قيمتي المرقم ListSortDirection التي توضح اتجاه الترتيب.

وتمتلك هذه الفئة الخاصيتين التاليتين:

واصف الخاصية PropertyDescriptor:

تقرأ أو تغير واصف الخاصية PropertyDescriptor التي سيتم الترتيب على أساسها.

اتجاه الترتيب SortDirection:

توضح اتجاه الترتيب، وهي تأخذ إحدى قيمتي المرقم ListSortDirection وقد تعرفنا عليهما سابقا.

فئة عرض البيانات DataView Class

هذه الفئة تمثل الواجهات IBindingListView و IBindingList و ITypedList، كما أنها ترث الفئة MarshalByValueComponent، مما يتيح لك إضافتها إلى صينية مكونات النموذج، وإن كان عليك أن تضيفها أولا إلى صندوق الأدوات.

ويمكنك استخدام كائن عرض البيانات للحصول على طريقة عرض مختلفة لسجلات الجدول، سواء بترتيبها أو باختيار جزء من السجلات تبعا لشرط معين.. كما يمكنك ربط هذا الكائن بأدوات عرض البيانات كما سنرى في الفصل التالي، وبهذا تتحكم في طريقة عرض البيانات في برنامجك.

ويمكنك الحصول على كائن العرض الافتراضي للجدول باستخدام الخاصية DefaultView كالتالي:

```
DataView Dv = TblAuthors.DefaultView;
```

ومن ثم يمكنك تغيير خصائصه لتغيير طريقة عرضه.

ولحدث إنشاء هذه الفئة الصيغ التالية:

١- الصيغة الأولى بدون معاملات.

٢- الصيغة الثانية تستقبل كائن الجدول DataTable الذي سيتعامل معه كائن العرض.

٣- الصيغة الثالثة تزيد على الصيغة السابقة بثلاثة معاملات، هي بالترتيب:

- نص يحتوي على شرط ترشيح السجلات RowFilter.
- نص يحتوي على شرط ترتيب السجلات Sort.
- إحدى قيم المرقم DataViewRowState، توضح حالة السجلات التي يتم عرضها.

والمثال التالي ينشئ كائن عرض، يعرض كتب توفيق الحكيم مرتبة تصاعديا على حسب أسمائها:

```
var Dv = new DataView(Ds.Books,
    "Parent(FK_Books_Authors).Author = 'توفيق الحكيم'",
    "Book", DataViewRowState.CurrentRows);
```

ويمكنك تجربة هذا المثال بضغط الزر "كتب توفيق الحكيم" في المشروع Views.

وبالإضافة إلى ما تمثله من خصائص الواجهات IBindingListView و IBindingList و ITypedList، تمتلك فئة العرض بعض الخصائص المماثلة في اسمها ووظيفتها لخصائص كائن إعدادات العرض DataViewSetting، مثل:

الجدول Table

مدير العرض DataViewManager

مرشح الصفوف RowFilter

مرشح حالة الصفوف RowStateFilter

الترتيب Sort

تطبيق الترتيب الافتراضي ApplyDefaultSort

والكود التالي يجعل كائن العرض يحتوي على أسماء الكتب التي تبدأ أسماؤها بالحروف من الألف إلى التاء فقط، ويرتبها تنازليّا عن طريق اسم الكتاب ورقم المؤلّف:

```
DataTable T = Ds.Tables["Books"];
DataView Dv = T.DefaultView;
Dv.RowFilter = " Book < 'ث' ";
Dv.Sort = "Book, AuthorID DESC";
```

كما تمتلك فئة العرض الخاصية التالية:

المفهرس Indexer:

يعيد كائن عرض الصف DataRowView الموجود في كائن العرض الحالي في الموضع المرسل كمعامل.. مثال:

```
DataRowView R = Dv[0];
```

لاحظ أن كائن العرض يعمل كقائمة، لهذا تستطيع المرور عبر كل صفوف كائن العرض باستخدام حلقة التكرار foreach كالتالي:

```
foreach (DataRowView R in Dv)
    MessageBox.Show(R["Book"].ToString( ))
```

وتمتلك فئة العرض الوسائل التالية:

إضافة سجل جديد AddNew:

تضيف سجلا جديدا إلى كائن العرض، تحتوي خاناته على القيم الافتراضية في الأعمدة التي لها قيم افتراضية، وعلى العدم DbNull في الأعمدة التي تسمح بهذا.. لاحظ أنّ هذا السجلّ سيضاف أيضا إلى الجدول الأصليّ أيضا الموجود في مجموعة البيانات، وعند القيام بعملية التحديث سيتم حفظه في قاعدة البيانات.. هذا يجعلك مطمئنا إلى أن السجلات الجديدة التي يضيفها المستخدم إلى جدول عرض DataGridView مرتبط بكائن عرض، ستتم إضافتها تلقائيا إلى مجموعة البيانات، وستعامل معاملة السجلات العادية.

وتعيد هذه الوسيلة كائن عرض الصف DataRowView الذي يشير إلى الصف الجديد الذي تمت إضافته، مما يتيح لك تغيير قيم خاناته.

حذف Delete:

تحذف السجلّ الذي ترسل إليها رقمه كمعامل.. هذا السجل سيتم حذفه في الحال من كائن العرض، وستكون أمامنا حالتان بالنسبة للجدول الأصلي:

١- إذا كان السجل مضافا إلى الجدول (RowState = Added)، فسيتم حذفه نهائيا من الجدول، ولن يمكنك استعادته باستدعاء الوسيلة DataTable.RejectChanges.

٢- إذا كان السجل أحد سجلات الجدول الأساسية، فسيظل في الجدول الأصلي مع جعل حالته محذوفا (RowState = Deleted).. ولو استدعيت الوسيلة DataTable.RejectChanges فستعيد السجل إلى كائن العرض DataView، مع تحويل حالته في الجدول الأصلي إلى Unchanged.

لاحظ أن ترتيب السجلات في كائن العرض قد يختلف عن ترتيبها المجموعة Rows في الجدول الأصلي، سواء بسبب إعادة ترتيبها، أو بسبب أخذ جزء فقط من سجلات الجدول الأصلي تحقق شرطا معينا.. لهذا عليك أن تستخدم الوسيلة Find لتبحث في كائن العرض عن السجل الذي تريد حذفه لتحصل على رقمه.. والمثال التالي يفترض أن Dv هو كائن عرض يعرض بعض سجلات جدول الكتب، ويستخدمه لحذف أحد الكتب:

```
Dv.Sort = "Book";
var I = Dv.Find("الطعام لكل فم");
if (I != -1)
  Dv.Delete(I);
```

بحث Find:

أرسل إلى هذه الوسيلة كائنا Object، لتبحث عن القيمة التي يحملها في العمود المستخدم لترتيب السجلات (كما تحدده الخاصية Sort).. وتعيد هذه الوسيلة رقم أول سجلّ يحتوي على القيمة المطلوبة، وإذا لم تعثر على أيّ سجلّ، تعيد -١.

وتوجد صيغة ثانية لهذه الوسيلة، تستقبل مصفوفة كائنات، وذلك إذا كان كائن العرض يستخدم أكثر من عمود في عملية الترتيب.

وقد استخدمنا هذه الوسيلة في المشروع CustomDataSet.. في هذا المشروع يستطيع المستخدم إدخال أسماء الطلاب في النافذة الرئيسية، وضغط الرابط الموجود في عمود الدرجات لعرض نافذة فيها درجات الطالب في المواد المختلفة.. لفعل هذا، اتبعنا الخطوات التالية:

- حصلنا على كائن العرض الافتراضي الخاص بجدول الطلبة، باستخدام الخاصية DsStudents.Students.DefaultView، ومررنا إليه رقم صف التلميذ الحالي (ممثلا بالمتغير Idx)، لنحصل على كائن عرض هذا الصف DataRowView.

```
var StdRel = DsStudents.Students.ChildRelations[
    0].RelationName;
var RowView = DsStudents.Students.
    DefaultView[Idx];
```

- أرسلنا إلى الوسيلة CreateChildView الخاصة بكائن الصف، اسم العلاقة التي تربط جدول التلاميذ بجدول الدرجات، لنحصل على كائن عرض يحتوي على درجات التلميذ الحالي، لنستخدمه كمصدر بيانات لجدول العرض الذي يعرض الدرجات.. هذا يجعل جميع الصفوف التي تضاف إلى هذا الجدول، تضع تلقائيا رقم هذا التلميذ في العمود StudentID:

```
var Grades = RowView.CreateChildView(StdRel);
```

- لو كانت هذه أول مرة نعرض فيها درجات هذا التلميذ، فسيظهر الجدول فارغا.. هذا غير مقبول، لأن علينا أن نعرض للمستخدم أسماء المواد، ليكتب هو درجات الطالب المناظرة لها مباشرة.. ونظرا لأننا نسمح للمستخدم بتحرير جدول المواد الدراسية بضغط الزر "عرض المواد"، فمن الممكن أن تضاف مواد جديدة لم نقم بإضافتها إلى نافذة الدرجات من قبل.. لهذا علينا أن نستخدم الوسيلة Find للتأكد من أن رقم المادة غير موجود، ومن ثم نضيفها.. السبب في هذا أن تكرار نفس المادة مع نفس الطالب سيؤدي إلى حدوث خطأ بسبب قيد التفرد المفروض على هذين الحقلين.. لكن استخدام الوسيلة Find يوجب علينا أولا أن نستخدم الحقل SubjectID كمفتاح للترتيب في الخاصية Sort:

```
Grades.Sort = "SubjectID";
```

بعد هذا سنمر على كل مادة في جدول المواد، ونبحث عن رقمها في جدول الدرجات، ولو لم تكن موجودة (نتيجة البحث = -١)، علينا استخدام الوسيلة AddNew لإضافة صف جديد إلى كائن العرض.. هذا الصف يكون فارغا،

ما عدا الحقل StudentID الذي يأخذ تلقائيا رقم التلميذ الذي نتعامل معه.. لهذا علينا نضيف رقم المادة إلى هذا الصف الجديد.. لاحظ أن اسم المادة سيضاف تلقائيا لأنه عمود محسوب مبني على قيمة الحقل StudentID.. لكن هذا لن يحدث طالما ظل هذا الصف الجديد هو الصف الحالي.. لحسن الحظ أن الصف الحالي يتغير تلقائيا إذا أضفنا صفا جديدا بعده، لكن المشكلة تظل في آخر صف نضيفه.. لهذا علينا استخدام الوسيلة EndEdit لإنهاء تحرير الصف الحالي.. هذا سيجبر العمود Subject على حساب قيمته، وعرض اسم المادة تلقائيا:

```
var SubjRows = DsStudents.Subjects.Rows;
for (var i = 0; i < SubjRows.Count; i++)
{
   var SbjID = SubjRows[i]["ID"];
   if (Grades.Find(SbjID) == -1)
   {
     var Rv = Grades.AddNew( );
     Rv["SubjectID"] = SbjID;
     Rv.EndEdit( );
   }
}
```

ومن الأذكى أن تخفي رقم التلميذ ورقم المادة عن المستخدم، لأنه سيتعامل فقط مع المادة ودرجة الطالب فيها.. وستجد هذا الكود كاملا في الإجراء ShowGrades الذي يتم استدعاؤه من الحدث CellContentClick في النموذج FrmStudent، ومن الحدث SelectedIndex في النموذج FrmGrades.

بحث عن الصفوف FindRows:

مماثلة للوسيلة السابقة في صيغتيها، ولكنها تعيد مصفوفة من النوع DataRowView، تحتوي على كلّ السجلات التي تحتوي على القيمة المطلوبة.

التحويل إلى جدول ToTable:

تنشئ جدولا جديدا وتنسخ إليه السجلات الموجودة في كائن العرض الحالي، وتعيد إليك كائن جدول DataTable يشير إليه.. ولهذه الوسيلة الصيغ التالية:

١- الصيغة الأولى بدون معاملات.

٢- الصيغة الثانية تستقبل نصا لتستخدمه كاسم للجدول.

٣- الصيغة الثالثة تستقبل معاملين:

- قيمة منطقية إذا جعلتها true فسيتم استبعاد الصفوف المكررة التي تتشابه قيم أي من خاناتها مع أي صفوف أخرى.
- مصفوفة نصية، تحتوي على أسماء الأعمدة التي تريد نسخها إلى الجدول.. وستظهر الأعمدة في الجدول بنفس ترتيبها في المصفوفة.. لاحظ أن هذه هي الطريقة الوحيدة التي تستطيع بها الحصول على جدول يختلف في عدد أعمدته وترتيبها عن الجدول الأصلي.

٤- الصيغة الرابعة تحتوي على معاملات الصيغتين السابقتين معا.

واجهة الكائن القابل للتحرير
IEditableObject Interface

تقدم الوسائل اللازمة للتعامل مع كائن يمكن وضعه في حالة التحرير (مثل كائن عرض صف البيانات DataRowView)، مع القدرة على إلغاء التغييرات أو قبولها.

وتمتلك هذه الواجهة الوسائل التالية:

بدء التحرير BeginEdit:

تبدأ عملية التحرير.

إلغاء التحرير CancelEdit:

تلغي عملية التحرير وتعيد إلى الكائن قيمه الأصلية.

إنهاء التحرير EndEdit:

تنهي عملية التحرير وتحفظ في الكائن القيم التي تم إدخالها أثناء عملية التحرير.

واجهة معلومات خطأ البيانات
IDataErrorInfo Interface

تقدم معلومات عن الخطأ الذي حدث في السجل، وهي تمتلك الخاصيتين التاليتين:

الخطأ Error:

تعيد نصا يشرح الخطأ الذي حدث في السجل.

خطأ العنصر Item:

أرسل إلى هذه الخاصية اسم العمود، لتعيد إليك نصا يشرح الخطأ الذي حدث في هذا العمود في السجل الحالي.

واجهة التنبيه بتغير خاصية
INotifyPropertyChanged Interface

تطلق الحدث PropertyChanged عند تغير قيمة خاصية معينة.. وتمتلك هذه الواجهة الحدث التالي:

الخاصية تغيرت PropertyChanged:

ينطلق عندما يحدث تغيير في قيمة الخاصية.. والمعامل الثاني e لهذا الحدث من النوع PropertyChangedEventArgs، وهو يمتلك الخاصية PropertyName التي تعيد اسم الخاصية التي تغيرت.

فئة عرض صف البيانات DataRowView Class

هـــــذه الفئـــــة تشمـــــل الواجهـــــات IEditableObject و IDataErrorInfo و INotifyPropertyChanged، وهي تحتوي على أحد السجلات المعروضة في كـــائن العرض DataView.. ولا يوجد لهذه الفئة حدث إنشاء، ولا يمكنك الحصول على نســخة منها إلا من خلال كائن العرض.

وتمتلك هذه الفئة الخصائص التالية:

عرض البيانات DataView:

تعيد كائن العرض DataView الذي ينتمي إليه الصف الحالي.

السجل Row:

تعيد كائن سجل البيانات DataRow الذي يعرضه كائن عرض الصف الحاليّ.

في حالة التحرير IsEdit:

تعيد true إذا كان السجلّ الحاليّ يتمّ تحريره.

هل هو جديد IsNew:

تعيد true إذا كان السجلّ الحاليّ جديدا، وذلـــك إذا تـــم إنشـــاؤه بواســـطة الوســـيلة DataView.AddNew.. مثال:

```
var Rv = Dv.AddNew( );
MessageBox.Show(Rv.IsNew.ToString( )) // true
```

المفهرس Indexer:

يقرأ أو تغير قيمة إحدى خانات الصف الحالي، التي يحددها اسم العمود أو رقمه المرسل كمعامل.. والمثال التالي يعرض قيم كل الخانات الموجودة في كل صفوف كائن العرض Dv:

```
foreach (DataRowView R in Dv)
{
        for (var I = 0; I < Dv.Table.Columns.Count; I++)
        {
                MessageBox.Show(R[I].ToString( ));
        }
}
```

نسخة السجل RowVersion:

تعيد إحدى قيم المرقم DataRowVersion، التي توضح نسخة السجل المعروضة حاليا.

وتمتلك فئة عرض الصف الوسيلتين التاليتين:

حذف Delete:

تحذف السجل الحالي من كائن العرض، ولها نفس تأثير الوسيلة DataView.Delete على السجل الأصلي في جدول البيانات.

إنشاء عرض تابع CreateChildView:

تستخدم هذه الوسيلة إذا كان عارض الصف الحالي ينتمي إلى جدول أساسيّ يرتبط بعلاقة بجدول ثانويّ.. وتستقبل هذه الوسيلة اسم العلاقة أو كائن العلاقة DataRelation الذي يمثلها، وتعيد كائن عرضٍ DataView يحتوي على كلّ سجلات الجدول الثانويّ التابعة لهذا السجلّ.. والمثال التالي يعيد كائن عرض يحتوي

على كلّ الكتب التي ألفها أول مؤلف في جدول الكتب، بافتراض أن العلاقة بين جدولي الكتب والمؤلفين اسمها AuthorsBooks:

```
DataRowView Rv = Ds.Tables["Authors"].DefaultView[0];
DataView Dv = Rv.CreateChildView("AuthorsBooks");
```

وقد استخدمنا هذه الوسيلة في المشروع DataSetSample، لعرض كتب المؤلف المحدد حاليا.. فعند تغير السجل المحدد في جدول عرض المؤلفين، ينطلق الحدث DataGridView.RowEnter، وفيه نفعل ما يلي:

- نستخدم الخاصية e.RowIndex لمعرفة رقم الصف المحدد حاليا.
- نظرا لأن رقم الصف في جدول العرض، هو نفسه رقم الصف في جدول المؤلفين بسبب الربط بينهما Binding، فسنرسله إلى مفهرس الفئة DefaultView للحصول على كائن عرض هذا الصف:

```
var TblAuthors = Ds.Tables["Authors"];
var RowView = TblAuthors.DefaultView[e.RowIndex];
```

لاحظ أن رقم كل صف في جدول العرض يظل ثابتا مهما غير المستخدم ترتيب الصفوف المعروضة!

- نستخدم الوسيلة CreateChildView للحصول على كائن عرض يحتوي على كتب المؤلف المحدد حاليا، ونعرضها في جدول عرض الكتب الموجود في النصف السفلي من النافذة:

```
DataView DvBooks =
    RowView.CreateChildView("AuthorsBooks");
DgBooks.DataSource = DvBooks;
```

لاحظ أنك عندما تبدأ تحرير السجل الجديد الموجود في نهاية جدول عرض الكتب، فإن الخانة AuthorID ستأخذ تلقائيا رقم المؤلف الذي ينتمي إليه كائن العرض الحالي.. هذا يريحك من كتابة أي كود إضافي، كما يسمح لك بإخفاء العمود AuthorID من الجدول دون قلق، فهو سيأخذ القيمة الصحيحة آليا دون أن يشغل المستخدم باله بهذا.. ولإخفاء هذا العمود، أضف هذا السطر إلى نهاية الكود السابق:

```
DgBooks.Columns["AuthorID"].Visible = false;
```

-١٤-

ربـــط البيـــانـــات
Data Binding

في تطبيقات قواعد البيانات، كثيرا ما تحتاج إلى عرض بيانات أحد السجلات في أدوات موضوعة على النموذج، مع وجود أزرار للتحرك إلى السجل التالي أو السجل السابق، ليتمكن المستخدم من التحكم في السجل المعروض حاليا.

ونظرا لأن فعل هذا يدويا قد يحتاج إلى كود طويل ومرهق، فقد قدمت لك دوت نت آلية جاهزة تسمى ربط البيانات Data Binding، تتيح لك ربط الكائنات بأدوات الويندوز بأقل قدر من الكود.

ويسمى الكائن الذي يتم ربطه بالأداة باسم **مصدر البيانات Data Source**.. وتشمل مصادر البيانات الأنواع التالية:

١- الكائنات البسيطة التي تحتوي على بعض الحقول Fields:

مثل كائن الحجم Size Object الذي يحتوي على الحقلين Width و Height.. والمشروع BindingToObject المرفق بأمثلة هذا الكتاب يريك مثالا على هذا الربط.

٢- الكائنات المركبة التي تحتوي على عدة كائنات:

كالمصفوفات Arrays التي تحتوي على أرقام أو نصوص أو كائنات محددة النوع، والمجموعات Collections التي تمثل واجهة القائمة IList.. وتعرض الأدوات في هذه الحالة عنصرا واحدا فقط في نفس اللحظة، وتقدم تقنية الربط الوسائل

اللازمة للتحرك إلى العنصر السابق أو التالي.. والمشروع BindingToArray يريك مثالا على هذا الربط.

٣- الكائنات المعقدة التي تحتوي على مجموعات داخلية:

وهي الكائنات التي تمثل الواجهة IBindingList أو الواجهة ITypedList، مثل مجموعة البيانات DataSet وجدول البيانات DataTable وعرض البيانات DataView ومدير عرض البيانات DataViewManager.. ونظرا لأن مصدر البيانات يحتوي على أكثر من مجموعة داخلية (مجموعة البيانات مثلا تحتوي على أكثر من جدول وأكثر من علاقة)، فيجب أن نحدد الخاصية التي سنأخذ البيانات منها (كاسم الجدول مثلا).. وتسمى هذه الخاصية باسم **عنصر البيانات Data Member**.

وتعرض الأدوات في هذه الحالة سجلا واحدا فقط في نفس اللحظة، وتقدم تقنية الربط الوسائل اللازمة للتحرك إلى السجل السابق أو التالي.. والمشروع BindingToDataSet يريك مثالا على هذا الربط.

واجهة المكون القابل للارتباط
IBindableComponent Interfac

تمتلك هذه الفئة العناصر الأساسية اللازمة لربط الأدوات بمصادر البيانات، وهي:

ارتباطات البيانات DataBindings:

تعيد نسخة من مجموعة ارتباطات الأداة ControlBindingsCollection، تحتوي على كائنات الربط Binding Objects التي تستخدمها الأداة الحالية.. وسنتعرف على الفئة Binding بالتفصيل لاحقا.

محتوى الربط BindingContext:

تقرأ أو تغير كائن محتوى الربط BindingContext الذي تستخدمه الأداة.. وسنتعرف على الفئة BindingContext بالتفصيل لاحقا.

تصفير الارتباطات ResetBindings:

تقوم هذه الوسيلة بإنعاش القيم التي تعرضها الأداة من خلال الارتباط.. هذه الوسيلة ليست على درجة ملموسة من الأهمية.

محتوى الربط تغير BindingContextChanged:

ينطلق هذا الحدث عند تغير قيمة الخاصية BindingContext.

الجدير بالذكر أن فئة الأداة الأم Control Class التي ترثها جميع الأدوات تمثل الواجهة IBindableComponent، ومن ثم فهي تمتلك جميع العناصر السابقة.. هذا معناه أن جميع أدوات الويندوز تصلح للارتباط بمصادر البيانات.

وتسمى الأدوات التي يتم الارتباط بها باسم **الأدوات المرتبطة بالبيانات Data-Bound Controls**، ونظرا لأن كل أداة تمتلك العديد من الخصائص، فيجب عليك أن تحدد الخاصية التي تريدها أن تعرض البيانات.. ولا مانع من أن تربط أكثر من

خاصية من خصائص الأداة، بأكثر من عنصر من عناصر مصدر البيانات.. مثلا: يمكنك ربط الخاصية Text الخاصة بزر الاختيار CheckBox بعنصر بيانات نصي وربط الخاصية Checked بعنصر بيانات منطقي Boolean.. وتسمى خاصية الأداة التي يتم الارتباط بها باسم **عنصر العرض Display Member**، لأنها تعرض قيمة خاصية الكائن.

ملخص:

- **مصدر البيانات Data Source:**
هو الكائن الذي يحتوي على البيانات التي يتم ربطها بالأداة.. ومثال ذلك: الجدول Books في مجموعة البيانات.

- **عنصر البيانات Data Member:**
هو الخاصية التي يتم ربط قيمتها بالأداة.. ومثال ذلك العمود Book في جدول الكتب.

- **الأداة المرتبطة بالبيانات Data-Bound Control:**
هي الأداة التي ترتبط بكائن وتعرض بعض بياناته.. مثل مربع النص TextBox أو اللافتة.. لاحظ أن كل الأدوات تصلح لعرض البيانات، وعليك اختيار ما يناسب وظيفة برنامجك منها.

- **عنصر العرض Display Member:**
هو خاصية الأداة التي ترتبط بعنصر البيانات وتعرض قيمته، كالخاصية Text في مربع النص أو اللافتة.. لاحظ أن كثيرا من خصائص الأدوات تصلح كعناصر عرض، حتى لو لم يرَ المستخدم قيمتها، فأحيانا تريد حفظ قيمة من مصدر البيانات في الأداة لاستخدامها في وظيفة البرنامج، كأن تحفظ رقم المؤلف في الخاصية Tag بينما تعرض اسمه في الخاصية Text.

وتستخدم في تقنية الربط مجموعة من الفئات الموجودة في نطاق الاسم System.Windows.Forms.. دعنا نتعرف على هذه الفئات.

فئة مجموعة الارتباطات
BindingsCollection Class

ترث هذه المجموعة فئة المجموعة الأساسية BaseCollection، وهي مجموعة تقليدية تمثل الواجهة ICollection، وتشتق منها العديد من المجموعات الخاصة بالارتباطات وأدوات ربط البيانات، والتي سنتعرف عليها لاحقا.

وتحتوي مجموعة الارتباطات BindingsCollection على كل الارتباطات التي تم إنشاؤها بين أداة معينة ومصادر البيانات المختلفة.. وكل عنصر من عناصر هذه المجموعة من نوع الفئة Binding التي سنتعرف عليها بعد قليل.

ويمكن الحصول على هذه المجموعة من خلال الخاصية BindingManagerBase.Bindings، كما سنرى بعد قليل.

ولا جديد في هذه المجموعة إلا امتلاكها للحدثين التاليين:

يتم تغيير المجموعة CollectionChanging:

ينطلق هذا الحدث عند تغيير المجموعة، والمعامل الثاني e لهذا الحدث من النوع CollectionChangeEventArgs، وقد تعرفنا عليه من قبل عند التعرف على كائن مجموعة الجداول DataTableCollection.

تم تغيير المجموعة CollectionChanged:

ينطلق هذا الحدث بعد حدوث التغيير في المجموعة فعلا.. والمعامل الثاني لهذا الحدث مماثل لمعامل الحدث السابق.

فئة مجموعة ارتباطات الأداة
ControlBindingsCollection Class

ترث هذه المجموعة الفئة BindingsCollection، ويمكن الحصول عليها من خلال الخاصية Control.Bindings.

وتمتلك هذه المجموعة الوسائل التقليدية للمجموعات، لكن الوسيلة Add الخاصة بها لها الصيغ التالية:

١- الصيغة الأولى تستقبل كائن الارتباط Binding المراد إضافته إلى المجموعة.

٢- الصيغة الثانية تستقبل ثلاثة معاملات:

- اسم عنصر العرض، مثل "Text".
- الكائن Object الذي يعمل كمصدر للبيانات، مثل كائن الحجم Size.
- اسم عنصر البيانات.. فمثلا، لو أردت ربط خاصية الارتفاع الخاصة بكائن الحجم Size، فأرسل إلى هذا المعامل النص "Height".. ويمكنك إرسال نص فارغ "" إلى هذا المعامل، وفي هذه الحالة سترتبط الأداة بالنص الناتج من الوسيلة ToString الخاصة بالكائن.. وإذا كان الكائن يحتوي على كائنات متداخلة (مثل مجموعة البيانات التي تحتوي على جداول، وكل منها تحتوي على أعمدة)، فيجب عليك كتابة مسار الخاصية كاملا بدون اسم الكائن.. فمثلا: يمكنك الارتباط بحقل اسم الكتاب في جدول الكتب باستخدام المسار "Books.Book"، مع إرسال مجموعة البيانات نفسها إلى المعامل الثاني.

٣- الصيغة الثالثة تزيد على الصيغة السابقة بمعامل رابع، إذا جعلت قيمته True فسيتم تطبيق التنسيق Format الخاص بك عند عرض العنصر في الأداة.

٤- الصيغة الرابعة تزيد على الصيغة السابقة بمعامل خامس، يستقبل إحدى قيم المرقم DataSourceUpdateMode، لوضعها في الخاصية DataSourceUpdateMode وسنتعرف عليها بعد قليل.

٥- الصيغة الخامسة تزيد على الصيغة السابقة بمعامل سادس، وهو يستقبل القيمة التي تريد وضعها في عنصر العرض إذا كان عنصر البيانات فارغا Nothing.. لاحظ أن هذا المعامل من النوع Object ليتيح لك إرسال أية قيمة تناسبك.

٦- الصيغة السادسة تزيد على الصيغة السابقة بمعامل سابع، يتيح لك إرسال نص يحمل التنسيق Format الذي تريد تطبيقه على قيمة عنصر البيانات عند وضعها في عنصر العرض.

٧- الصيغة السابعة تزيد على الصيغة السابقة بمعامل ثامن من نوع الواجهة IFormatProvider، لوضع قيمته في الخاصية FormatInfo التي سنتعرف عليها بعد قليل.

وتعيد إليك كل هذه الصيغ ــ ما عدا الصيغة الأولى ــ مرجعا إلى كائن الارتباط Binding الذي تم إنشاؤه وإضافته إلى المجموعة.

وتستطيع استخدام الوسيلة Add أيضا لإضافة ارتباط بكائن بسيط حتى لو كان مجرد متغير نصي، مثل:

```
string Name = "Mohammad";
TextBox1.DataBindings.Add("Text", Name, "");
```

هذا الكود سيجعل مربع النص TextBox1 يعرض النص Mohammad.

كما يمكنك استخدامها لإضافة ارتباط بكائن مكون من أكثر من عنصر، مثل الفئات والسجلات.. مثال:

```
Size Sz = new Size(100, 200);
TextBox2.DataBindings.Add("Text", Sz, "Width");
```

هذا الكود سيجعل مربع النص TextBox2 يعرض الرقم ١٠٠.

ويمكنك استخدام الوسيلة Add لربط أكثر من مصدر بيانات بنفس الأداة.. هذا الكود صحيح:

```
TextBox1.DataBindings.Add("Text", Name, "");
TextBox1.DataBindings.Add("Tag", Sz, "Width");
```

لكن الوسيلة Add ستسبب خطأ في البرنامج لو حاولت ربط أكثر من مصدر بيانات بنفس عنصر العرض في نفس الأداة.. هذا الكود خاطئ:

```
TextBox1.DataBindings.Add("Text", Name, "");
TextBox1.DataBindings.Add("Text", Sz, "Width");
```

لهذا قبل أن تغير ارتباط الخاصية، عليك أن تزيل كائن الارتباط أولا من المجموعة باستخدام الوسيلة Remove كالتالي:

```
var Bnd = TextBox1.DataBindings.Add("Text", Name, "");
TxtName.DataBindings.Remove(Bnd);
TextBox1.DataBindings.Add("Text", Sz, "Width");
```

لاحظ كذلك أن عنصر البيانات يجب أن يكون خاصية وليس متغيرا.. بمعنى آخر: لا يمكنك ربط الأداة بحقل Field من حقول الكائن.. تذكر أن الحقل هو متغير عام Public Variable معرف على مستوى الفئة، مثل:

```
class Student
{
   public int ID;
   public int Age;
   public string Name;
}
```

الآن لو عرفت كائنا من فئة الطالب وليكن:

```
Student Std = new Student { ID = 1, Age = 15,
                  Name = "Ahmad" };
```

فإن محاولة ربط أي حقل خاص بالكائن Std (وليكن الحقل Name) بأية أداة ستؤدي إلى حدوث خطأ في البرنامج:

```
TxtId.DataBindings.Add("Text", Std, "Name");
```

لهذا عليك تغيير الحقول العامة في فئة الطالب لتصير خصائص.. كل ما عليك هو استخدام الخصائص ذاتية التعريف Auto Implemented Properties التي قدمتها سي شارب ٢٠٠٨ كالتالي:

```
class Student
{
        public int ID {get;set;}
        public int Age  {get;set;}
        public string Name { get; set; }
}
```

الآن لو جربت ربط أي خاصية من خصائص الكائن Std، فسيعمل كل شيء على ما يرام.. والمشروع BindingToObject يريك الكود الكامل لربط خصائص فئة الطالب بمربعات النصوص.. وعليك عند فحص هذا المشروع أن تلاحظ ما يلي:

١- أن أي تغيير تجريه على مربعات النصوص سيؤثر على الكائن الأصليStd .. ولو جربت تغيير بيانات الطالب في مربعات النصوص وضغط الزر "بيانات الطالب" فستعرض الرسالة القيم الموجودة في مربعات النصوص، رغم أن الكود المكتوب في حدث ضغط الزر يعرض بيانات المتغير Std.

٢- إذا حاولت أن تكتب قيمة خاطئة في أي مربع نص ككتابة حروف في مربع النص الخاص برقم الطالب، فسيتم إلغاؤها بمجرد مغادرته.. هذا معناه أن تقنية الربط تجيز البيانات تلقائيا قبل مغادرة الأداة، فإذا كانت ستسبب خطأ عند وضعها في عنصر البيانات يتم إلغاء القيمة من عنصر العرض وإعادته إلى قيمته السابقة.

كما يمكنك أيضا الارتباط بكائنات معقدة، مثل مصفوفة تحتوي على عناصر من نوع فئة التلميذ، أو كائنات أكثر تعقيدا مثل مجموعة البيانات التي تحتوي على جداول، بكل منها أعمدة يعتبر كل عمود منها مصفوفة (لأن به صفوفا) ويصلح كعنصر بيانات.. والمثال التالي ينشئ كائن ارتباط ويضيفه إلى مجموعة ارتباطات مربع النص، ليجعله يعرض اسم الكتاب الحالي في جدول الكتب:

```
Binding B = TextBox1.DataBindings.Add("Text",
            Ds, "Books.Book");
```

ويمكن فعل نفس الشيء أيضا بالكود التالي:

```
Binding B = TextBox1.DataBindings.Add("Text",
            Ds.Tables["Books"], "Book");
```

ويريك المشروع BindingTextBox مثالا طريفا على هذا، حيث سنجعل مربع نص يعرض اسم الكتاب الحالي، ومربع نص آخر يعرض اسم مؤلفه.. وسنعرض جدول الكتب كله في جدول عرض DataGridView الذي سنتعرف على طريقة ربطه لاحقا.

الجميل في الأمر أن المستخدم كلما انتقل من صف إلى آخر في جدول العرض، يعرض مربعا النص اسم الكتاب الموجود في هذا الصف واسم مؤلفه تلقائيا، وبدون أن نكتب أي كود!.. السبب في هذا أن جدول العرض يغير الصف الحالي في كائن الارتباط، فيقوم تلقائيا بتحديث القيم المعروضة في جميع الأدوات المرتبطة به!.. لكن لكي تعمل هذه الطريقة، يجب أن يكون مصدر البيانات المرتبط به جدول العرض هو نفسه مصدر بيانات مربعي النص.. هكذا مثلا:

```
DataGridView1.DataSource = Ds;
DataGridView1.DataMember = "Books";
TxtBook.DataBindings.Add("Text", Ds, "Books.Book");
TxtAuthor.DataBindings.Add("Text", Ds, "Books.Author");
```

أو يمكن استخدام جدول الكتب كمصدر بيانات للاختصار:

```
DataGridView1.DataSource = Ds.Books;
TxtBook.DataBindings.Add("Text", Ds.Books, "Book");
TxtAuthor.DataBindings.Add("Text", Ds.Books, "Author");
```

لكن الكود التالي لن يجعل البرنامج يعمل بشكل صحيح، لأن مصدر بيانات جدول العرض (وهو Ds.Books) مختلف عن مصدر بيانات مربعي النص (وهو Ds):

```
DataGridView1.DataSource = Ds.Books;
TxtBook.DataBindings.Add("Text", Ds, "Books.Book");
TxtAuthor.DataBindings.Add("Text", Ds, "Books.Author");
```

كذلك فإن الكود التالي أيضا لن يجعل البرنامج يعمل بشكل صحيح، لأن مصدر بيانات جدول العرض (وهو Ds) مختلف عن مصدر بيانات مربعي النص (وهو Ds.Books):

```
DataGridView1.DataSource = Ds;
DataGridView1.DataMember = "Books";
TxtBook.DataBindings.Add("Text", Ds.Books, "Book");
TxtAuthor.DataBindings.Add("Text", Ds.Books, "Author");
```

وتمتلك مجموعة الارتباطات الخصائص الجديدة التالية:

الأداة Control:

تعيد الأداة التي تنتمي إليها مجموعة الارتباطات الحاليّة.

المكون القابل للارتباط BindableComponent:

تعيد واجهة المكون القابل للارتباط IBindableComponent التي تنتمي إليها مجموعة الارتباطات الحالية.

الطريقة الافتراضية لتحديث مصدر البيانات DefaultDataSourceUpdateMode:

تحدد القيمة الافتراضية للخاصية DataSourceUpdateMode لكل كائن ربط في المجموعة، وهي تأخذ إحدى قيم المرقم DataSourceUpdateMode التي سنتعرف عليها لاحقا.

فئة الارتباط Binding Class

تقوم هذه الفئة بربط كائن يعمل كمصدر بيانات، بإحدى الأدوات، بحيث يأخذ عنصر العرض في الأداة قيمة عنصر البيانات في الكائن تلقائيا، وتغير قيمة أحدهما كلما تغيرت قيمة الآخر.

ولحدث إنشاء هذه الفئة نفس صيغ الوسيلة ControlBindingsCollection.Add، ما عدا الصيغة الأولى التي تستقبل كائن ارتباط Binding.

افترض أن لدينا مصفوفة أعداد صحيحة معرفة على مستوى النموذج كالتالي:

```
int[] A = { 1, 2, 3, 4 };
```

سنعرّف الآن كائنا يربط الخاصيّة Text في مربّع النصّ بعناصر هذه المصفوفة:

```
var B = new Binding("Text", A, "");
TextBox1.DataBindings.Add(B);
```

لاحظ أن حدث الإنشاء استقبل ثلاثة معاملات:

- اسم خاصية الأداة وهي هنا "Text".
- مجموعة البيانات وهي هنا المصفوفة A.
- اسم عنصر العرض وقد تركناه فارغا لربط الكائن نفسه (المصفوفة).. لكن لو كنا نتعامل مع سجل الطالب Student Structure مثلا (كما فعلنا في المشروع BindingToArray)، فيمكن أن نرسل إلى هذا المعامل اسم أي حقل من حقوله، كالاسم "Name" أو العمر "Age".. أو غير ذلك.

وتمنحك الفئة Binding الخصائص التالية:

المكون القابل للارتباط BindableComponent:

تعيد واجهة المكون القابل للارتباط IBindableComponent التي تمثلها الأداة التي ينتمي إليها الارتباط الحالي.

الأداة Control:

تعيد الأداة Control التي ينتمي إليها الارتباط الحاليّ.

اسم الخاصية PropertyName:

تقرأ أو تغير اسم عنصر العرض.

مصدر البيانات DataSource:

تعيد الكائن Object الذي يعمل كمصدر للبيانات في هذا الارتباط.

معلومات عنصر الربط BindingMemberInfo:

تعيد نسخة من سجل معلومات عنصر الربط BindingMemberInfo Structure، تحتوي على معلومات حول عنصر البيانات.. وسنتعرف على هذا السجل لاحقا.

أساس مدير الربط BindingManagerBase:

تعيد كائن أساس الارتباط BindingManagerBase، الذي يتيح لك التحكم في الارتباط الحاليّ.. وسنتعرف على الفئة BindingManagerBase بالتفصيل لاحقا.

هل هو مرتبط IsBinding:

تعيد True إذا كان الارتباط فعالا، وتعيد False إذا تم إيقاف الارتباط عن العمل مؤقتا.

طريقة تحديث الأداة ControlUpdateMode:

تحدد كيفية تحديث عنصر العرض، عندما تتغير قيمة عنصر البيانات.. وتأخذ هذه الخاصية إحدى قيمتي المرقم ControlUpdateMode التاليتين:

Never	لا يتم تحديث قيمة عنصر العرض عند تغير قيمة عنصر البيانات.
OnPropertyChanged	يتم تحديث قيمة عنصر العرض فور تغير قيمة عنصر البيانات.. هذه هي القيمة الافتراضية.

القيمة الفارغة NullValue:

تحدد القيمة التي ستوضع في عنصر العرض، إذا كانت لعنصر البيانات القيمة Nothing.. لاحظ أن هذه الخاصية ستكون بلا فائدة إذا كانت للخاصية ControlUpdateMode القيمة None.

طريقة تحديث مصدر البيانات DataSourceUpdateMode:

تحدد كيفية تحديث قيمة عنصر البيانات عند تغير قيمة عنصر العرض، وهي تأخذ إحدى قيم المرقم DataSourceUpdateMode التالية:

Never	لا يتم تحديث عنصر البيانات.. هذا يعني أن أي تغيير يحدث في عنصر العرض (كأن يكتب المستخدم في مربع النص مثلا) لن يؤثر على قيمة عنصر البيانات.
OnProperty Changed	يتم تغيير قيمة عنصر البيانات بمجرد تغيير قيمة عنصر العرض.. فمثلا لو كتب المستخدم في مربع النص، فإن ما كتبه يوضع فورا في عنصر البيانات.
OnValidation	لا يتم نقل التغيير من عنصر العرض إلى عنصر البيانات إلا عند انطلاق حدث تمام التحقق من الصحة Control.Validated، وذلك عند مغادرة الأداة أو عرض عنصر آخر من عناصر الكائن (إذا كان الكائن يحتوي على مجموعة من العناصر مثل المصفوفات).. هذه هي القيمة الافتراضية.

القيمة الفارغة لمصدر البيانات DataSourceNullValue:

تحدد القيمة التي ستوضع في عنصر البيانات، إذا كانت لعنصر العرض القيمة Nothing.. لاحظ أن هذه الخاصية ستكون بلا فائدة إذا كانت للخاصية DataSourceUpdateMode القيمة None.

نص التنسيق FormatString:

تستقبل نصا يعرف صيغة التنسيق الذي سيستخدم لتنسيق البيانات قبل وضعها في عنصر العرض.. ويمكنك استخدام نفس صيغ تنسيق الأرقام والتواريخ التي تعرفنا عليها في ملاحق كتاب برمجة إطار العمل.

تفعيل التنسيق FormattingEnabled:

إذا جعلت قيمة هذه الخاصية True فسيتم تطبيق التنسيق الموضح في الخاصية FormatString على البيانات قبل وضعها في عنصر العرض.

معلومات التنسيق FormatInfo:

تقرأ أو تغير واجهة مزود التنسيق IFormatProvider التي يستخدمها كائن الارتباط.. لقد عرفنا في كتاب برمجة إطار العمل أن الفئات NumberFormatInfo، DateTimeFormatInfo، CultureInfo تمثل واجهة مزود التنسيق IFormatProvider، لكن بالنسبة لهذه الخاصية، يمكنك استخدام نسخة من الفئة CultureInfo للتحكم في اللغة والثقافة التي ستستخدم عند تنسيق البيانات.. وفي الوضع الافتراضي تتعامل هذه الخاصية مع الثقافة (اللغة) المعرفة على جهاز المستخدم.

ويمتلك كائن الربط الوسيلتين التاليتين:

قراءة القيمة ReadValue:

تجبر الأداة على عرض قيمة عنصر البيانات.

كتابة القيمة WriteValue:

تجبر الأداة على وضع قيمة عنصر العرض، في عنصر البيانات.

ويمنحك كائن الارتباط الأحداث التالية:

تنسيق Format:

ينطلق قبل كتابة قيمة عنصر البيانات في عنصر العرض، ليسمح لك بتنسيق البيانات قبل أن تعرضها الأداة.. والمعامل e لهذا الحدث من النوع ConvertEventArgs، وهو يمتلك الخاصيتين التاليتين:

	DesiredType	تعيد كائن النوع Type، الذي يمثل نوع بيانات عنصر العرض.
	Value	تقرأ أو تغيير البيانات التي سيتمّ وضعها في عنصر العرض.. هذه الخاصية من النوع Object للتعامل مع أنواع البيانات المختلفة.

مثلا، يمكنك أن تعرض البيانات في الأداة بتنسيق التاريخ القصير كالتالي:

```
e.Value = Convert.ToDateTime(e.Value).ToString("d/MM/yy");
```

تحويل Parse:

ينطلق عندما تتغيّر قيمة عنصر العرض.. فإذا كنت قد غيرت تنسيق البيانات باستخدام الحدث Format، فاستخدم هذا الحدث لاستخلاص البيانات الأصليّة وإعادتها إلى النوع المناسب لوضعها في خاصية الكائن.

والمعامل e لهذا الحدث مماثل لذلك الخاصّ بالحدث السابق.. انظر كيف نستعيد التاريخ من النصّ الذي نسقناه في الحدث السابق:

```
e.Value = Date.Parse(e.Value.ToString( ));
```

انتهى الربط BindingComplete:

ينطلق هذا الحدث بعد وضع قيمة عنصر البيانات في عنصر العرض، أو بعد وضع قيمة عنصر العرض في عنصر البيانات.. والمعامل الثاني e لهذا الحدث من النوع BindingCompleteEventArgs، وهو يمتلك الخصائص التالية:

Binding	تعيد كائن الارتباط Binding الذي أطلق الحدث.
Binding Complete Context	تخبرك باتجاه عملية الربط، وهي تعيد إحدى قيمتي المرقم BindingCompleteContext التاليتين: - ControlUpdate: يتم تحديث الأداة. - DataSourceUpdate: يتم تحديث مصدر البيانات.
Binding Complete State	توضح حالة عملية الربط، وهي تعيد إحدى قيم المرقم BindingCompleteState التالية: - Success: نجحت عملية الربط. - DataError: فشلت عملية الربط بسبب خطأ في البيانات.. في هذه الحالة يرفض مصدر البيانات أو الأداة القيمة الجديدة، لكن لا يحدث خطأ في البرنامج. - Exception: فشلت عملية الربط وحدث خطأ في البرنامج.
ErrorText	تعيد نصا يصف الخطأ الذي حدث في عملية الربط.
Exception	تعيد كائن الاستثناء Exception الذي يحتوي على تفاصيل الخطأ الذي حدث عند ربط البيانات.

والمثال التالي يعرف كائن ربط Binding Object بين الخاصيّة Text لمربّع نصّ، وبين العمود Book في جدول الكتب في مجموعة البيانات Ds:

```
var B = new Binding("Text", Ds, "Books.Book");
```

هذا الارتباط لن يأخذ حيّزا من التنفيذ إلا إذا أضفناه إلى مجموعة الارتباطات DataBindings الخاصّة بمربّع النصّ، كالتالي:

```
TextBox1.DataBindings.Add (B);
```

وسنرى لاحقا كيف نتحرك بأنفسنا عبر صفوف جدول الكتب المختلفة، لنعرض أسماء الكتب المختلفة في مربع النص.. وعلى كل حال، ستجد هذا مطبقا في المشروع ViewAndEditBooks.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

سجل معلومات عنصر الربط
BindingMemberInfo Structure

يحتوي هذا السجل على معلومات عن عنصر البيانات.. ولا تتضح فائدة هذا السجل عند الارتباط بكائنات بسيطة، وإنما تتضح عند الارتباط مع كائنات تحتوي على كائنات متداخلة.. في هذه الحالة عليك إرسال مسار الكامل لاسم الخاصية التي تعمل كعنصر بيانات، إلى حدث إنشاء هذا الكائن.. مثال:

```
var Bmi = BindingMemberInfo ("Books.Book");
```

حيث Books هو اسم جدول الكتب، و Book هو الحقل الذي سيتم عرض قيمته في الأداة.. لاحظ أن هذا المسار لا يحتوي على اسم الكائن (وهو مجموعة البيانات Ds في مثالنا هذا)، فهذه المعلومة موجودة في كائن الربط Binding الذي ينتمي إليه السجل BindingMemberInfo.. ويمكنك أيضا الحصول على نسخة من هذا السجل باستخدام الخاصية Binding.BindingMemberInfo.

ويمتلك هذا السجل الخصائص التالية:

عنصر الربط BindingMember:

تعيد المسار الكامل لعنصر البيانات... وفي المثال الذي ضربناه ستعيد هذه الخاصية النص: Books.Book .

حقل الربط BindingField:

تعيد اسم عنصر البيانات (بدون المسار الكامل).. هذا يعني أنها ستعيد النص Book في المثال السابق.

مسار الربط BindingPath:

تعيد مسار عنصر البيانات (بدون اسم الخاصية).. ولو استخدمنا هذه الخاصية مع المثال السابق، فستعيد النص Books.

ولو كان الكائن بسيطا وعنصر البيانات ليس له مسار، ففي هذه الحالة ستكون القيمة العائدة هي اسم الخاصية، مثل Width لو كان الارتباط بكائن الحجم Size.

فئة محتوى الربط BindingContext Class

هذه الفئة تمثل واجهة المجموعة ICollection، وهي تعمل كمجموعة للقراءة فقط تحتوي على كائنات مدير الربط BindingManagerBase الخاصة بأداة معينة.. وسنتعرف على الفئة BindingManagerBase بعد قليل.

ويمكنك الحصول على نسخة من هذه الفئة باستخدام الخاصية Control.BindingContext.. والمثال التالي يعيد إليك مجموعة تحتوي على كائنات أساس مدير الربط للنموذج:

```
BindingContext BC = this.BindingContext;
```

لاحظ أن مجموعة محتوى الربط BindingContext الخاصة بالأداة الحاوية تحتوي على كل كائنات مدير الربط BindingManagerBase الخاصة بكل الأدوات الموجودة على هذه الأداة الحاوية.. هذا يعني أن محتوى الربط BC في المثال السابق سيحتوي على كائنات أساس الربط لكل الأدوات الموضوعة على النموذج (بشرط أن تكون داخلة في ارتباطات).. هذا يفيدك في تسهيل كتابة الكود عندما توجد العديد من الأدوات المرتبطة على النموذج.

وتمتلك هذه المجموعة الخاصية التالية:

العنصر Item:

هذه هي الخاصيّة الافتراضيّة، وهي تعيد مدير الربط BindingManagerBase الموجود في المجموعة تبعا للمعاملات المرسلة.. ولهذه الخاصية صيغتان:

١. الصيغة الأولى لها معامل من النوع Object، يستقبل مصدر البيانات الذي تريد الحصول على مدير الربط الخاص به.. والكود التالي يوقف الارتباطات بين مجموعة البيانات Ds وكل الأدوات الموجودة على النموذج:

```
this.BindingContext[Ds].SuspendBinding( );
```

٢. الصيغة الثانية تزيد على الصيغة السابقة بمعامل نصي، يستقبل مسار عنصر البيانات.. هذا مفيد إذا كان الكائن يحتوي على العديد من مصادر البيانات وتريد التعامل مع واحد منها فقط.. وفي المشروع BindingToDataSet استخدمنا الجملة التالية للحصول على مدير الربط:

var Bm = this.BindingContext[DsBooks, "Books"];

لاحظ أن مدير الربط Bm في هذه الحالة يتعامل مع جدول الكتب، لهذا نستطيع استخدام الخاصية Bm.Position للتحكم في الكتاب المعروض حاليا في الأدوات.. بينما لو استخدمت الجملة التالية:

var Bm = this.BindingContext[DsBooks];

فستحصل على مدير ربط يتعامل مع مجموعة البيانات نفسها، ونظرا لأنها تحتوي على العديد من القوائم الداخلية (مثل مجموعة الجداول ومجموعة العلاقات وغيرهما)، فلن يستطيع مدير الربط Bm في هذه الحالة التعامل مع جدول الكتب، وستشير الخاصية Bm.Count إلى أن هناك عنصرا واحدا فقط في مدير الربط (وهو مجموعة البيانات نفسها)، ولهذا لن تستطيع الانتقال إلى سجلات أخرى باستخدام الخاصية Bm.Position.

كما تمتلك هذه المجموعة الوسيلتين التاليتين:

تحتوي على Contains:

تعيد True إذا كانت المجموعة تحتوي على مدير الربط المحدد في المعاملات.. ولهذه الوسيلة نفس صيغتي الخاصية الافتراضية Item.

تحديث الربط UpdateBinding:

أرسل إلى هذه الوسيلة مجموعة محتوى الربط BindingContext، وكائن الربط Binding الذي تريد إزالته من مجموعة أخرى وإضافته إلى المجموعة المحددة في المعامل الأول.

فئة أساس مدير الربط BindingManagerBase Class

هذه الفئة أساسية مجردة Abstract Base Class، ومنها تشتق الفئات التي تعمل كمدير للربط.. والفئات التاليتان ترثان هذه الفئة:

١. فئة مدير التسلسل CurrencyManager Class.

٢. فئة مدير الخاصية PropertyManager Class.

ولتعريف متغير من هذا الكائن، استخدم الصيغة التالية:

```
BindingManagerBase BM;
```

ولوضع نسخة جديدة من مدير الربط في هذا المتغير، يمكنك استخدام الخاصية BindingManagerBase من كائن الارتباط Binding.. مثال:

```
var Bnd = TextBox1.DataBindings.Add("Text", Obj, "");
BM = Bnd.BindingManagerBase;
```

لاحظ أن نوع المدير الذي سيوضع في المتغير BM يتوقف على نوع الكائن.. فلو كان كائنا بسيطا فسيحتوي المتغير BM على نسخة من مدير الخاصية PropertyManager الذي يتحكم بالارتباط بعنصر البيانات.. أما إذا كان الكائن مركبا ويحتوي على قائمة من العناصر، فسيحتوي المتغير BM على نسخة من مدير التسلسل CurrencyManager الذي يتحكم في ربط قائمة عناصر الكائن.

وتمتلك هذه الفئة الخصائص التالية:

الارتباطات Bindings:

تعيد مجموعة الارتباطات BindingsCollection التي تحتوي على جميع الارتباطات التي تشترك فيها الأداة الحالية.

العدد Count:

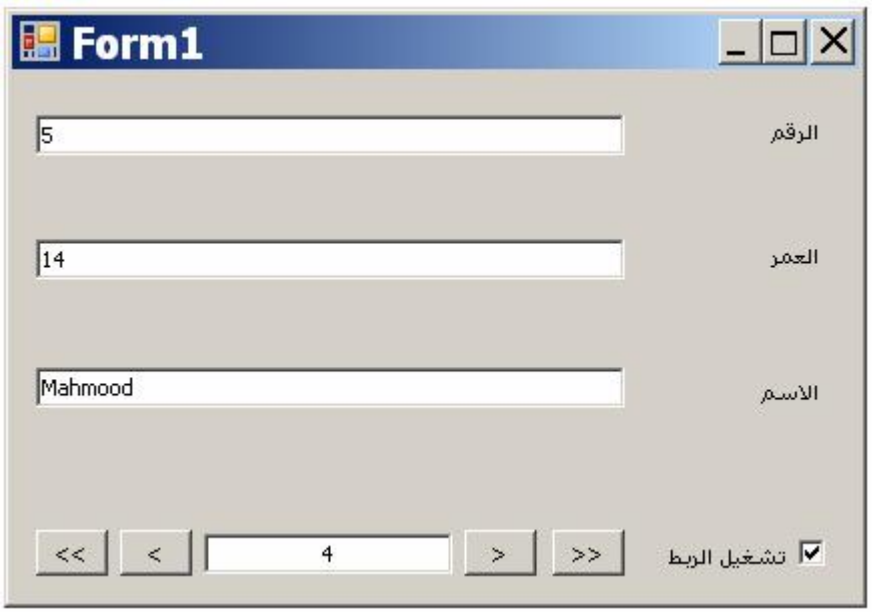
تعيد عدد العناصر المشتركة في الارتباط.. هذا العدد سيكون دائما ١ إذا كان الكائن بسيطا وتم الارتباط بإحدى خصائصه.. أما إذا كان الكائن معقدا وتم الارتباط بقائمة عناصر موجودة داخله، فإن هذه الخاصية تعيد عدد عناصر هذه القائمة (مثل عدد سجلات الجدول إذا كان الارتباط بعمود في أحد جداول مجموعة البيانات).

الموضع Position:

تقرأ أو تغيّر موضع العنصر الذي تعرضه الأداة حاليّا.. هذا مفيد عند الارتباط بكائن معقد يحتوي على قائمة من العناصر، ففي بدء الارتباط ستوضع في عنصر العرض قيمة أول عنصر في هذه القائمة، ويمكنك بعد هذا أن تستخدم الخاصية Position لعرض أي عنصر آخر في القائمة.. والمثال التالي يعرض في مربع النص ثالث كتاب في جدول الكتب:

```
Binding Bnd = new Binding("Text", Ds, "Books.Book");
TxtBook.DataBindings.Add(Bnd);
Bnd.BindingManagerBase.Position = 2;
```

وقد استخدمنا هذه الخاصية في التطبيق BindindSample2 لنتيح للمستخدم التحرك عبر خانات مصفوفة التلاميذ وعرض بيانات كل تلميذ في مربعات النصوص، وذلك باستخدام أزرار الاتجاهات أسفل النموذج:



المريح في الأمر أننا لا نحتاج إلى تغيير الموضع لكل مربع نص على النموذج، فكل ما علينا هو الحصول على مدير الربط الخاص بمصفوفة التلاميذ من خلال محتوى الربط الخاص بالنموذج كالتالي:

```
BindingManagerBase Bm = this.BindingContext[Std];
```

وبهذا يؤدي تغيير قيمة الخاصية Bm.Position إلى تغيير العنصر الحالي المعروض في جميع أدوات النموذج.

كما يتيح البرنامج BindindSample2 للمستخدم كتابة رقم الخانة مباشرة في مربع النص الذي يتوسط الأزرار (واسمه TxtPos)، وعندما يضغط Enter من لوحة المفاتيح يتم عرض التلميذ الموجود في هذه الخانة.. لاحظ أن الخاصية Position ترفض أي موضع غير صحيح دون أن يحدث خطا في البرنامج.. لهذا لو جربت أن تكتب الرقم ١٠ مثلا في مربع النص وتضغط Enter، فإن الخاصية Position ستنتقل تلقائيا إلى آخر خانة مسموح بها وهي الخانة رقم ٤ في هذا المثال.

الحالي Current:

تعيد الكائن Object المرتبط حاليا بعنصر العرض.. وفي حالة الارتباط بكائن بسيط يحتوي على عدة خصائص (مثل كائن الحجم Size)، تعيد هذه الخاصية هذا الكائن، أما عند الارتباط بكائن معقد يحتوي على قائمة من العناصر، فإن هذه الخاصية تعيد العنصر الحالي في القائمة (الموجود في الموضع الذي تحدده الخاصية Position).

هل الربط متوقف IsBindingSuspended:

تعيد True إذا كان الربط بين الأداة ومصدر البيانات متوقفا حاليا.

وتمتلك هذه الفئة الوسائل التالية:

إضافة جديد AddNew:

تضيف عنصرا جديد إلى قائمة عناصر مصدر البيانات.. وتسبب هذه الوسيلة خطأ في البرنامج إذا كانت الأداة مرتبطة بكائن بسيط لا يحتوي على قائمة داخلية، أو إذا كان الكائن مصفوفة.. وعند استخدام هذه الوسيلة مع صفوف البيانات، تكون حالة السجل الجديد هي RowState.Added، ولن تتم إضافته إلى قاعدة البيانات إلا عند تحديثها.

حذف من موضع RemoveAt:

تحذف العنصر الموجود في الموضع المرسل كمعامل.. وتسبب هذه الوسيلة خطأ في البرنامج إذا كانت الأداة مرتبطة بكائن بسيط لا يحتوي على قائمة داخلية، أو كان الكائن مصفوفة أو كان لا يمثل الواجهة IBindingList.. وعند استخدام هذه الوسيلة للتعامل لحذف صف بيانات، فسيتم تغير حالة حالته إلى Deleted، ولن يتم حذفه من قاعدة البيانات إلا عند تحديثها.

إلغاء التحرير الحالي CancelCurrentEdit:

تُلغي عمليّة التحرير الحاليّة وتعيد إلى السجل قيمه الأصلية.. وليس لهذه الوسيلة أي تأثير إلا على الكائنات التي تمثل الواجهة IEditableObject مثل فئة سجلّ العرض DataRowView Class.

إنهاء التحرير الحالي EndCurrentEdit:

تُنهي عمليّة التحرير مع إبقاء التغييرات التي حدثت للسجل.. وليس لهذه الوسيلة أي تأثير إلا على الكائنات التي تمثل الواجهة IEditableObject مثل فئة سجلّ العرض DataRowView Class.

معرفة خصائص العنصر GetItemProperties:

تعيد مجموعة واصفات الخصائص PropertyDescriptorCollection التي تصف خصائص المجموعة المشتركة في الارتباط.. مثلا: عند الارتباط بحقل اسم الكتاب في جدول الكتب، ستحتوي هذه المجموعة على واصف الخصائص PropertyDescriptor لكل عمود في جدول الكتب.. والمثال التالي يعرض اسم أول حقل في السجل الحالي، ويعرض قيمته:

```
var BM = Bnd.BindingManagerBase;
var PD = BM.GetItemProperties( )[0];
MessageBox.Show(PD.Name); // ID
MessageBox.Show(PD.GetValue(BM.Current).ToString ( ));
```

حيث ستعرض الرسالة الأولى اسم الحقل ID بينما ستعرض الرسالة الثانية قيمة الحقل ID في السجل الحالي.

إيقاف الارتباط SuspendBinding:

توقف الارتباط بين الأدوات ومصدر البيانات مؤقتا.. وقد استخدمنا هذه الوسيلة لإيقاف الربط في التطبيق BindingToArray، وذلك عند إزالة علامة الاختيار من مربع الاختيار CheckBox الموجود أسفل النموذج.

مواصلة الارتباط ResumeBinding:

تستأنف الارتباط بين الأدوات ومصدر البيانات.

وتمتلك هذه الفئة الأحداث التالية:

الربط اكتمل BindingComplete:

مماثل للحدث Binding.BindingComplete.

الموضع تغير PositionChanged:

ينطلق إذا تغيّرت قيمة الخاصية Position.. والمعامل الثاني e لهذا الحدث من النوع EventArgs، الذي لا يحمل أية معلومات هامة عن الحدث.

وقد استخدمنا هذا الحدث في التطبيق BindingToArray، لتحديث الموضع المعروض في مربع النص TxtPos الموجود أسفل النموذج، كلما تغير الموضع الحالي بسبب ضغط أزرار التحرك.. لاحظ أننا ربطنا هذا الحدث بالإجراء المستجيب له باستخدام الجملة AddHandler في حدث تحميل النموذج كالتالي:

```
Bm.PositionChanged += Bm_PositionChanged;
```

السجل الحالي تغير CurrentChanged:

ينطلق إذا تغيّرت قيمة الخاصية Current.

العنصر الحالي تغير CurrentItemChanged:

ينطلق إذا تغيرت حالة العنصر الحالي الذي تعرضه الأداة.. يحدث هذا إذا تغيرت قيمة إحدى خصائص هذا العنصر، أو إذا تم استبداله أو حذفه.

خطأ البيانات DataError:

ينطلق إذا قام مدير الربط بمعالجة خطأ حدث أثناء عملية الربط.. والمعامل الثاني e لهذا الحدث من النوع BindingManagerDataErrorEventArgs، وهو يمتلك الخاصية Exception التي تعيد كائن الاستثناء Exception الذي يحتوي معلومات عن الخطأ الذي حدث.

فئة مدير الخاصية PropertyManager Class

هذه الفئة ترث الفئة BindingManagerBase، وهي تعمل كمدير يتحكم في ربـــط كـــائن بسيط له عدة خصائص ولا يحتوي على قائمة عناصر داخلية.
ولا تمتلك هذه الفئة أية خصائص أو وسائل أو أحداث جديدة غير ما ترثه من الفئة الأم.

فئة مدير التسلسل CurrencyManager Class

هذه الفئة ترث الفئة BindingManagerBase، وهي تعمل كمدير يتحكم في ربـــط كـــائن مركب يحتوي على قائمة عناصر داخلية.
وإضافة إلى ما ترثه من الفئة الأم، تمتلك هذه الفئة العنصرين التاليين:

القائمة List:

هذه الخاصية من نوع واجهة القائمة IList، هي تعيد القائمة الداخلية التـــي يحتويهـــا الكائن.

إنعاش Refresh:

تعيد ملء مصفوفة العناصر المرتبطة.. استخدم هذه الوسيلة عند الارتباط بكائنـــات لا تعطي تنبيها عند تغير عناصرها، مثل المصفوفات Arrays.

ربط الأدوات في وقت التصميم:

يقدم لك مصمم النماذج Form Designer في دوت نت تسهيلات كثيرة لربط الأدوات في وقت التصميم، لتقليل الكود الذي تحتاجه لأداء هذه العملية.. ولكي ترى هذا عمليا، ابدأ مشروعا جديدا وأسمه ViewAndEditBooks، وصمّم واجهة استخدامه كما في الصورة التالية:

أضف إلى النموذج مهيئ بيانات اسمه DaAuthorBooks، واجعله يستخدم الاستعلام التالي للحصول على أسماء المؤلفين وأسماء كتبهم:

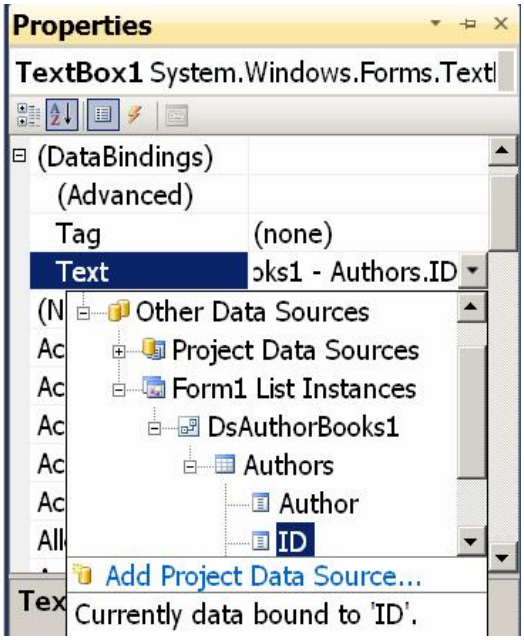
```
SELECT Authors.Author, Books.ID, Books.Book
FROM   Authors INNER JOIN
Books ON Authors.ID = Books.AuthorID
```

أنشئ مجموعة بيانات محددة النوع Typed DataSet من هذا المهيئ بالطريقة المعهودة، وأسمها DsAuthorBooks، وأضف نسخة منها إلى النموذج اسمها DsAuthorBooks1.

الآن، نريد ربط مربّعات النصوص بهذه المجموعة.. اتبع الخطوات التالية:

- حدد مربّع النصّ الذي سيعرض رقم الكتاب، ومن نافذة الخصائص حدّد المقطع المسمّى DataBinding.. ستجده في بداية الخصائص، خروجا عن الترتيب الأبجديّ للخصائص، وسيكون موضوعا بين قوسين.

- اضغط العلامة + المجاورة للمقطع DataBinding، لإسدال خصائص الأداة التي يمكن ربطها بمجموعة البيانات.. بالنسبة لمربّع النصّ، ستجد أنّ بإمكانك ربط الخاصيتين Tag و Text.. في الغالب يتمّ ربط الخاصيّة Tag برقم السجلّ ID، وذلك لتسهيل التعامل مع الحقل عند الحاجة.. ولكن في مثالنا هذا، سنعرض هذا الرقم في الخاصيّة Text لتظهر للمستخدم.
- حدّد الخاصيّة Text، واضغط زرّ الإسدال الموجود في خانة القيمة.. ستعرض لك القائمة المنسدلة اختيارين:



١- None: لإلغاء ربط الخاصية بأي مصدر بيانات.

٢- Other Data Sources: ولو أسدلت عناصر هذا الفرع، فستجد تحته عنصرين فرعيين:

أ. Project Data Sources: ستجد تحت هذا العنصر مصادر البيانات العامة للمشروع، مثل فئات مجموعات البيانات محددة النوع مثل DsAuthorBooks.. ولو اخترت أيا من هذه الفئات، فسيتم تعريف نسخة منها لاستخدامها في النموذج الحالي.

ب. Form1 List Instances: ستجد تحت هذا الفرع نسخ الأدوات الموضوعة على النموذج الحالي، والتي تصلح للعمل كقوائم ومصادر بيانات.. وفي مشروعنا هذا، ستجد تحته نسخة مجموعة البيانات DsAuthorBooks1.. لو أسدلت عناصر هذه المجموعة، فستجد تحتها أسماء الجداول المعرفة في مجموعة البيانات محددة النوع.. وستجد في مثالنا هذا جدولا واحدا اسمه Authos، وذلك لأن مهيئ البيانات قد منح الجدول الناتج من استعلام الربط

Join Query الاسم الافتراضي Authors.. أسدل حقول هذا الجدول، واختر الحقل ID.

بهذا نكون قد ربطنا الخاصية Text لمربع النص بالحقل ID في مجموعة البيانات.. وبنفس الطريقة يمكنك ربط الخاصية Text لمربع النص الثاني بالحقل Book، ومربع النص الثالث بالحقل Author.

الآن أنهينا ربط أدواتنا بمصدر البيانات، بحيث لو ملأنا مجموعة البيانات بالسجلات، فستظهر قيم حقول السجلّ الحاليّ في مربّعات النصّ، بدون أن نكتب أي كود لفعل هذا.. وكلّما تحرّكنا من سجلّ إلى آخر، يتمّ عرض قيم حقول السجلّ الجديد في الأدوات آليّا.

عند هذه النقطة، لو شغّلت المشروع وضغطت زرّ تحميل البيانات، فستظهر قيم السجلّ الأوّل في مربّعات النصّ، كلّ حقل في مربّعه الذي حدّدناه.

نريد الآن أن نكتب كود الأزرار التي تسمح للمستخدم بالتنقل بين سجلات مجموعة البيانات.. سيكون الأمر بسيطا، فكل ما علينا هو استخدام مدير الربط BindingManagerBase، والذي يمكننا الحصول عليه من خلال الخاصية BindingContext الخاصة بالنموذج كالتالي:

```
BindingManagerBase Bm =
    this.BindingContext[DsAuthorBooks1, "Authors"];
```

الآن تستطيع تغيير الموضع كما تريد، باستخدام الخاصيتين Position و Count التابعتين لمدير الربط.. مثلا، في زر الانتقال إلى السجل التالي، استخدمنا الكود:

```
if (Bm.Position < Bm.Count - 1)
{
  Bm.Position += 1;
  LbPosition.Text = (Bm.Position + 1).ToString( ) + " /  " +
                    Bm.Count.ToString( );
}
```

لاحظ أن محاولة تغيير السجل الحالي قد تؤدي إلى حدوث خطأ في البرنامج، وذلك لأن مدير الربط سيفحص مربعات النصوص، فإن كانت بعض قيمها تغيرت، فسيحاول حفظها في مجموعة البيانات، وسيحدث خطأ إذا كان المستخدم قد أدخل قيمة غير مناسبة لأحد الحقول.. لحل هذه المشكلة، استخدم المقطع Try Catch لمعالجة أي خطأ من هذا النوع،

وفي المقطع Catch استخدم الجملة التالية لإلغاء تحرير السجل الحالي (الذي سبب المشكلة):

Bm.CancelCurrentEdit( );

لاحظ أن هذا الكود سيعيد قيم كل مربعات النصوص إلى ما كانت عليه.. سيكون هذا مستفزا للمستخدم للغاية لو كان عدد مربعات النصوص كبيرا وكان الخطأ ناتجا عن قيمة خاطئة في واحد منها فقط.. لهذا سيكون من الأذكى أن تلغي تحرير مربع النص الذي سبب المشكلة، أو أن تترك القيم الحالية كما هي، وتترك للمستخدم معرفة مربع النص الذي سبب المشكلة من خلال رسالة الخطأ.

وألفت نظرك مجددا إلى أن كل التغييرات التي يجريها المستخدم على مربعات النصوص يتم حفظها في مجموعة البيانات (وليس في قاعدة البيانات)، لهذا على المستخدم ضغط زر الحفظ لإرسال التغييرات من مجموعة البيانات إلى قاعدة البيانات.. هذا الزر يستخدم أمر التحديث Update الخاص بمهيئ البيانات، لكن هذا يحتاج إلى بعض العمل منا، لأن مهيئ البيانات لا ينتج أمر التحديث إذا كان أمر التحديد يعيد حقولا من أكثر من جدول، تاركا لك أنت التحكم في الحقول التي تريد تحديثها وكيفية تحديثها.. ونظرا لأننا سنسمح في هذا البرنامج بتحديث الحقل Books.Book فقط، فسنستخدم أمر التحديث التالي:

```
UPDATE Books
SET Book = @Book
WHERE ID = @Original_ID
```

وستجد تعريف هذا الأمر ومعاملاته في حدث تحميل النموذج.

ربط مربعات القوائم Binding List Boxs:

رأينا حتى الآن كيف نربط الأدوات البسيطة كمربعات النصوص بمصادر البيانات المختلفة.. لكن ماذا لو أردنا ربط أدوات أكثر تعقيدا مثل القائمة ListBox والقائمة المركبة ComboBox وقائمة الاختيار CkeckedListBox؟

لو حاولت استخدام كائن الربط لربط مصدر البيانات بالخاصية Items لهذه الأدوات، فكل ما ستحصل عليه هو رسالة خطا، تخبرك أنه لا يمكن الارتباط بالخاصية Items لأنها للقراءة فقط!

إذن فما الحل؟

في الحقيقة، هناك طريقة أخرى لربط مربعات القوائم بمصادر البيانات، فهذه الأدوات لا تحتاج إلى المرور من سجل إلى آخر، فهي قادرة على عرض كل السجلات دفعة واحدة، ومن أجل هذا فهي تمتلك خصائص مجهزة لهذا الغرض، وهي:

مصدر البيانات DataSource:

ضع في هذه الخاصية الكائن الذي تريد الارتباط به.

عنصر العرض DisplayMember:

تستقبل نصا، يحدد اسم خاصية الكائن التي سيتم عرض قيمته في القائمة.. وفي المشروع BindingListToArray جعلنا قيمة هذه الخاصية "Name"، لهذا تعرض القائمة أسماء الطلاب.

لاحظ أنك لو تركت هذه الخاصية فارغة، فستعرض القائمة النص الذي تعيده الوسيلة ToString الخاصة بكل عنصر من عناصر مصدر البيانات.

القيمة المحددة SelectedValue:

تعيد القيمة المحددة حاليا في القائمة، وهي تتوقف على قيمة الخاصية ValueMember.

عنصر القيمة ValueMember:

تستقبل نصا، يحدد اسم خاصية الكائن التي ستتم قراءتها عند استخدام الخاصية SelectedValue.. وفي المشروع BindingListToArray جعلنا قيمة هذه الخاصية "Id"، لهذا فإن الخاصية SelectedValue تعيد رقم الطالب المحدد حاليا في القائمة، ويمكنك تجربة هذا بضغط الزر الموجود أسفل القائمة.

ولو تركت الخاصية ValueMember فارغة، فإن الخاصية SelectedValue ستعيد العنصر المحدد في القائمة حاليا مثلها مثل الخاصية SelectedItem.

تعال نستخدم هذه الخصائص في تطوير المشروع ViewAndEditBooks، فهو يبدو عقيما لو حاولت استخدامه لعرض البيانات من قواعد البيانات الضخمة، حيث إنّ التحرّك بين آلاف السجلات واحدا بعد آخر يبدو نوعا من العبث.. لهذا لا بدّ من إنشاء واجهة أكثر ملاءمة لهذا الوضع.. وكحل مبدئي، تعال نستخدم قائمة مركّبة ComboBox لعرض أسماء الكتب، بحيث يختار المستخدم منها اسم الكتاب مباشرة بدلا من ضغط أزرار الانتقال.. صمّم النموذج ليبدو كما في الصورة، وهو موجود في المشروع BookList المرفق بهذا الكتاب:

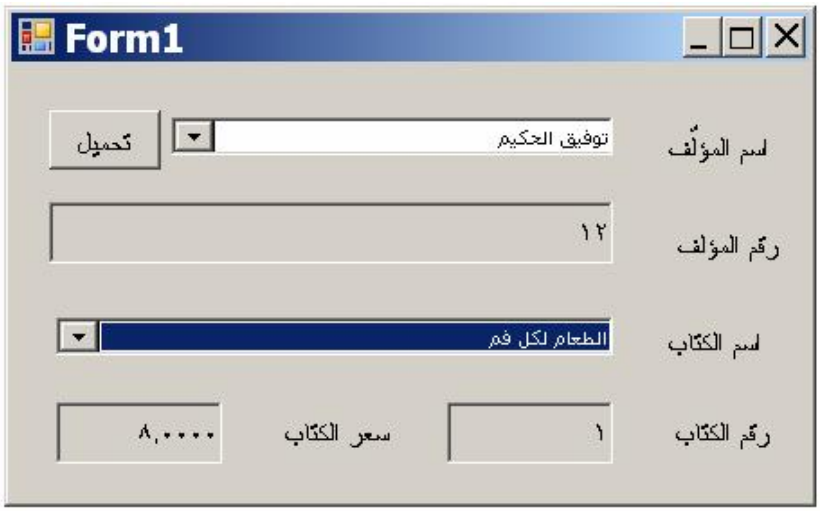
تعرف طبعا كيف تربط مربعي النص اللذين يعرضان اسم المؤلف ورقم الكتاب.. ما يهمنا الآن هو كيفية ربط القائمة المركبة.

حدّد القائمة المركّبة، ومن نافذة الخصائص اختر الخاصيّة DataSource، واضغط زر الإسدال الموجود في خانة قيمتها، ومن القائمة المنسدلة اختر Other Data Sources ثم Form1 List Instances، ثم DsAuthorBooks1.

بعد هذا انتقل إلى الخاصيّة DisplayMember في نافذة الخصائص، واضغط زر الإسدال الموجود في خانة قيمتها، واختر الجدول Authors، ومن حقوله اختر الحقل Book.

إذا شغّلت التطبيق الآن، وضغطت زرّ تحميل البيانات، فستجد أنّ القائمة المركّبة قد امتلأت بأسماء الكتب.. المدهش أنّك لو اخترت اسم أيّ كتاب من القائمة، فسيؤدي هذا إلى تغيير السجل الحالي، ومن ثم سيظهر رقمه واسم مؤلفه في مربّعي النصّ تلقائيا، وبدون أن تكتب سطرا واحدا من الكود لفعل هذا!

حسن.. نريد الآن تطوير المشروع السابق، بحيث نعرض أسماء المؤلفين في قائمة مركبة، ونعرض كتب المؤلف المحدد حاليا في قائمة مركبة أخرى، كما هو موضح في الصورة التالية:



اتبع هذه الخطوات:

١- أنشئ مشروعا جديدا اسمه AuthorsBooks_Lists.

٢- أنشأ مهيئ بيانات اسمه DaAuthors يعيد أسماء المؤلفين وأرقامهم.

٣- أنشأ مهيئ بيانات اسمه DaBooks يعيد أسماء الكتب وأرقامها وأسعارها.. ويعيد أيضا الحقل AuthorID لكي نستخدمه في إنشاء علاقة بين الجدولين.

٤- أنشئ مجموعة بيانات محددة النوع اسمها DsAuthorsBooks تحتوي على الجدولين.

٥- اضغط الأداة DsAuthorsBooks1 في صينية المكونات بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Edit in Dataset Designer.. ستظهر لك نافذة مخطّط XML.. أنشئ علاقة بين الجدولين اسمها Authors_Books كما فعلنا من قبل.

٦- بالنسبة للقائمة المركّبة التي ستعرض أسماء المؤلّفين، ومربّع النصّ الذي سيعرض رقم المؤلف، لن تختلف طريقة ربطهما بمجموعة البيانات في شيء عن المثال السابق.

٧- أما بالنسبة للقائمة المركّبة التي ستعرض أسماء الكتب فسيختلف الأمر قليلا.. حدّد الخاصيّة DataSource في نافذة الخصائص، وضع فيها القيمة DsAuthorsBooks1.. ثمّ حدّد الخاصيّة DisplayMember، واضغط زرّ الإسدال الموجود في خانة القيمة.. هذه المرة لا تختر جدول الكتب، بل اختر جدول المؤلّفين Authors.. ستجد ضمن عناصره الفرعيّة عنصرا جديدا، هو Authors_Books.. هذا العنصر هو اسم العلاقة التي أنشأناها.. أسدل فروع هذا العنصر.. ستجد تحته أسماء حقول جدول الكتب.. اختر الحقل Book.. بهذا لن تعرض قائمة الكتب كلّ الكتب الموجودة في قاعدة البيانات، بل ستعرض فقط الكتب التي تنتمي إلى المؤلّف الحاليّ من خلال العلاقة بينهما.

لاحظ أنك تستطيع أداء هذا من الكود باستخدام الجملة التالية:

```
CmbBook.DisplayMember =
        "Authors.Authors_Books.Book";
```

٨- بالنسبة لمربّع النصّ الذي سيعرض رقم الكتاب، اربط الخاصـيّة Text بالحقـل ID الموجود في العلاقة Authors_Books تحت جدول المـؤلّفين Authors.. وافعـل شيئا مشابها لربط مربع النص الأخير بالحقل Authors.Authors_Books.Price.

لاحظ أنك تستطيع أداء هذا من الكود كما يلي:

```
TextBox1.DataBindings.Add("Text",
    DsAuthorsBooks1, "Authors.Authors_Books.ID");
TextBox2.DataBindings.Add("Text",
    DsAuthorsBooks1, "Authors.Authors_Books.Price");
```

٩- وأخيرا، اكتب الكود الذي يملأ مجموعة البيانات بسجلات الجدولين في حدث ضـغط زر التحميل:

```
DaAuthors.Fill(DsAuthorsBooks1, "Authors");
DaBooks.Fill(DsAuthorsBooks1, "Books");
```

الآن لو جرّبت البرنامج، فلا ريبَ أنّك ستتيه دهشةً وسعادةً، فلديك واجهة استخدام رائعـة، تعمل بطريقة مثاليّة، في برنامج لم نكتب فيه أكثر من سطرين من الكود!

لاحظ أننا لا نملك طريقة مباشرة لاستخدام العلاقة Authors_Books بطريقة عكسية فـي عملية الربط.. مثلا: لا تستطيع أن تجعل مربع نص يعرض مؤلف الكتاب الحالي بالجملـة التالية:

```
TextBox1.DataBindings.Add("Text",
        Ds, "Books.Authors_Books.Author");
```

فهذه الجملة ستسبب خطأ في البرنامج، لأن العنصر Authors_Books ليس جـزءا مـن جدول الكتب!.. إن عملية الربط تعتبر العلاقة جزءا من الجدول الرئيسـي فقـط، ولـيس الجدول الفرعي!

وقد واجهتنا هذه المشكلة في المشروع BindingTextBox الذي أنشأناه فـي بدايـة هـذا الفصل، فـنحن فـي هـذا المشـروع نعـرض جـدول الكتـب فـي جـدول عـرض DataGridView، ونريد أن نربط مربع النص باسم مؤلف الكتاب المحدد حاليا في جدول العرض.. في هذه الحالة لا يمكننا أن نستخدم الجملة التالية:

```
TextBox1.DataBindings.Add("Text",
        Ds, "Authors.Authors_Books.Author");
```

لأنها ستعرض في مربع النص اسم أول مؤلف فقط، ولن يتغير مهما تغير الكتاب الحالي.. السبب في هذا أن كائن الربط الخاص بجدول العرض، يتعامل مع سجلات جـــدول الكتـــب فقط، وليست له أي علاقة بجدول المؤلفين!

ولحل هذه المشكلة، عرفنا عمودا إضافيا اسمه Author وجعلنا خفيا، وجعلناه يحمل اســـم مؤلف الكتاب الحالي من خلال العلاقة بينهما كالتالي:

```
DataColumn Col = new DataColumn("Author",
   typeof(string), "Parent.Author", MappingType.Hidden);
```

ثم أضفنا هذا العمود إلى جدول الكتب كالتالي:

```
Ds.Books.Columns.Add(Col);
```

الآن صار من السهل ربط مربع النص بهذا العمود كالتالي:

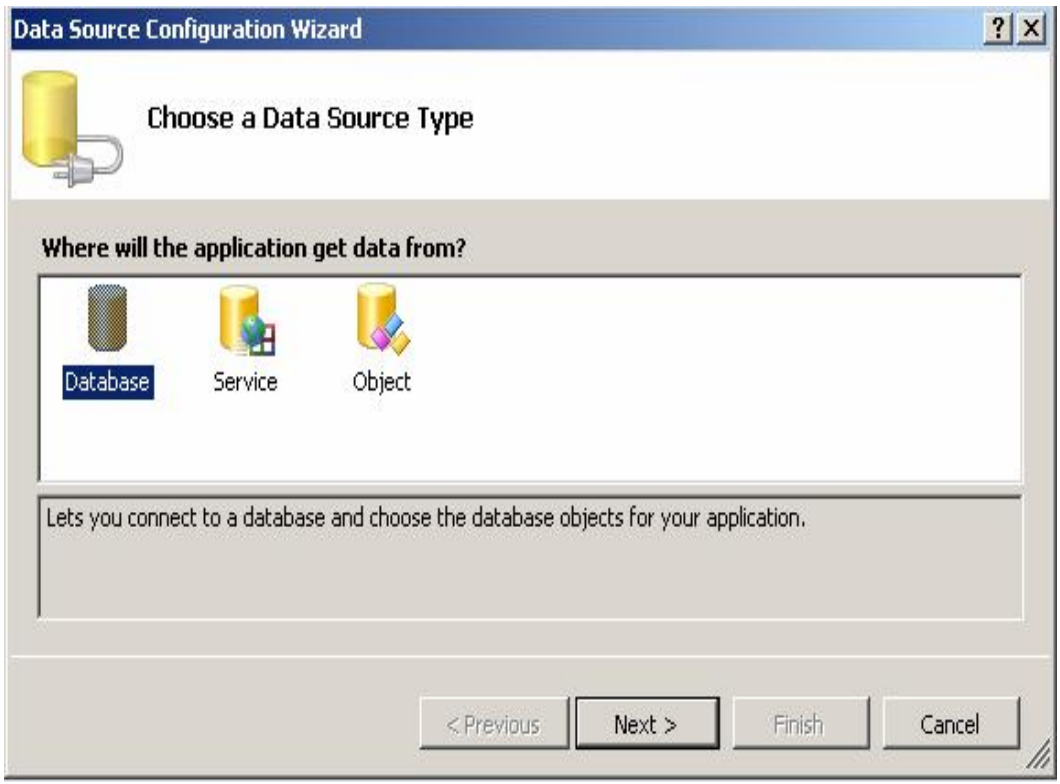
```
TxtAuthor.DataBindings.Add("Text", Ds, "Books.Author");
```

ولو جربت المشروع فستجده يعمل على ما يرام.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

المعالج السحري لتهيئة مصادر البيانات
Data Source Configuration Wizard

يستخدم هذا المعالج لإضافة مصادر البيانات إلى مشروعك.. ويمكنك تشغيله بضغط قائمة البيانات Data Menu من شريط القوائم الرئيسية أعلى مصمم النموذج، وضغط الأمر Add New Data Source.. ستظهر لك نافذة اختيار نوع مصدر البيانات، الموضحة في الصورة:



هذه النافذة تتيح لك اختيار أحد أنواع مصادر البيانات التالية:

١- قاعدة بيانات DataBase:

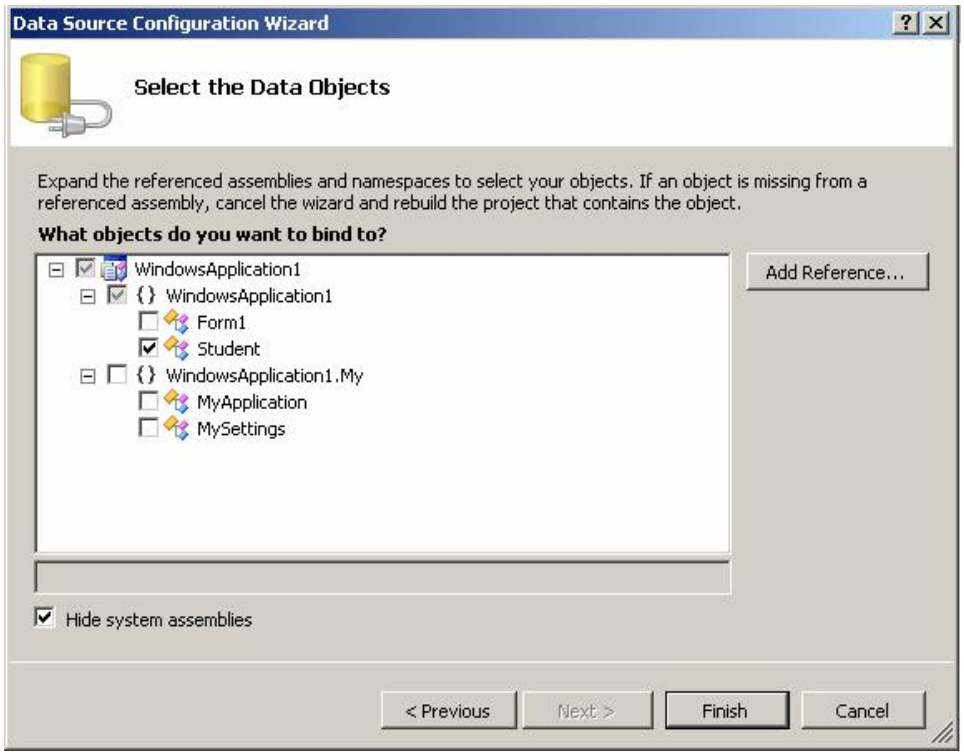
يتيح لك هذا النوع إنشاء مصدر بيانات يتعامل مع قاعدة بيانات، حيث يتم إنتاج مجموعة بيانات محددة النوع Typed DataSet ومهيئات الجداول اللازمة للتعامل مع كل جدول من جداولها.

٢- خدمة Service:

يتيح لك هذا النوع إنشاء مصدر بيانات يتعامل مع خدمة إنترنت Web Service.. هذا النوع خارج نطاق هذا الكتاب.

٣- كائن Object:

يتيح لك هذا النوع إنشاء مصدر بيانات يتعامل مع أي كائن في مشروعك.. مثلا، لو عرفت فئة اسمها Students، فيمكنك جعلها مصدر بيانات، باختيار هذا النوع ثم ضغط Next واختيارها من النافذة التالية كما هو موضح في الصورة:

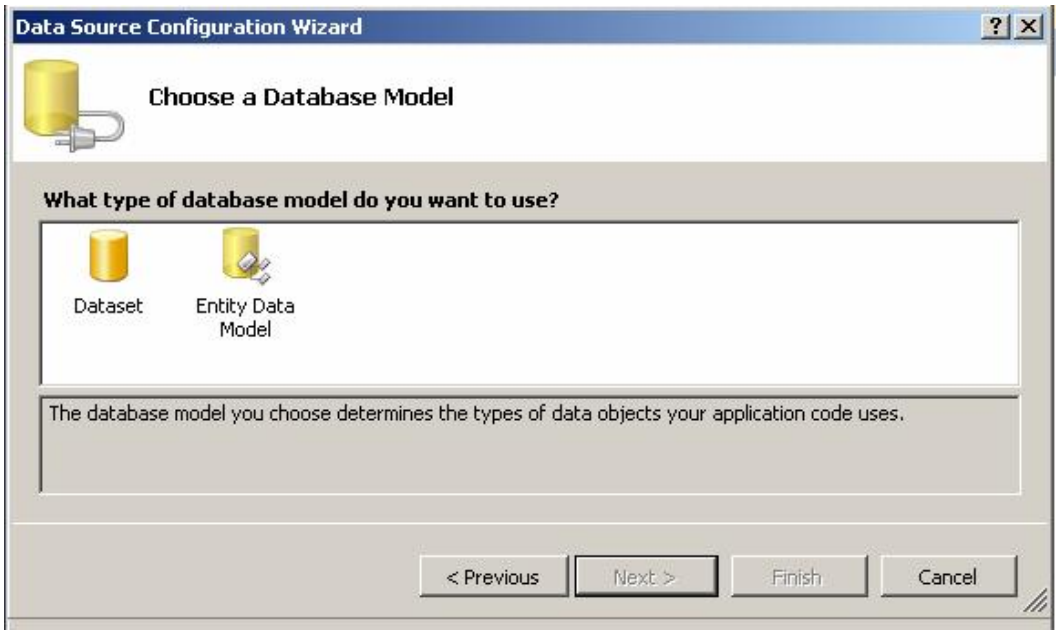


ويمكنك اختيار أي فئة من فئات إطار العمل لاستخدامها كمصدر بيانات لو أردت.. لفعل هذا أزل علامة الاختيار من مربع الاختيار Hide System Assemblies لتظهر فئات إطار العمل في القائمة.. وإذا أردت عرض فئات من خارج مشروعك، فاضغط الزر Add Reference وأضف مرجعا إلى المكتبات التي توجد بها.. وبعد أن تختار مصدر بيانات أو أكثر من القائمة، اضغط الزر Finish.

٤- تطبيق SharePoint:

يتيح لك هذا النوع إنشاء مصدر بيانات يناسب تطبيق SharePoint 2010.. هذا الموضوع خارج نطاق هذا الكتاب.

دعنا الآن نتعامل مع النوع الأكثر ملاءمة لنا هنا.. اختر النوع Database واضغط الزر Next.. ستظهر لك نافذة اختيار نموذج قاعدة البيانات Database Model كما في الصورة:



ستجد في هذه النافذة خيارين:

أ- DataSet:

يتم إنشاء مجموعو بيانات محددة النوع، واستخدامها كمصدر بيانات.

ب- Entity Data Model:

هذا الاختيار مناسب للمشاريع التي تستخدم LinQ-To-SQL و Entity Framework، وسنؤجله إلى الكتاب القادم بإذن الله.

اختر DataSet واضغط Next.

ستظهر لك نافذة الاتصال بقاعدة البيانات، وقد تعرفنا عليها كثيرا من قبل.. اختر الاتصال بقاعدة بيانات الكتب Books.mdf، واضغط Next.

ستظهر لك نافذة تسألك إن كنت تريد حفظ نص الاتصال في إعدادات المشروع Settings أم لا.. اترك علامة الاختيار كما هي، وعدل الاسم الذي تريد أن تستخدمه لحفظ نص الاتصال في الإعدادات لو أردت، واضغط Next.

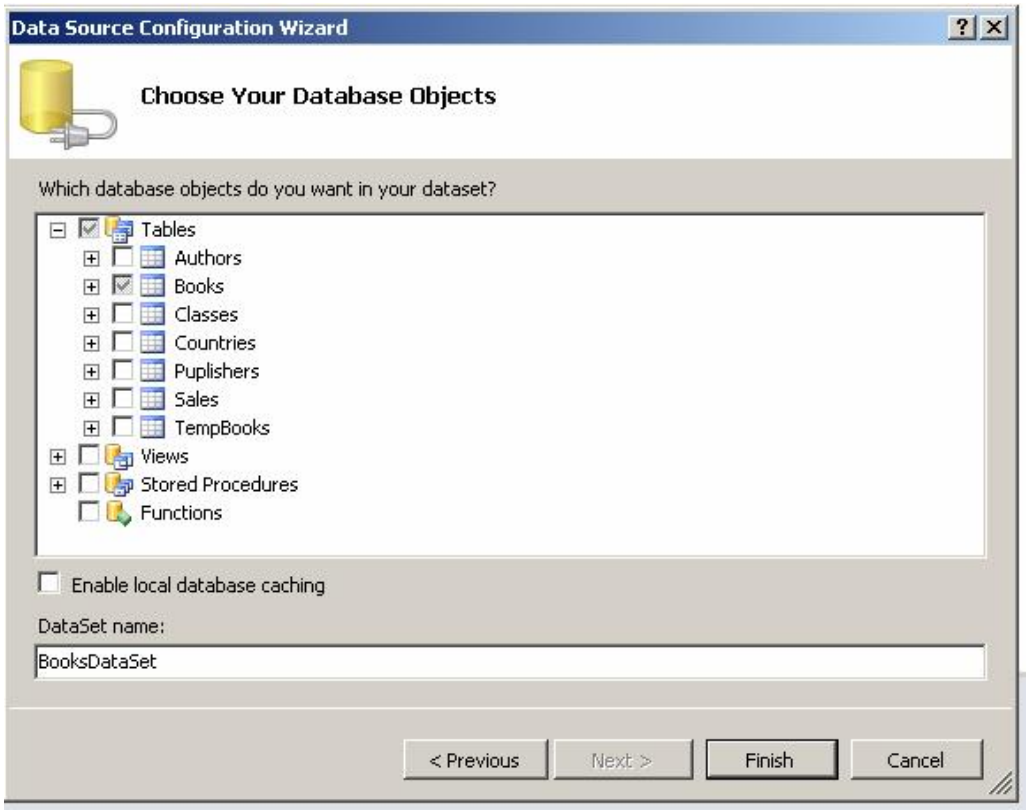
انتظر لحظة إلى أن يتم الاتصال بقاعدة البيانات وتحميل مكوناتها.. ستظهر لك نافذة تتيح اختيار كائنات قاعدة البيانات التي تريد التعامل معها، وستجد فيها كل الجداول والعروض والإجراءات المخزنة المتاحة في قاعدة البيانات، كما تبين الصورة:

اختر جدولي المؤلفين والكتب في مثالنا هذا.. لاحظ أن وضع علامة الاختيار بجوار اسم الجدول يحدد كل أعمدته.. لكنك تستطيع ضغط العلامة + المجاورة لاسم الجدول لإسدال أعمدته، حيث يمكنك إزالة علامة الاختيار المجاورة لبعضها، وبهذا توفر على برنامجك تجميل بيانات لا ضرورة لها.

ويمكنك تعديل الاسم الافتراضي لفئة مجموعة البيانات محددة النوع، من خلال مربع النص السفلي.

تستطيع الآن أن تضغط Finish لإنهاء المعالج السحري وإنشاء مجموعة البيانات، أو يمكنك وضع علامة الاختيار أمام الاختيار:

Enable Local Database Caching

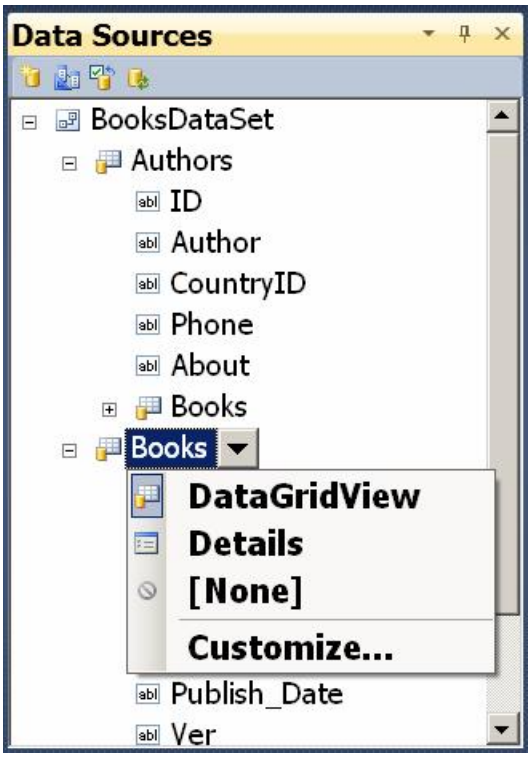


هذا الاختيار يتيح حفظ بعض بيانات الجداول على جهاز المستخدم لتكون جاهزة للاستخدام، وذلك إذا كان معدل تغيرها في قاعدة البيانات بطيئا، مما يقلل من عدد مرات الاتصال بالخادم، وبالتالي يحسن أداء وسرعة البرنامج.. إذا اخترت هذا الخيار، فعليك أن تضغط Next لمواصلة المعالج.. لكننا سنترك هذا إلى الكتاب القادم.. اضغط Finish لإنهاء المعالج.

سيؤدي هذا إلى إضافة الملف BooksDataSet.xsd إلى المشروع.. ولو نقرت هذا الملف مرتين، فسيظهر مصمم مجموعة البيانات، الذي سيعرض لك جدول المؤلفين وجدول الكتب والعلاقة بينهما، كما ستجد فيه مهيئ جدول المؤلفين AuthorDataAdapter ومهيئ جدول الكتب BookDataAdapter.

متصفح مصادر البيانات:

لو فتحت القائمة الرئيسية Data وضغطت الأمر Show Data Sources، فسيظهر لك متصفح مصادر البيانات Data Sources Explorer كما هو موضح في الصورة.



هذه النافذة تعرض جميع مصادر البيانات الموجودة في المشروع (مثل مجموعات البيانات).. وستجد فيها اسم مجموعة البيانات BoksDataBooks التي أنشأها معالج مصادر البيانات، ولو أسدلت عناصرها، فستجد تحتها جدولي المؤلفين والكتب، ولو أسدلت كلا منهما فستجد تحته أسماء أعمدته.

ويتيح لك متصفح مصادر البيانات إضافة مصادر بيانات جديد وتعديل المصادر الموجودة به، وذلك من خلال الأزرار التي تظهر أعلاه، وهي:

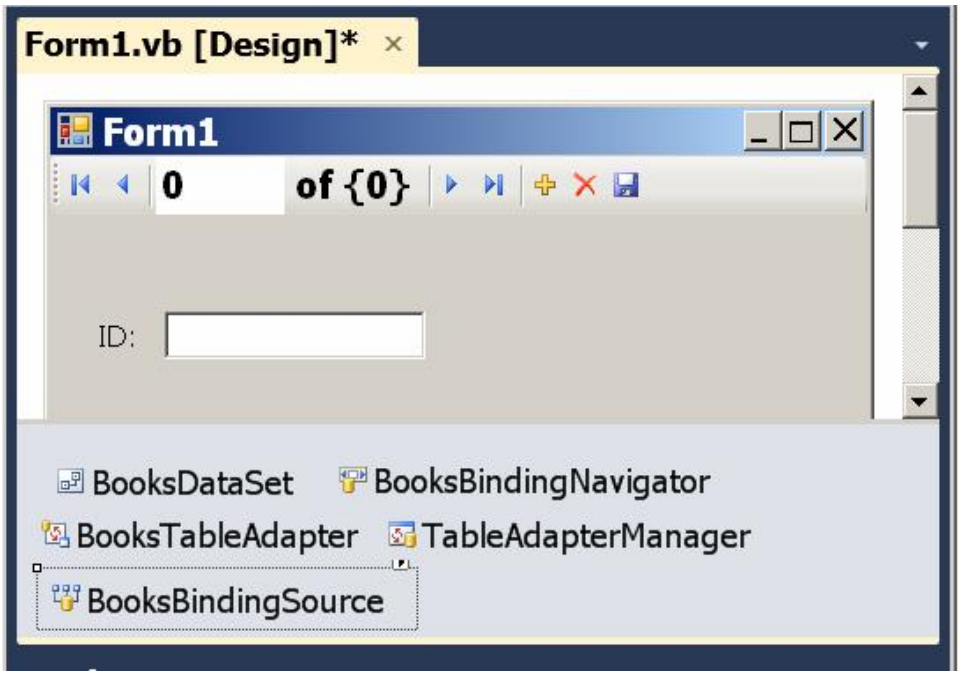
Add New Data Source: يؤدي ضغط هذا الزر إلى تشغيل المعالج السحري لمصادر البيانات.

Edit Data Source With Designer: يؤدي ضغط هذا الزر فتح مصمم مجموعة البيانات لتحريرها.

Configure Data Source With Wizard: يؤدي ضغط هذا الزر إلى تشغيل المعالج السحري لمصادر البيانات، لكنه يعرض نافذة اختيار كائنات قاعدة البيانات مباشرة، لتستطيع إضافة الجداول أو حذفها.

Refresh: يؤدي ضغط هذا الزر إلى إنعاش مصدر البيانات، للتفاعل مع أية تغييرات حدثت في قاعدة البيانات.

ويقدم لك متصفح مصادر البيانات تسهيلات هائلة لتصميم النماذج التي تعرض البيانات، فبمجرد سحب اسم أي حقل من متصفح مصادر البيانات وإلقائه على النموذج، يتم إضافة العديد من الأدوات إلى النموذج كما هو موضح في الصورة:



وكما تلاحظ من الصورة، فإن الأدوات التي أضيفت هي:

١- لافتة تعرض اسم الحقل، اسمها البرمجي XLabel حيث X هو اسم الحقل.

٢- مربع نص يعرض قيمة الحقل، اسمه البرمجي XTextBox.

٣- نسخة من فئة مجموعة البيانات BooksDataSet لاستخدامها في الحصول على البيانات.

٤- نسخة من مهيئ الجدول الذي يوجد به الحقل.. فمثلا لو سحبت الحقل Book فسيضاف مهيئ الجدول BooksTableAdapter إلى صينية المكونات.

٥- نسخة من مدير التوصيل TableAdapterManager للتحكم تحديث مجموعة البيانات.

٦- أداة مصدر الربط BindingSource لاستخدامها في ربط الأدوات بمجموعة البيانات.. وسنتعرف على هذه الأداة بعد قليل.

٧- نسخة من الأداة BindingNavigator لتتيح للمستخدم التحرك عبر السجلات.. وسنتعرف على هذه الأداة بعد قليل.

٨- يتم إنتاج كود تحميل البيانات في حدث تحميل النموذج Load آليا.. مثلا، يتم إنتاج الكود التالي ليملأ جدول المؤلفين بالبيانات من قاعدة البيانات:

```
this.AuthorsTableAdapter.Fill(this.BooksDataSet.Authors);
```

٩- يضاف زر لحفظ التغييرات إلى شريط موجه الربط BindingNavigator، ويضاف الكود التالي إلى حدث ضغط هذا الزر:

```
this.Validate( );
this.AuthorsBindingSource.EndEdit( );
this.TableAdapterManager.UpdateAll(this.BooksDataSet);
```

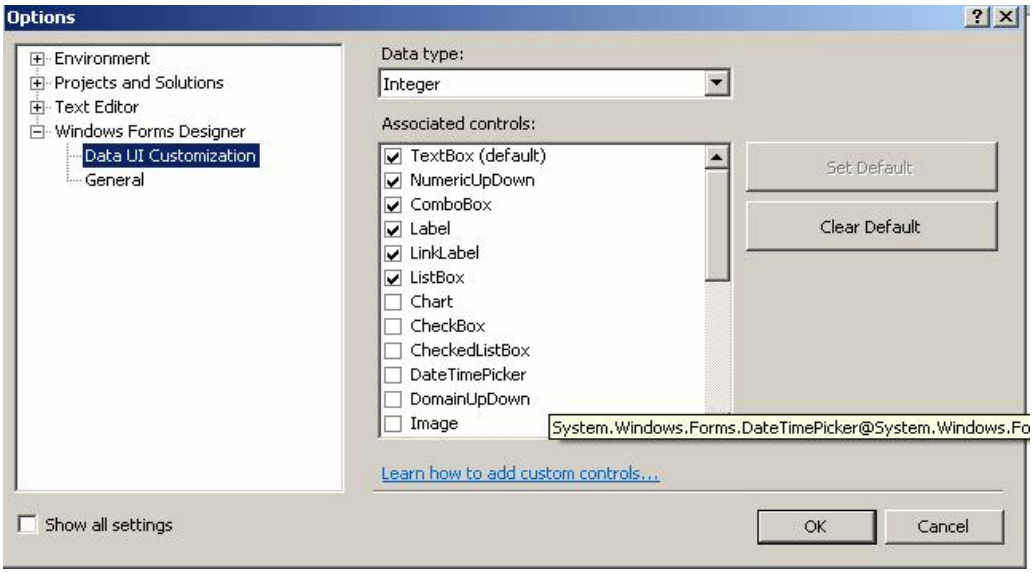
أليس شيئا رائعا؟.. أنت لا تحتاج إلى فعل أي شيء تقريبا، سوى سحب الحقول وإلقائها على النموذج لتحصل على برنامج كامل الوظيفة!

ويمكنك ربط الحقل بالأداة بطريقة أخرى، وذلك بوضع الأداة على النموذج أولا، ثم سحب الحقل من نافذة المصادر وإلقائه على الأداة.. هذا سيضبط خصائص الأداة تلقائيا لتعرض قيمة هذا الحقل.

كما أنك لست مجبرا على عرض قيمة الحقل في مربع نص، فلو حددت اسم الحقل في متصفح مصادر البيانات، فسيظهر زر إسدال بجواره، ولو ضغطته فستظهر قائمة موضعية، بها أسماء الأدوات التي يمكنك استخدامها لعرض قيمة الحقل.. ولو اخترت القيمة None فلن يتم وضع أدوات لعرض هذا الحقل عند إلقائه على النموذج.

وفي الوضع الافتراضي يكون مربع النص TextBox هو الأداة المستخدمة لعرض قيمة الحقل، لكنك تستطيع اختيار أية أداة أخرى لجعلها تعرض قيمته.. ولو لم تجد الأداة

المناسبة بين الأدوات الظاهرة في القائمة، فاضغط الأمر Customize الموجود في نهاية القائمة لعرض النافذة الموضحة في الصورة:



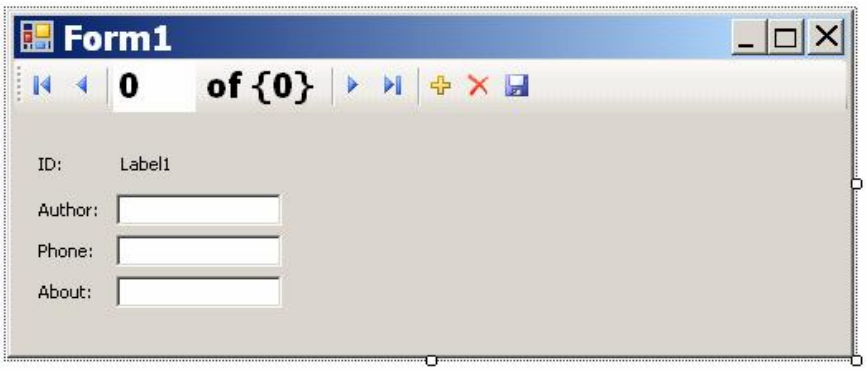
في هذه النافذة يمكنك اختيار نوع البيانات من القائمة المنسدلة Data Type، لتظهر في القائمة السفلية الأدوات التي يمكنها عرض هذا النوع من البيانات، حيث ستجد علامة الاختيار بجوار الأدوات المسموح باستخدامها، ويمكنك وضع علامة الاختيار بجوار أية أدوات أخرى تريد أن تسمح باستخدامها مع هذا النوع، ثم تضغط OK.

وتتيح لك نافذة مصادر البيانات التعامل مع الجدول كله دفعة واحدة.. فلو حددت اسم الجدول Authors في متصفح مصادر البيانات، فسيظهر زر إسدال بجواره، وعند ضغطه ستظهر قائمة موضعية بها الخيارات التالية:

- None: لا يتم وضع أية أدوات على النموذج عند إسقاط الجدول عليه.
- DataGridView: لو اخترت هذا الخيار، وسحبت جدول المؤلفين وألقيته على النموذج، فسيضاف جدول عرض إلى النموذج، وسيحتوي على أعمدة لعرض حقول جدول المؤلفين.. ويؤدي ضغط أزرار التحرك الموجودة على شريط موجه الربط، إلى تغيير السجل المحدد حاليا في جدول العرض، كما أن ضغط زر

الحذف سيحذف السجل المحدد حاليا، وضغط زر الإضافة سيضيف سجلا جديدا إلى نهاية جدول العرض.

- Details: لو اخترت هذا الخيار، وسحبت جدول المؤلفين وألقيته على النموذج، فستضاف أداة عرض خاصة بكل حقل على حدة، وبجوارها لافتة تحمل اسم هذا الحقل.. ويختلف نوع أداة العرض الخاصة بكل حقل على حسب الاختيار الذي حددته لكل حقل (كما شرحنا سابقا).. مثلا: قبل أن تسحب جدول المؤلفين، عليك أن تغير نوع أداة عرض الحقل ID إلى لافتة Lable حتى لا تسمح للمستخدم بتغيره، كما يمكنك اختيار العنصر None مع الحقل CountryID لمنع عرض رقم دولة المؤلف.. بعد هذا لو سحبت جدول المؤلفين وألقيته على النموذج، فسيتم وضع الأدوات عليه كما في الصورة:



الأمر رائع فعلا، فقد كان تصميم النماذج في مشاريع قواعد البيانات يستهلك معظم وقت إنتاج البرنامج.. لكن الآن صار الأمر في منتهى البساطة، فأنت تحصل على معظم العمل بالسحب والإسقاط، مع الكثير من الكود المولد آليا!

وبنفس الطريقة يمكنك إضافة نماذج أخرى إلى المشروع، وإسقاط جداول أخرى عليها.

وتمتلك نافذة مصادر البيانات ميزة إضافية هامة، هي السماح لك بعرض البيانات المترابطة.. ولو أسدلت عناصر جدول المؤلفين، فستجد آخر عنصر منها اسمه Books.. هذا العنصر أضيف ليمثل العلاقة المعرفة بين جدول المؤلفين وجدول الكتب في مجموعة البيانات BooksDataSet.. ولو أسدلت العنصر Books، فستجد تحته كل حقول جدول

الكتب، ولو سحبتها وألقيتها على النموذج، فستعرض بيانات كتب المؤلف الحالي.. لاحظ أنه من غير المنطقي عرض كل حقل فرعي على حدة إذا كنت تتعامل مع علاقة واحد بمتعدد One-To-Many.. فالمناسب في مثالنا هذا، استخدام جدول لعرض كتب المؤلف الحالي، كما ترى في الصورة:

وستجد هذا التصميم في التطبيق DataSourceWizard المرفق بهذا الكتاب.

لو شغلت هذا التطبيق فسيمكنك الانتقال بين المؤلفين باستخدام شريط موجه الربط، حيث ستعرض الأدوات العلوية بيانات المؤلف الحالي، وسيعرض الجدول السفلي كتب هذا المؤلف.. هذا مشروع Master-Details كامل يعمل بكفاءة دون أن نكتب فيه حرفا واحدا من الكود!.. أليس شيئا مثيرا؟

لاحظ أن وجود العمود AuthorID في جدول العرض لا معنى له.. لكن للأسف، لو حاولت إزالة هذا العمود باختيار None من القائمة المنسدلة للحقل AuthorID قبل سحب عنصر العلاقة Books على النموذج، فلن تنجح.. فجدول العرض يعرض كل الأعمدة شئت أم أبيت، وتكون كل هذه الأعمدة أعمدة مربعات النصوص DataGridViewTextBoxColumn مهما كان نوع الأداة التي اخترتها لعرض قيمة

الحقل!.. لهذا عليك تحديد جدول العرض واستخدام نافذة الخصائص لحذف هذا العمود من مجموعة أعمدة جدول العرض Columns Collection.. وسنتعرف على جدول العرض بالتفصيل في الفصل التالي.

ورغم كل التسهيلات التي تمنحها لنا نافذة مصادر البيانات، إلا أنها أحيانا لا تعطينا بالضبط ما نريده.. مثلا: لو أردت عرض أي حقل في قائمة List أو قائمة مركبة ComboBox، فإن سحب الحقل وإلقائه على النموذج يربط الخاصية Text التابعة لهاتين الأداتين بالحقل، ولا يتم ملؤهما بقيم الحقل!

ولحل هذه المشكلة، عليك التدخل يدويا، واستخدام نافذة الخصائص لإزالة الارتباط بالخاصية Text، واستخدام الخاصيتين DataSource و DataMember بدلا منها.

والتطبيق MasterDetails يريك مثالا على هذا.. لإنشاء مثل هذا التطبيق، افعل ما يلي:

- من نافذة مصادر البيانات، اسحب الحقل Authors.Author وألقه على النموذج لعرض اسم المؤلف الحالي.
- اختر عرض الحقل Authors.Books.Book في قائمة مركبة ComboBox، وألقه على النموذج ليعرض أسماء الكتب.
- من نافذة الخصائص افتح الخاصية (DataBindings) وأزل الارتباط مع الخاصية Text.. ويمكنك بدلا منها أن تنشئ ارتباطا مع الخاصية SelectedValue حتى يتم حفظ القيمة التي يختارها المستخدم آليا في السجل الحالي.
- توجه إلى الخاصية DataSource الخاصة بالقائمة المركبة، واضغط زر الإسدال، ومن القائمة اختر مصدر الربط AuthorsBindingSource وأسدل عناصره الفرعية.. ستجد تحته اسم العلاقة بين جدول المؤلفين وجدول الكتب وهي FK_Books_Authors.. اختر هذه العلاقة كمصدر للبيانات.. سيؤدي هذا إلى إضافة مصدر ربط جديد إلى البرنامج اسمه FKBooksAuthorsBindingSource، وستوضع قيمته تلقائيا في الخاصية DataSource!

- توجه إلى الخاصية DataMember، واضغط زر الإسدال، ومن القائمة اختر الحقل Book.. الآن تأكدنا أن القائمة المركبة ستعرض كتب المؤلف الحالي، لأننا ربطناها من خلال العلاقة بين المؤلفين والكتب.
- من نافذة مصادر البيانات اسحب الحقل Authors.Books.Price وألقه على النموذج.. حدد مربع النص وافتح نافذة الخصائص وافتح الخاصية (DataBindings)، وتوجه إلى الخاصية Text.. اضغط زر الإسدال، واختر العنصر FKBooksAuthorsBindingSource لربط مربع النص من خلال العلاقة.
- يمكنك تكرار هذا مع أكثر من حقل من حقول جدول الكتب.. مثلا، لو سحبت الحقل Publish_Date وألقيته على النموذج، فستظهر أداة اختيار التاريخ والوقت DateDateTimePicker لعرض قيمته.. وأيضا عليك أن تغير ارتباط الخاصية Value الخاصة بهذه الأداة، لتجعلها ترتبط من خلال المصدر FKBooksAuthorsBindingSource كما فعلنا مع مربع النص.

الآن سيكون شكل النموذج كالتالي:

لو شغلت البرنامج، فسيمكنك التحرك عبر المؤلفين باستخدام شريط موجه الربط، حيث ستظهر كتب المؤلف الحالي في القائمة المركبة، وكلما اخترت كتابا منها، يظهر تاريخ

نشره في أداة التاريخ، وسعره في مربع النص.. وبهذا نكون حصلنا على طريقة أخرى لعرض البيانات الرئيسية والتفاصيل وتفاصيل التفاصيل!.. صحيح أننا أجرينا بعض التعديلات اليدوية هذه المرة، ولكن صحيح أيضا أننا إلى الآن لم نكتب سطرا واحدا من الكود بأنفسنا!.. مرحى، ما أمتع الكسل!

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

واجهة مزود مدير التسلسل
ICurrencyManagerProvider Interface

تمتلك هذه الواجهة العنصرين التاليين:

مدير التسلسل CurrencyManager:

تعيد مدير التسلسل CurrencyManager التابع لمصدر البيانات الحالي..

معرفة مدير التسلسل التابع GetRelatedCurrencyManager:

أرسل إلى هذه الوسيلة اسم القائمة أو العمود الموجود في مصدر البيانات الحالي، لتعيد إليك مدير التسلسل CurrencyManager الخاص به.. ويمكنك أن ترسل نصا فارغا "" أو Nothing إلى هذه الوسيلة، وفي هذه الحالة ستعيد إليك مدير التسلسل الخاص بمصدر البيانات ككل، وهو نفس مدير التسلسل الذي تحصل عليه من الخاصية CurrencyManager.

واجهة إلغاء إضافة الجديد
ICancelAddNew Interface

تضيف هذه الواجهة إلى الفئة التي تمثلها القدرة على قبول العنصر الجديد المضاف أو التراجع عن إضافته، وهي تملك الوسيلتين التاليتين:

إلغاء الجديد CancelNew:

أرسل إلى هذه الوسيلة رقم العنصر الذي أضفته سابقا إلى المجموعة، لتقوم بالتراجع عن إضافة (تقوم بحذفه).

إنهاء الجديد EndNew:

أرسل إلى هذه الوسيلة رقم العنصر الذي أضفته سابقا إلى المجموعة، لتقوم بقبوله نهائيا.. هذا يعني أنك لا تستطيع استخدام الوسيلة CancelNew بعد هذا للتراجع عن إضافة هذا العنصر.

واجهة إطلاق أحداث التغير
IRaiseItemChangedEvents Interface

تمتلك هذه الواجهة الخاصية الوحيدة التالية:

إطلاق أحداث تغير العنصر RaisesItemChangedEvents:

تعيد هذه الخاصية True، إذا كانت الفئة التي تمثل هذه الواجهة ستطلق الحدث ListChanged إذا حدث تغير في أحد عناصر القائمة الداخلية الخاصة بها.

فئة قائمة الربط عامة النوع BindingList<T> Class

هذه الفئة موجودة في النطاق System.ComponentModel، وهي ترث الفئة Collection<T>، وتمثل الواجهات IbindingList و IList و ICancelAddNew و IRaiseItemChangedEvents.

وتعمل هذه الفئة كمجموعة عامة النوع Generic Type Collection، تدعم تقنية ربط البيانات Binding.

ولحدث إنشاء هذه الفئة صيغتان:

١- الصيغة الأولى بدون معاملات.

٢- والصيغة الثانية تستقبل قائمة عامة النوع IList<T>، لتنسخ عناصرها إلى قائمة الربط.

وإضافة إلى ما ترثه من الفئة الأم، وما تمثله من خصائص ووسائل الواجهات المذكورة، تمتلك هذه الفئة الوسيلتين الجديدتين التاليتين:

تصفير الارتباطات ResetBindings:

تطلق الحدث ListChanged مع إرسال القيمة Reset إلى الخاصية e.ListChangedType.

تصفير العنصر ResetItem:

تطلق الحدث ListChanged مع إرسال القيمة ItemChanged إلى الخاصية e.ListChangedType.

واجهة مصدر القائمة
IListSource Interface

تعمل هذه الواجهة كمصدر للحصول على قائمة List من كائنات لا تمثل واجهة القائمة IList، مما يجعل من الممكن استخدام هذه الكائنات كمصدر للبيانات DataSource عند ربطها بأدوات عرض البيانات.

وتمتلك هذه الواجهة العنصرين التاليين:

تحتوي على مجموعة قوائم ContainsListCollection:

تعيد True إذا كانت المجموعة الخاصة بالكائن الحالي تحتوي على قوائم داخلية.

الحصول على القائمة GetList:

تعيد مجموعة تمثل واجهة القائمة IList، تحتوي على عناصر الكائن الحالي.

فئة مصدر الربط BindingSource Class

هذه الفئة ترث الفئة Component لهذا ستجدها في صندوق الأدوات Toolbox تحت الشريط Data، ويمكنك إضافتها إلى صينية مكونات النموذج.
كما تمثل هذه الفئة الواجهات IList و IBindingListView و IcurrencyManagerProvider و ICancelAddNew.
وتحتوي هذه الفئة على قائمة داخلية Internal List تحتوي على عناصر مصدر البيانات، ليتم ربطها بالأدوات الموضوعة على النموذج، وبهذا تسهل هذه الفئة عملية الربط Binding وتتيح لك التحكم فيها كما سنرى بعد قليل.

ولحدث إنشاء هذه الفئة الصيغ التالية:

١- الصيغة الأولى بدون معاملات.

٢- الصيغة الثانية لها معامل واحد من نوع الواجهة IContainer، وهو يستقبل الأداة الحاوية التي سينتمي إليها مصدر البيانات، ليتعامل مع الأدوات الموضوعة عليها.. تذكر أن الأدوات الحاوية تشمل النموذج Form واللوحة Panel ومربع التجميع GroupBox... إلخ.

٣- الصيغة الثالثة تستقبل الكائن Object الذي يعمل كمصدر للبيانات، ونصا يمثل اسم عنصر البيانات.

وإضافة إلى ما تمثله من خصائص الواجهات المذكورة، تمتلك هذه الفئة الخصائص التالية:

مصدر البيانات DataSource:

تستقبل هذه الخاصية الكائن Object الذي يعمل كمصدر البيانات.. هذا سيؤدي إلى ما يلي:

- إذا كان الكائن من النوع T، فإن نوع عناصر القائمة الداخلية للفئة BindingSource سيحدد على أنه من النوع T، ولن تقبل هذه القائمة أي بيانات لا يمكن تحويلها إلى هذا النوع.

- إذا وضعت في هذه الخاصية القيمة Nothing، فستظل القائمة الداخلية غير محددة النوع، وستأخذ نوع أول عنصر تضيفه إليها باستخدام الوسيلة Add.. لاحظ أن خطأ سيحدث في البرنامج إذا وضعت قيمة في الخاصية DataMember بينما للخاصية DataSource القيمة Nothing.
- إذا كان الكائن بسيطا لا يحتوي على قائمة من العناصر، فإن القائمة الداخلية ستكون فارغة.
- إذا كان الكائن الذي وضعته في هذه الخاصية مصفوفة Array أو مجموعة Collection، فإن عناصرها ستوضع في القائمة الداخلية.
- إذا كان الكائن معقدا ويحتوي على قائمة من العناصر أو أكثر من قائمة، فيجل عليك ذكر اسم القائمة في الخاصية DataMember (كاسم الجدول في مجموعة البيانات مثلا)، حيث ستوضع عناصر هذه القائمة في القائمة الداخلية.

ويمكنك أن تضع في هذه الخاصية نوع أحد الكائنات بدلا من أن تضع الكائن نفسه.. فبدلا من أن تضع في هذه الخاصية مجموعة بيانات كالتالي:

Bs.DataSource = Ds

يمكنك أن تضع نوع مجموعة البيانات كالتالي:

Bs.DataSource = Ds.GetType( )

وإذا كانت لديك مجموعة بيانات محددة النوع Typed DataSet اسمها BooksDs فيمكنك استخدام نوعها كمصدر بيانات كالتالي:

Bs.DataSource = GetType(BooksDs)

ولكن، فيم يفيدنا هذا؟

في بعض الأحيان تحتاج إلى تصميم بعض أدوات عرض البيانات في وقت التصميم (مثل جدول عرض البيانات DataGridView)، وهذا معناه أنك تحتاج إلى عرض أعمدة الجداول المرتبطة في هذه الأداة.. لكن في وقت التصميم قد لا تكون هناك كائنات معرفة من الفئات التي تعمل كمصادر للبيانات، لهذا تسمح لك هذه الخاصية بوضع نوع هذه الفئات فيها، لتستنتج منه طريقة العرض المطلوبة.

ويمكنك وضع قيمة هذه الخاصية بطريقة مرئية في وقت التصميم، وذلك باستخدام نافذة الخصائص، حيث سيعرض لك زر الإسدال شجرة العناصر المتاحة.. في هذه الشجرة ستجد عنصرين رئيسيين:

١- None: وهي القيمة الافتراضية، وهي تجعل لهذه الخاصية القيمة Nothing.

٢- Other Data Sources: وتحتها الاختياران التاليان:

أ. Project Data Sources: ويوجد تحتها كل فئات مصادر البيانات المتاحة في المشروع كله.. ويؤدي اختيار أي فئة من هذه الفئات، إلى إنشاء نسخة جديدة منها وإضافتها إلى النموذج.

ب. Form Data Sources: ويوجد تحتها كل الكائنات المعرفة في النموذج الحالي وتصلح كمصادر بيانات، مثل القوائم Lists ومجموعات البيانات DataSets وغيرها.

وفي الهامش السفلي للنافذة المسدلة، يوجد رابط اسمه:

Add Project data Source

عند الضغط عليه يتم تشغيل المعالج السحري لتهيئة مصادر البيانات Data Source Configuration Wizard، ليمكنك إنشاء مصدر بيانات جديد وإضافته تحت الفرع Project Data Sources.

عنصر البيانات DataMember:

تستقبل اسم الخاصية أو اسم القائمة أو العمود الموجود في مصدر البيانات، والذي يتم أخذ البيانات منه.

القائمة List:

تعيد نسخة من الواجهة IList تحتوي على القائمة الداخلية التي تحتوي على العناصر المرتبطة.. لاحظ أن نوع القائمة العائدة يتحدد تبعا لما يلي:

قيمة الخاصية DataSource	قيمة الخاصية DataMember	القائمة العائدة من الخاصية List
null	نص فارغ	مصفوفة قائمة ArratList فارغة.
null	أي قيمة	عند قراءة الخاصية List سيحدث خطأ في البرنامج.
مصفوفة		مصفوفة Array.
كائن يمثل الواجهة IListSource		القيمة العائدة من الوسيلة IListSource.GetList
كائن يمثل الواجهة IBindingList		نسخة من الواجهة IBindingList.
كائن يمثل الواجهة IList		نسخة من الواجهة IList.
كائن بسيط من النوع T لا يحتوي على قائمة		نسخة من الواجهة IBindingList<T> بها عنصر واحد.
كائن يمثل الواجهة ICustomType Descriptor		مصفوفة قائمة ArrayList بها عنصر واحد.
كائن يمثل الواجهة IEnumerable		مصفوفة قائمة ArratList نسخت إليها عناصر الكائن.
نوع المصفوفات Array Type	عنصر من النوع T	نسخة فارغة من الفئة BindingList<T>.
نوع يمثل الواجهة IListSource أو الواجهة ITypedList		نسخة جيدة فارغة من نوع هذا الكائن.

قيمة الخاصية DataSource	قيمة الخاصية DataMember	القائمة العائدة من الخاصية List
نوع يمثل الواجهة IList	عنصر من النوع T	نسخة فارغة من الفئة BindingList<T>.
نوع بسيط لا يحتوي على قائمة		نسخة فارغة من الفئة BindingList<T>.
نوع يمثل الواجهة ICustomTypeDescriptor		عند قراءة الخاصية List سيحدث خطأ في البرنامج.

الموضع Position:

تقرأ أو تغير موضع العنصر الحالي في القائمة الداخلية لمصدر البيانات، وهو العنصر الذي يتم عرضه في الأدوات المرتبطة بمصدر الربط.

الحالي Current:

تعيد كائنا Object يحتوي على العنصر الحالي في القائمة الداخلية، وهو العنصر الموجود في الموضع المحدد في الخاصية Position.

الترتيب Sort:

تحدد طريقة ترتيب العناصر في القائمة، وهي تستقبل نصا يحتوي على اسم العمود المستخدم في الترتيب، متبوعا باتجاه الترتيب (ASC أو DESC).

هل الربط متوقف IsBindingSuspended:

تعيد True إذا كان الربط متوقفا حاليا.

إطلاق أحداث تغير القائمة RaiseListChangedEvents:

إذا جعلت قيمة هذه الخاصية True (وهي القيمة الافتراضية)، فسينطلق الحدث BindingSource.ListChanged عندما يحدث تغيير في عناصر القائمة الداخلية.

وتمتلك هذه الفئة الوسائل التالية:

إلغاء التحرير CancelEdit:

تنهي عملية تحرير العنصر الحالي، وتلغي أي تغيير حدث له.

إنهاء التحرير EndEdit:

تنهي عملية تحرير العنصر الحالي، وتبقي على التغييرات التي حدثت له.

التحرك إلى الأول MoveFirst:

تجعل أول عنصر في القائمة الداخلية هو العنصر الحالي (Position = 0).

التحرك إلى الأخير MoveLast:

تجعل آخر عنصر في القائمة الداخلية هو العنصر الحالي (Position = Count -1).

التحرك إلى التالي MoveNext:

تجعل العنصر التالي في القائمة الداخلية هو العنصر الحالي (Position += 1).

التحرك إلى السابق MovePrevious:

تجعل العنصر السابق في القائمة الداخلية هو العنصر الحالي (Position -= 1).

إزالة الحالي RemoveCurrent:

تزيل العنصر الحالي من القائمة الداخلية.. لاحظ أن خطأ سيحدث في الحالات التالية:

- لو كانت للخاصية BindingSource.AllowRemove القيمة False.

- لو كانت القائمة الداخلية للقراءة فقط Read Only أو ثابتة الحجم Fixed Size.
- لو كان الموضع الحالي غير مقبول، سواء كان أصغر من صفر أو كان أكبر من أو يساوي عدد عناصر القائمة.

تصفير العنصر الحالي ResetCurrentItem:

تطلق الحدث ListChanged لتطلب من الأدوات التي تعرض العنصر الحالي أن تنعش القيم التي تعرضها.

تصفير الارتباطات ResetBindings:

تطلق الحدث ListChanged لتطلب من كل الأدوات المرتبطة بمصدر البيانات أن تنعش القيم التي تعرضها، وهي تستقبل معاملا منطقيا، إذا جعلت قيمته True فهذا معناه أن هناك تغييرا في مخطط مصدر البيانات نفسه (كحدوث تغيير في أعمدة الجدول)، وإذا جعلته False فهذا معناه أن التغيير قد حدث في بعض عناصر القائمة الداخلية فقط.

ويتم استدعاء هذه الوسيلة آليا عند تغير قيمة الخاصية DataSource أو الخاصية DataMember، كما يمكنك أن تستدعيها بنفسك إذا قمت بتغيير بعض العناصر في مصدر البيانات.

ولكن.. لماذا نحتاج إلى استخدام الوسيلة ResetBindings لإنعاش كل العناصر، بينما يكفينا إنعاش العنصر الحالي باستخدام الوسيلة ResetCurrentItem؟.. ألا تعرض الأدوات العنصر الحالي فقط؟

والإجابة هي أن بعض الأدوات تعرض أكثر من عنصر في نفس الوقت (كالقائمة ListBox وجدول عرض البيانات DataGridView)، بينما بعض الأدوات تعرض السجل الحالي فقط (مثل مربع النص واللافتة).. لهذا إذا حدث تغيير في عدد من العناصر وكنت تعرض البيانات في قائمة أو جدول عرض، فعليك باستدعاء الوسيلة ResetBindings، أما إذا كنت تستخدم أدوات عرض بسيطة كمربع النص واللافتة

وحدث تغير في العنصر الحالي، فاستخدم الوسيلة ResetCurrentItem.. أما إذا كان التغير في عنصر غير العنصر الحالي، فلا تحتاج إلى إنعاش الأدوات البسيطة، لأنها ستنعش نفسها تلقائيا عند الانتقال إلى العنصر الذي تغير.

تصفير العنصر ResetItem:

تطلق الحدث ListChanged لتطلب من الأدوات التي تعرض العنصر الذي تغير في السجل الحالي بأن تنعش القيمة التي تعرضها.. وتستقبل هذه الوسيلة رقم العنصر المراد إنعاشه.

ويتم استدعاء هذه الوسيلة آليا كلما حدث تغيير لأحد عناصر القائمة الداخلية، لكن بإمكانك استدعاؤها بنفسك أيضا.

إيقاف الربط SuspendBinding:

توقف ربط المصدر الحالي بالأدوات مؤقتا.

مواصلة الربط ResumeBinding:

تعيد ربط الأدوات بالمصدر الحالي.

وإضافة إلى ما تمثله من أحداث الواجهات التي تمثلها، تمتلك هذه الفئة الأحداث التالية، وكلها مألوف لنا لهذا لن نكرر شرحها هنا:

- **الربط اكتمل BindingComplete**
- **الحالي تغير CurrentChanged**
- **العنصر الحالي تغير CurrentItemChanged**
- **عنصر البيانات تغير DataMemberChanged**
- **مصدر البيانات تغير DataSourceChanged**
- **الموضع تغير PositionChanged**
- **خطأ البيانات DataError**

فئة مساعد ربط القوائم ListBindingHelper Class

تحتوي هذه الفئة على بعض الوسائل المشتركة Shared Methods، التي تستخدمها الفئة BindingSource في التعامل مع مصدر البيانات.. وهذه الوسائل هي:

معرفة القائمة GetList:

تستقبل الكائن الذي يعمل كمصدر بيانات، وتعيد قائمة البيانات التي يحتويها، والتي يمكن الارتباط بها إن وجدت، فإن لم توجد، فإن هذه الوسيلة تعيد كائن مصدر البيانات نفسه.

وتوجد صيغة أخرى لهذه الوسيلة، لها معامل ثان، يستقبل اسم الخاصية التي تعمل كعنصر البيانات في مصدر البيانات، للحصول على قائمة العناصر الخاصة بها.

معرفة اسم القائمة GetListName:

تعيد اسم القائمة إن وجدت، أو اسم نوع مصدر البيانات، ولها معاملان:

- الكائن الذي يعمل كمصدر للبيانات.
- مصفوفة من واصفات الخصائص PropertyDescriptor، التي تحدد القائمة المراد معرفة اسمها.

معرفة نوع عناصر القائمة GetListItemType:

هذه الوسيلة مماثلة في صيغتيها للوسيلة السابقة، إلا أنها تعيد كائن النوع Type، الذي يمثل نوع عناصر القائمة.

معرفة خصائص عناصر القائمة GetListItemProperties:

تعيد مجموعة واصفات الخصائص PropertyDescriptorCollection، التي تصف خصائص عنصر القائمة.. ولهذه الوسيلة ثلاثة معاملات:

- الكائن الذي يعمل كمصدر للبيانات.

- اسم عنصر البيانات.
- مصفوفة من واصفات الخصائص PropertyDescriptor، التي تحدد القائمة المراد التعامل معها.

وتوجد صيغتان أخريان لهذه الوسيلة، إحداهما تستقبل المعامل الأول فقط، والأخرى تستقبل المعاملين الأول والثالث فقط.

فئة موجه الربط BindingNavigator Class

هذه الفئة ترث فئة رف الأدوات ToolStrip Class، لهذا فهي تعمل كرف أدوات يعرض مجموعة من الأزرار، التي تتيح للمستخدم التحرك عبر سجلات مصدر البيانات وحذف السجل الحالي أو إضافة سجل جديد، كل هذا بدون أن تكتب أنت حرفا من الكود!

ويبدو رف الأدوات الذي يعرضه موجه الربط في وقت التصميم كما في الصورة:

لاحظ أن آخر زر على الشريط يتيح لك إضافة أزرار وأدوات جديدة إلى الشريط، بنفس الطريقة التي تعلمناها في كتاب برمجة الويندوز.. هذا معناه أنك تستطيع استغلال مساحة الشريط لإضافة أزرار وقوائم ومربعات نصوص ولافتات تؤدي أية وظائف أخرى خاصة بك (كالقص واللصق وعرض حالة البرنامج وغير هذا)، وبهذا لا تحتاج إلى إضافة رف أدوات آخر خاص بك.

والصورة التالية تريك كيف يبدو موجه الربط عند تشغيل المشروع Navigator المرفق بأمثلة هذا الكتاب:

مرة أخرى أذكرك: لو أضفت سجلا جديد بضغط زر الإضافة الموجود على شريط موجه الربط، أو حذفت السجل الحالي بضغط زر الحذف، أو غيرت قيمة أي حقل في السجل الحالي بتغيير قيمة أحد مربعات النص، فإن هذه التغييرات ستؤثر فقط على مجموعة البيانات DataSet، لكن تظل مهمة تحديث قاعدة البيانات متروكة لك.. وإذا كنت لا ترغب أن يعبث المستخدم بقيم بعض الحقول، فاجعل مربعات النصوص المناظرة لها للقراءة فقط، أو اربط هذه الحقول بلافتات منذ البداية.. مع ملاحظة أن تغيير المستخدم لقيمة المعرف ID لن تؤثر في شيء، لأن هذا الحقل مولد آليا، ومصدر البيانات لا يستطيع تغييره.

ولو لم تكن ترغب في أن يحذف المستخدم السجلات أو يضيف سجلات جديدة، فيمكنك إزالة زر الحذف أو زر الإضافة من فوق الشريط في وقت التصميم، أو يمكنك تعطيلهما، وسترى كيف نفعل هذا بعد قليل ونحن نتعرف على خصائص موجه الربط.

ولحدث إنشاء الفئة BindingNavigator الصيغ التالية:

١- الصيغة الأولى بدون معاملات.

٢- الصيغة الثانية تستقبل كائن مصدر الربط BindingSource، الذي سيتحكم من خلاله موجه الربط في سجلات مصدر البيانات.

٣- الصيغة الثالثة تستقبل معاملا منطقيا، إذا جعلته False فلن يعرض موجه الربط أزرار التحكم القياسية (أزرار الانتقال وزر الحذف وزر الإضافة).

٤- الصيغة الرابعة تستقبل كائنا من نوع الواجهة IContainer (مثل النموذج)، ليتم عرض شريط موجه الربط عليه.

وإضافة إلى ما ترثه من خصائص الفئة ToolStrip، تمتلك الفئة BindingNavigator الخصائص التالية:

مصدر الربط BindingSource:

تقرأ أو تغير كائن مصدر الربط BindingSource الذي يستخدمه موجه الربط للتحكم في السجلات.

عنصر إضافة جديد AddNewItem:

تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم لإضافة سجل جديد إلى مصدر البيانات.. وفي الوضع الافتراضي يكون هذا العنصر من النوع ToolStripButton.. ويكون زر الإضافة معطلا على شريط موجه الربط، إذا كانت للخاصية BindingSource.AllowNew القيمة False.

لاحظ أنك قد تجد زر الإضافة معطلا في بعض البرامج.. إذا حدثت معك هذه المشكلة، فيمكنك وضع القيمة Nothing في هذه الخاصية من الكود، أو اختيار القيمة (None) من القائمة المنسدلة في نافذة الخصائص.. هذا سيعطل الوظيفة الآلية للزر، لكنه لن يحذفه من على شريط موجه الربط.. لهذا يمكنك نقره مرتين بالفأرة، وكتابة السطر الوحيد التالي في حدث ضغطه:

```
AuthorsBindingNavigator.BindingSource.AddNew( );
```

عنصر الحذف DeleteItem:

تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم لحذف السجل الحالي من مصدر البيانات.. وفي الوضع الافتراضي يكون هذا العنصر من النوع ToolStripButton.. ويكون زر الحذف معطلا على شريط موجه الربط، إذا كانت للخاصية BindingSource.AllowRemove القيمة False.

وإذا وجدت زر الحذف معطلا في بعض الحالات، فضع القيمة Nothing في هذه الخاصية من الكود، أو (None) من القائمة المنسدلة في نافذة الخصائص، ثم انقر مرتين بالفأرة فوق زر الحذف الموجود على شريط موجه الربط ، واكتب السطر الوحيد التالي في حدث ضغطه:

```
AuthorsBindingNavigator.BindingSource.RemoveCurrent( );
```

عنصر التحرك إلى الأول MoveFirstItem:

تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم للانتقال إلى أول سجل في مصدر البيانات.. وفي الوضع الافتراضي يكون هذا العنصر من النوع ToolStripButton.

عنصر التحرك إلى الأخير MoveLastItem:

تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم للانتقال إلى آخر سجل في مصدر البيانات.. وفي الوضع الافتراضي يكون هذا العنصر من النوع ToolStripButton.

عنصر التحرك إلى التالي MoveNextItem:

تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم للانتقال إلى السجل التالي في مصدر البيانات.. وفي الوضع الافتراضي يكون هذا العنصر زر رف الأدوات ToolStripButton.

عنصر التحرك إلى السابق MovePreviousItem:

تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم للانتقال إلى السجل السابق في مصدر البيانات.. وفي الوضع الافتراضي يكون هذا العنصر من النوع ToolStripButton.

عنصر الموضع PositionItem:

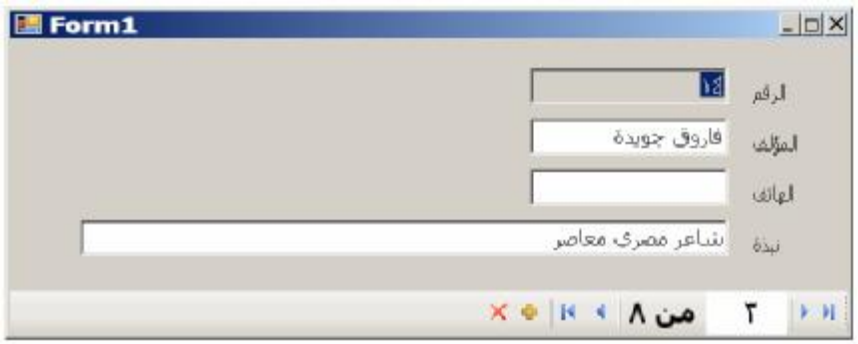
تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم لعرض رقم السجل المعروض حاليا.. وفي الوضع الافتراضي يستخدم مربع نص رف الأدوات ToolStripTextBox لهذا الغرض، وذلك للسماح للمستخدم بكتابة رقم السجل الذي يريده وضغط زر الإدخال Enter للانتقال إليه مباشرة.. ويعرض مربع النص موضع السجل الحالي باستخدام الخاصية BindingSource.Position.

عنصر العد CountItem:

تقرأ أو تغير عنصر رف الأدوات ToolStripItem المستخدم لعرض العدد الكلي للسجلات في مصدر البيانات.. وفي الوضع الافتراضي تستخدم لافتة رف أدوات ToolStripLabel لهذا الغرض، وهي تعرض قيمة الخاصية BindingSource.Count.

تنسيق عنصر العد CountItemFormat:

تستقبل نصا يمثل الصيغة التي سيستخدمها عنصر العد CountItem لعرض عدد السجلات.. وفي الوضع الافتراضي تكون قيمة هذه الخاصية "{0} of".. وفي المشاريع العربية عليك تغييرها إلى صيغة مناسبة، مثل "من {٠}" ليبدو الشريط كما في الصورة:



وتمتلك الفئة BindingNavigator الوسائل التالية:

إضافة العناصر القياسية AddStandardItems:

تضيف أزرار الانتقال والحذف والإضافة ومربع نص الموضع ولافتة عدد السجلات إلى شريط موجه الربط.. هذا مفيد إذا أردت إنشاء موجه ربط من الكود وليس في وقت لتصميم.. وفي حالة وجود العناصر القياسية بالفعل على الشريط، فإن هذه الوسيلة لا تحذفها، بل تضيف نسخة أخرى منها، لكنها لا تصير هي العناصر الفعالة.

إجازة Validate:

تجعل النموذج يفحص قيم الأدوات الموجودة عليه، وتعيد True إذا كانت بياناتها صحيحة.

كما تمتلك الفئة BindingNavigator الحدث التالي:

إنعاش العناصر RefreshItems:

ينطلق إذا حدثت تغيرات في مصدر البيانات، تستدعي تحديث أزرار ولافتات موجه الربط.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

ملحق: ٢

أنواع بيانات سيكويل المدارة
Managed SQL Data Types

يمنحك النطاق System.Data.SqlTypes عددا من السجلات Structures والفئات Classes التي تمثل أنواع البيانات الخاصة بخادم سيكيول.. هذا يسهل عليك إرسال واستقبال البيانات عند التعامل مع قواعد بيانات سيكيول.

وفيما يلي، نتعرف على هذه السجلات.. لا تنسَ إضافة الجملة التالية أعلى صفحة الكود، قبل تجربة أي مثال في هذا الفصل:

using System.Data.SqlTypes;

وستجد أمثلة على بعض هذه الأنواع في المشروع SqlDataTypes.

لاحظ أن جميع الأنواع التي سنتعرف عليها تمثل الواجهة INullable، لهذا فهي تستطيع أن تحتوي القيمة Null، مما يعني أن الخانة التي يتعامل معها الكائن في قاعدة البيانات فارغة.. كما يعني أن جميع هذه الأنواع تمتلك الخاصية IsNull، التي تعيد true إذا كان الكائن فارغا (يحتوي على Null)، وفي هذه الحالة يجب ألا تحاول قراءة قيمة هذا الكائن، وإلا حدث خطأ في البرنامج.

سجل القيمة المنطقية SqlBoolean Structure

يستطيع هذا السجل أن يحتوي على قيمة منطقية: true أو false.

ولحدث إنشاء هذا السجل ثلاث صيغ:

١. الأولى بدون معاملات، وهي تنشئ نسخة قيمتها Null.

٢. والثانية تستقبل قيمة منطقية Boolean لوضعها في السجل.. مثال:

```
SqlBoolean Sb = new SqlBoolean(true);
```

٣. والثالثة تستقبل عددا صحيحا Integer لوضعه في هذا السجل، حيث يعتبر الصفر false وأي عدد آخر true.

ويمتلك هذا السجل الخصائص التالية:

خطأ false:

تعيد سجلا منطقيا SqlBoolean قيمته false.

صواب true:

تعيد سجلا منطقيا SqlBoolean قيمته true.

عدم Null:

تعيد سجلا منطقيا SqlBoolean قيمته Null.

صفر Zero:

تعيد سجلا منطقيا SqlBoolean قيمته ٠ (هذا يعني أن قيمته false).

واحد One:

تعيد سجلا منطقيا SqlBoolean قيمته ١ (هذا يعني أن قيمته true).

هل هو خطأ IsFalse:

تعيد true إذا كانت قيمة السجل الحالي false.

هل هو صواب IsTrue:

تعيد true إذا كانت قيمة السجل الحالي true.

القيمة Value:

تعيد قيمة منطقية Boolean تعبر عن قيمة السجل الحالي.. وتسبب هذه الخاصية خطأ إذا كان السجل منعدما، لهذا عليك فحصه أولا باستخدام الخاصية IsNull قبل استخدام هذه الخاصية.

القيمة الرقمية ByteValue:

تعيد وحدة ثنائية Byte تعبر عن قيمة السجل الحالي (صفر للخطأ و ١ للصواب).. وتسبب هذه الخاصية خطأ إذا كان السجل منعدما، لهذا عليك فحصه أولا باستخدام الخاصية IsNull قبل استخدام هذه الخاصية.. مثال:

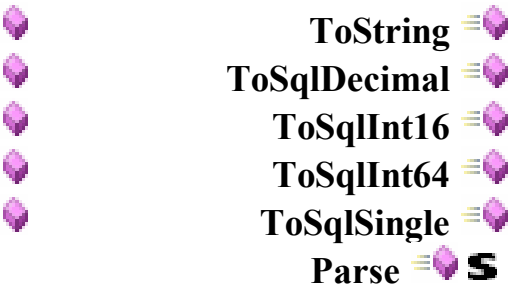
```
if (!Sb.IsNull)
{
   MessageBox.Show(Sb.ByteValue.ToString ( )); // 1
   MessageBox.Show(Sb.Value.ToString ( )); // true
}
```

ويمتلك هذا السجل المعاملات Operators اللازمة لإجراء العمليات الحسابية والمنطقية اللازمة.. كما أنه يمتلك وسائل مشتركة Static Methods لأداء نفس وظائف هذه المعاملات.. والجدول التالي يلخص المعاملات المتاحة والدوال المناظرة لها:

المعامل	الوسيلة
&	And
|	Or
^	Xor
!	OnesComplement (المعكوس الثنائي)
==	Equals
!=	NotEquals
>	GreaterThan

GreaterThanOrEquals	>=
LessThan	<
LessThanOrEquals	<=

كما يمتلك هذا السجل عدة وسائل للتحويل، مثل:



انظر المثال التالي:

```
var Sb = SqlBoolean.Parse("false");
SqlByte B = Sb.ToSqlByte( );
MessageBox.Show(B.ToString( )); // 0
```

مع ملاحظة أنك لا تحتاج إلى استخدام هذه الوسائل، لأن هذا السجل يعرف أيضا معاملات التحويل الضمني Implicit Operators ومعاملات التحويل الصريح Explicit Operators اللازمة لتحويل القيم الأخرى إلى هذا السجل، أو تحويل هذا السجل إلى قيم أخرى مباشرة.. مثال:

```
SqlBoolean Sb1 = true;
SqlByte B1 =(SqlByte) Sb1;
MessageBox.Show(B1.ToString( )); // 1
```

ملحوظة:
كل الأنواع التي سنشرحها فيما بعد مزودة بالمعاملات الحسابية (الطرح والجمع والضرب والقسمة وباقي القسمة) والمعاملات المنطقية (& و | و ^ و !) ومعاملات التحويل الضمني والصريح.. ولا يحتوي كل نوع إلا على المعاملات التي تناسب القيم الموجودة فيه (النصوص مثلا لا تملك معاملات منطقية)، لهذا لن نكرر ذكر هذا في باقي الأنواع، إلا إذا كان هناك معامل يقوم بوظيفة مختلفة عن المألوف.

سجل الوحدة الثنائية SqlByte Structure

هذا السجل يحفظ وحدة ثنائية بدون إشارة، أي أنه يتعامل مع الأعداد من ٠ إلى ٢٥٥.
ويستقبل حدث إنشاء هذا السجل وحدة ثنائية Byte لنسخ قيمتها إليه.. مثال:

```
SqlByte B = new SqlByte(5);
```

ويمتلك هذا السجل الخصائص التالية:

أقل قيمة MinValue:

تعيد أقل قيمة يمكن وضعها في السجل.

أقصى قيمة MaxValue:

تعيد أكبر قيمة يمكن وضعها في السجل.

صفر Zero:

تعيد نسخة من السجل SqlByte تحتوي على القيمة صفر.

العدم Null:

تعيد نسخة من السجل SqlByte لا تحتوي على أي قيمة.

القيمة Value:

تعيد وحدة ثنائية Byte تحمل القيمة المحفوظة في السجل.. وتسبب هذه الخاصية خطأ في البرنامج إذا كان السجل منعدما، لذا عليك أن تستخدم الخاصية IsNull أولا للتأكد من وجود قيمة في السجل.

لاحظ أن ما ينطبق على السجل SqlByte ينطبق على السجلات الرقمية الأخــرى، فهــي تمتلك نفس الخصائص، وتستطيع حفظ قيمة أو Null، والاختلاف الوحيد هو نــوع القيمــة المحفوظة.. لذا فلا داعي لتكرار نفس الكلام مع الأنواع التالية، فأنت تستطيع فهمها بمجرد النظر:


سجل الأعداد القصيرة SqlInt16 Structure


سجل الأعداد الصحيحة SqlInt32 Structure


سجل الأعداد الطويلة SqlInt64 Structure


سجل الأعداد المفردة SqlSingle Structure


سجل النقود SqlMoney Structure


سجل الأعداد المزدوجة SqlDouble Structure


سجل التاريخ والوقت SqlDateTime Structure


سجل المعرف المتفرد العام SqlGuid Structure

سجل الأعداد العشرية SqlDecimal Structure

يحفظ هذا السجل الأعداد العشرية، بنفس طريقة السجل Decimal، كما أن حـــدث إنشـــاء هذا السجل يمتلك صيغا شبيهة بحدث إنشاء السجل Decimal، التي يمكنك مراجعتها فـــي كتاب برمجة إطار العمل.

ويزيد هذا السجل على سجلات الأنواع العددية الأخرى بالخصائص التالية:

أقصى دقة MaxPrecision:

يعيد أقصى عدد من الخانات الصحيحة والعشرية يمكن استخدامه في العدد.

أقصى مقياس MaxScale:

يعيد أقصى عدد من الخانات العشرية يمكن استخدامه في العدد.

البيانات الثنائية BinData:

تعيد مصفوفة ثنائية Byte Array تحتوي على التمثيل الثنائي للعدد العشري.

البيانات Data:

تعيد مصفوفة أعداد صحيحة Integer Array تحتوي على التمثيـــل الثنـــائي للعـــدد العشري.

هل هو موجب IsPositive:

تعيد true إذا كان العدد العشري موجبا.

الدقة Precision:

تعيد عدد الخانات الصحيحة والعشرية في العدد الحالي.

المقياس Scale:

تعيد عدد الخانات العشرية في العدد الحالي.

فئة الحروف SqlChars Class

هذه الفئة تتعامل مع مصفوفة حروف، بحيث يمكن استخدامها للتعامل مع أنواع سيكيول التالية: varchar, nvarchar, char, nchar, text, ntext، مع ملاحظة أن أقصى عدد من الحروف يمكن وضعه في هذه الفئة هو أقصى قيمة للعدد الصحيح (أي حوالي ٢ مليار حرف).

ولحدث إنشاء هذه الفئة ثلاث صيغ:

١. الأولى بدون معاملات، وهي تنشئ نسخة قيمتها Null.
٢. والثانية تستقبل مصفوفة حروف Char Array.
٣. والثالثة تستقبل نسخة من السجل SqlString لأخذ الحروف من النص الموجود فيها.

وتمتلك هذه الفئة الخصائص التالية:

العدم Null:

تعيد نسخة فارغة من الفئة SqlChars.

المفهرس Indexer:

يقرأ أو يغير الحرف الموجود في الموضع المرسل كمعامل.

الطول Length:

تعيد عدد الحروف الموجودة حاليا في الكائن.

أقصى طول MaxLength:

تعيد أقصى عدد من الحروف يمكن وضعه في الكائن.. هذا العدد يساوي صفرا مبدئيا، لكنه يساوي طول النص عند وضع نص في الكائن.. وعند تقصير طول الكائن، يظل أقصى طول كما هو دون أن ينقص.

التخزين Storage:

تعيد إحدى قيم المرقم StorageState التي توضح نوع المخزن الذي يتم فيه حفظ الحروف داخل الكائن، وهذه القيم هي:

Buffer	تحفظ الحروف في مصفوفة داخلية.
UnmanagedBuffer	تحفظ الحروف في الذاكرة باستخدام مؤشرات غير مدارة بإطار العمل.
Stream	تحفظ الحروف في مجرى بيانات Stream.

المخزن الوسيط Buffer:

تعيد مصفوفة الحروف التي يتعامل معها الكائن داخليا.. لاحظ أن أي تغيير في هذه المصفوفة يؤثر على محتويات الكائن.

القيمة Value:

تعيد مصفوفة حروف بها نسخة من محتويات الكائن.. لاحظ أن أي تغيير في هذه المصفوفة لا يؤثر على محتويات الكائن، على عكس المصفوفة التي تعيدها الخاصية Buffer.

كما تمتلك فئة الحروف الوسائل التالية:

تغيير الطول SetLength:

أرسل إلى هذه الخاصية عدد الحروف الذي تريد وجودها في الكائن.. لاحظ أنك لو أرسلت عددا أكبر من أقصى طول MaxLength فسيحدث خطأ.. هذا معناه أنك تستطيع تصغير محتويات الكائن، حيث سيتم حذف الحروف الزائدة عن الطول الجديد، لكن ستظل المصفوفة الداخلية تحجز الخانات التي تم الاستغناء عنها، لهذا تستطيع أن تكبر الطول مرة أخرى، بشرط عدم تجاوز الطول الأقصى.

ولو كان الكائن يتعامل مع مخزن وسيط غير مـــدار Unmanaged Buffer فســـيتم تحويله إلى مخزن مدار Managed Buffer بعد تنفيذ هذه الوسيلة.

وضع العدم SetNull:

تمحو محتويات الكائن الحالي وتجعل طوله صفرا.

قراءة Read:

تنسخ عددا من الحروف من الكائن الحالي إلى مصفوفة، ولها المعاملات التالية:

- موضع بداية القراءة من الكائن.
- مصفوفة الحروف التي سيتم النسخ إليها.
- موضع بداية الكتابة في المصفوفة.
- عدد الحروف المنسوخة.

وتعيد هذه الوسيلة عدد الحروف التي تم نسخها.. الحكمة في هذا أن عـــدد الحـــروف المنسوخة قد يكون أقل من المطلوب، إذا لم يكن الكائن يحتوي على العدد المطلـــوب من الحروف.

والمثال التالي ينسخ ٥ حروف من الكائن بدءا من الحرف الرابع:

```
SqlChars Sc = new SqlChars("This is a test");
char[] C = new char[5];
Sc.Read(3, C, 0, 5);
MessageBox.Show("'" + new String (C) + "'");
```

كتابة Write:

تنسخ عددا من الحروف من مصفوفة إلى الكائن الحالي بدءا من موضع معين.. ولهـــا نفس معاملات الوسيلة السابقة.

لاحظ أنك تستطيع كتابة حروف في موضع تال لآخر حرف في الكـــائن، بشـــرط ألا تتجاوز الطول الأقصى للكائن MaxLength.

والمثال التالي ينسخ كل حروف المصفوفة إلى الكائن بدءا من الحرف الرابع:

```
SqlChars Sc = new SqlChars("This is a test");
char[] X = { 'A', 'B', 'C', 'D', 'E' };
Sc.Write(3, X, 0, 5);
MessageBox.Show(new string(Sc.Value));
```

التحويل إلى نص سيكويل ToSqlString:

تعيد نسخة من السجل SqlString تحتوي على نص مكـــون مـــن حـــروف الكـــائن الحالي.

سجل النص SqlString Structure

يختلف نص سيكويل في طريقة تمثيله الداخلية، عن فئة النص String Class العادية.. فعلى سبيل المثال: يأخذ النص العادي معلومات الثقافة من اللغة الافتراضية المعرفة على جهاز المستخدم، بينما لا يفعل نص سيكويل هذا، فلو لم تمده بمعرف الثقافة، فإنه يستخدم مقاييس داخلية خاصة به لمقارنة النصوص.. ولو حاولت مقارنة نسختين من نص سيكويل لكل منهما معرف ثقافة LCID مختلف عن الآخر، فإن خطأ سيحدث في البرنامج بسبب عدم قدرته على إجراء عملية المقارنة.

(أنصح بمراجعة فصل العولمة Globalization في مرجع برمجة إطار العمل).

ولحدث إنشاء هذا السجل الصيغ التالية:

١. الأولى بدون معاملات، وهي تنشئ نسخة قيمتها Null.
٢. والثانية تستقبل نصا String لنسخه إلى السجل.
٣. والثالثة تزيد على الصيغة السابقة بمعامل ثان يستقبل معرف الثقافة LCID الذي تريد استخدامه عند مقارنة النص بأي نص آخر.. مثال:

```
SqlString Ss = new SqlString("محمد", System.Globalization.
        CultureInfo.CurrentCulture.LCID);
```

٤. والرابعة تزيد على الصيغة السابقة بمعامل ثالث، يستقبل إحدى قيم المرقم SqlCompareOptions التالية:

None	تتم المقارنة بالخيارات الافتراضية للثقافة التي يرتبط بها السجل الحالي.
IgnoreCase	تتجاهل المقارنة حالة الأحرف.
IgnoreNonSpace	تتجاهل المقارنة كل الرموز التي لا تعتبر فواصل بين الحروف، مثل علامات التشكيل في اللغة العربية.. هذا مفيد عند البحث عن كلمة مع تجاهل التشكيل.

IgnoreKanaType	تتجاهل المقارنة الرموز الصوتية في اللغة اليابانية.
IgnoreWidth	تتجاهل المقارنة إن كانت الحروف اليابانية مكتوبة بالعرض الكامل أم بنصف العرض.
BinarySort	يتم ترتيب الحروف تبعا لقيمها الرقمية في ترميز ASCII أو Unicode، وليس تبعا للترتيب الهجائي.
BinarySort2	يتم ترتيب الحروف ثنائيا، باستخدام قيمها في الترميز.

لاحظ أنك تستطيع دمج أكثر من قيمة معا باستخدام المعامل |.

٥. والخامسة تستقبل معرفة الثقافة LCID وخيارات المقارنة SqlCompareOptions ومصفوفة ثنائية Byte Array تحتوي على التمثيل الثنائي للنص.
٦. والسادسة تزيد على الصيغة السابقة بمعامل رابع، عليك جعله true إذا كان النص ممثلا بالترميز الموسع Unicode.
٧. والسابعة تزيد على الصيغة الخامسة بمعامل رابع يستقبل موضع بداية القراءة من المصفوفة، ومعامل خامس يستقبل عدد الحروف التي تريد قراءتها منها.
٨. والثامنة تزيد على الصيغة السابقة بمعامل رابع، عليك جعله true إذا كان النص ممثلا بالترميز الموسع Unicode.

ويمتلك سجل النص الخصائص التالية:

ترتيب ثنائي BinarySort:

تعمل كثابت يعني أن ترتيب الحروف يتم تبعا لقيمها الرقمية في ترميز ASCII وليس تبعا للترتيب الهجائي.

ترتيب ثنائي BinarySort2:

تعمل كثابت يعني أن ترتيب الحروف يتم تبعا لقيمها الرقمية في الترميز.

تجاهل الحالة IgnoreCase:

تعمل كثابت يعني أن مقارنة الحروف تتجاهل حالتها (صغيرة أم كبيرة).

تجاهل الحروف غير الفاصلة IgnoreNonSpace:

تعمل كثابت يعني أن المقارنة تتجاهل كل الرموز التي لا تعتبر فواصل بين الحروف، مثل علامات التشكيل في اللغة العربية.

تجاهل نوع الكانا IgnoreKanaType:

تعمل كثابت يعني أن مقارنة الحروف تتجاهل الرموز الصوتية في اللغة اليابانية.

تجاهل العرض IgnoreWidth:

تعمل كثابت يعني أن مقارنة الحروف تتجاهل إن كانت الحروف اليابانية مكتوبة بالعرض الكامل أم بنصف العرض.

العدم Null:

تعيد نسخة فارغة من السجل SqlString.

معلومات الثقافة CultureInfo:

تعيد كائن معلومات الثقافة CultureInfo الذي يستخدمه السجل الحالي في تنسيق وترتيب ومقارنة النصوص.

المعرف المحلي للثقافة LCID:

تعيد عددا صحيحا يستخدم كمعرف للثقافة التي يتعامل معها السجل الحالي.

معلومات المقارنة CompareInfo:

تعيد كائن معلومات المقارنة CompareInfo الذي يستخدمه السجل الحالي في مقارنة النصوص.

خيارات المقارنة SqlCompareOptions:

تعيد إحدى قيم المرقم SqlCompareOptions التي توضح الخيارات المستخدمة لمقارنة النصوص.

القيمة Value:

تعيد String يمثل النص الموجود في السجل الحالي، أو تسبب خطأ إذا كان السجل فارغا.

ويمتلك هذا السجل الوسائل الهامة التالية:

تشبيك Concat:

تدمج نسختين من نص سيكيول في نص واحد، وتعيده كسجل جديد.. مثال:

```
var LCID = System.Globalization.
        CultureInfo.CurrentCulture.LCID;
SqlString Ss1 = new SqlString("محمد", LCID);
SqlString Ss2 = new SqlString(" محمود", LCID);
var Ss3 = SqlString.Concat(Ss1, Ss2);
MessageBox.Show(Ss3.Value); // محمد محمود
```

ويمكنك إنجاز نفس المهمة باستخدام الوسيلة Add كالتالي:

```
var Ss3 = SqlString.Add(Ss1, Ss2);
```

أو باستخدام معامل الجمع كالتالي:

```
var Ss3 = Ss1 + Ss2;
```

خيـــــارات المقارنـــــة مـــــن خيـــــارات مقارنـــــة ســـــيكويل CompareOptionsFromSqlCompareOptions:

أرسل إلى هذه الوسيلة إحدى قيم المرقم SqlCompareOptions، لتعيد إليك القيمـــة المناظرة لها في المرقم CompareOptions الذي تعرفنا عليه في كتاب برمجة إطار العمل.

نسخ Clone:

تعيد نسخة جديدة من السجل SqlString بها نفس قيمة السجل الحالي.

قراءة البيانات غير الموسعة GetNonUnicodeBytes:

تعيد مصفوفة ثنائية Bytes Array، تحتوي على تمثيل النص الحـــالي فـــي ترميـــز ASCII.

قراءة البيانات الموسعة GetUnicodeBytes:

تعيد مصفوفة ثنائية Bytes Array، تحتوي على تمثيل النص الحـــالي فـــي ترميـــز Unicode.

سجل البيانات الثنائية SqlBinary Structure

يمثل هذا السجل مصفوفة من الوحدات الثنائية Byte Array.. ويمكنك ملء هذا السجل بالبيانات بإرسال مصفوفة ثنائية إلى حدث إنشائه.. مثال:

```
SqlBinary Sb1 = new SqlBinary(new byte[] {100, 220, 3});
```

ونظرا لأن هذا السجل يعرف معامل التحويل الضمني Implicit Operator، فيمكنك وضع مصفوفة ثنائية في هذا السجل مباشرة:

```
SqlBinary Sb1 = new byte[] {100, 220, 3};
```

ويمتلك هذا السجل الخصائص التالية:

القيمة المنعدمة Null:

تعيد سجلا فارغا.. مثال:

```
var Sb3 = SqlBinary.Null;
```

الطوال Length:

تعيد عدد الوحدات الثنائية الموجودة في السجل.. مثال:

```
MessageBox.Show(Sb1.Length.ToString( )); // 3
```

المفهرس Indexer:

يعيد الوحدة الثنائية Byte الموجودة في موضع معين في السجل.. مثال:

```
MessageBox.Show(Sb1[1].ToString( )); // 220
```

المؤسف أن هذا المفهرس للقراءة فقط، لذا فلا يمكنك استخدامه لتغيير العنصر الموجود في موضع معين من المصفوفة.. ولست أدرى ما الحكمة من هذا!

القيمة Value:

تعيد مصفوفة ثنائية Byte Array تحتوي على القيم الموجودة في السجل الحالي.

كما يمتلك هذا السجل عددا من الوسائل المشتركة Shared، وهي تقوم بنفس وظائف المعاملات Operators المعرفة لهذا السجل.. ويهمنا هنا أن نشير إلى بعضها لأن طريقة عملها مختلفة نوعا:

إضافة Add:

تشبيك Concat:

المعامل + :

تقوم بتشبيك سجلين، بدمج المصفوفة الثانية بعد نهاية المصفوفة الأولى، وتعيد المصفوفة الجديدة في سجل جديد.. مثال:

```
var Sb = SqlBinary.Add(Sb1, Sb2);
```

أو:

```
var Sb = SqlBinary.Concat(Sb1, Sb2);
```

أو باختصار:

```
var Sb = Sb1 + Sb2;
```

بعد تنفيذ هذا المثال، سيحتوي سجل البيانات الثنائية Sb على القيم التالية:

١٠٠، ٢٢٠، ٣، ١، ٠، ٢.

يساوي Equals:

المعامل == :

تعيد true إذا كان السجلان لهما نفس الطول ويحتويان على نفس القيم بنفس الترتيب.. مثال:

```
MessageBox.Show(SqlBinary.Equals(Sb1,Sb2).ToString());
                                                    // false
```

أو باختصار:

```
MessageBox.Show((Sb1 == Sb2).ToString( )); // false
```

لاحظ أن القيمة العائدة من هذه الوسيلة هي من النوع SqlBollean، حيث تكون نتيجة المقارنة Null إذا كان أي من السجلين منعدما.

فئة الوحدات الثنائية SqlBytes Class

هذه الفئة مشابهة بدرجة كبيرة للسجل SqlBinary، إلا أنها تمتلك ميزة إضافية، وهي قدرتها على التعامل مع الوحدات الثنائية Bytes من خلال مجرى بيانات Stream.. هذا يتيح لك قراءة البيانات من ملف FileStream أو من مجرى بيانات الذاكرة MemoryStream أو من خلال الشبكة NetworkStream.

ولحدث إنشاء هذه الفئة الصيغ التالية:

١. الصيغة الأولى بدون معاملات.
٢. والثانية تستقبل بياناتها من مصفوفة ثنائية Byte Array.
٣. والثالثة تستقبل بياناتها من سجل بيانات ثنائية SqlBinary.
٤. والرابعة تستقبل بياناتها من مجرى بيانات Stream.

وتشبه هذه الفئة أيضا الفئة SqlChars في كل خصائصها ووسائلها، ما عدا أن التعامل هنا يكون مع مصفوفة ثنائية Byte Array بدلا من مصفوفة حروف Char Array.. لهذا لا نحتاج إلى إعادة شرح هذه الخصائص والوسائل:

Null العدم	**IsNull هل هو منعدم**
Indexer المفهرس	**Length الطول**
MaxLength أقصى طول	**Storage التخزين**
Buffer المخزن الوسيط	**Value القيمة**
SetLength تغيير الطول	**SetNull وضع عدم**
Read قراءة	**Write كتابة**

الجديد فقط، هو الخاصية والوسيلة التاليتان:

مجرى البيانات Stream:

تقرأ أو تغير مجرى البيانات الذي يتعامل معه الكائن الحالي.. ويؤدي استخدام هذه الخاصية إلى تحميل كل البيانات من مجرى البيانات إلى الذاكرة، ولو كانت هذه البيانات ضخمة للغاية، فقد تؤدي إلى استهلاك مساحة الذاكرة وحدوث خطأ من النوع OutOfMemoryException.

التحويل إلى بيانات ثنائية ToSqlBinary:

تعيد سجل بيانات ثنائية SqlBinary يحتوي على الوحدات الثنائية Bytes الموجودة في الكائن الحالي.

فئة XML
SqlXml Class

تحفظ هذه الفئة وثيقة XML، وهي تعتمد داخليا على "قارئ بيانات XML" XmlReader، لهذا يجب مراعاة أن يكون تنسيق بيانات XML الذي تضعها في هذه الفئة موافقا للمعايير التي تقبلها الفئة XmlReader.. وسنتعرف على كيفية التعامل مع بيانات XML وفئاتها في كتاب مستقل بإذن الله.

ولحدث إنشاء هذه الفئة ثلاث صيغ:

١- الأولى بدون معاملات.

٢- والثانية تستقبل البيانات من مجرى بيانات Stream.

٣- والثالثة تستقبل البيانات من "قارئ XML" XmlReader.

وتمتلك هذه الفئة الخصائص التالية:

العدم Null:

تعيد نسخة من الفئة SqlXml لا تحتوي على أي قيمة.

القيمة Value:

تعيد نصا String يحتوي على وثيقة XML المحفوظة في الكائن الحالي.

كما تمتلك هذه الفئة الوسيلة التالية:

إنشاء قارئ CreateReader:

تعيد "قارئ XML" XmlReader لاستخدامه في قراءة محتويات الكائن الحالي.

ملحوظة:

كل أنواع بيانات سيكويل السابقة تدعم التعامل مع XML، من خلال:

- تمثيل الواجهة IXmlSerializable لحفظ محتويات الكائن في وثيقة XML وقراءتها منها في أي وقت.
- امتلاك وسيلة مشتركة Shared Method اسمها GetXsdType، تستقبل "نوع مخطط XML" XmlSchemaSet، وتعيد نسخة من الفئة XmlQualifiedName تحتوي على الاسم الكامل لنوع XML المناظر.

حفظ الملفات خارج قاعدة البيانات:

يقدم لك سيكيول سيرفر ٢٠٠٨ إمكانية رائعة، وهي قدرتك على حفظ البيانات الثنائية الضخمة BLOB (تكون في الغالب أكبر من ١ ميجا بايت) التي ترسلها إلى عمود من النوع varbinary(MAX) في ملف خاص مستقل عن ملف قاعدة البيانات، يتم حفظه في مجلد خاص على الخادم.. هذا يحقق لك الفوائد التالية:

١- يضمن عدم تضخم حجم ملف قاعدة البيانات بصورة كبيرة.

٢- يقلل من الزمن اللازم لقراءة هذه البيانات.

٣- يستطيع النوع varbinary(MAX) حفظ بيانات حجمها تقريبا ٢ جيجا بايت في قاعدة البيانات، بينما عند استخدام ملفات خارجية لا يكون هناك اي حد لجم الملف، إلا مقدار المساحة المتوفرة على القرص الصلب!

٤- قدرتك على التعامل مع هذه الملفات من خلال استعلامات قاعدة البيانات، أو التعامل معها مباشرة من خلال نظام ملفات الويندوز Windows File System.

٥- تقدم لك دوت ت ٢٠١٠ فئة خاصة للتعامل مع البيانات المحفوظة خارج قاعدة البيانات، وهي الفئة SqlFileStream التي سنتعرف عليها لاحقا.

وهناك أربع خطوات عليك اتباعها، حتى تستطيع حفظ قيم الأعمدة الضخمة في ملفات مستقلة.. هذه الخطوات هي:

١- تفعيل استخدام مجرى البيانات FILESTREAM في خدمات الويندوز:

يتم هذا كما يلي:

- من قائمة البرامج Programs Menu، اضغط:

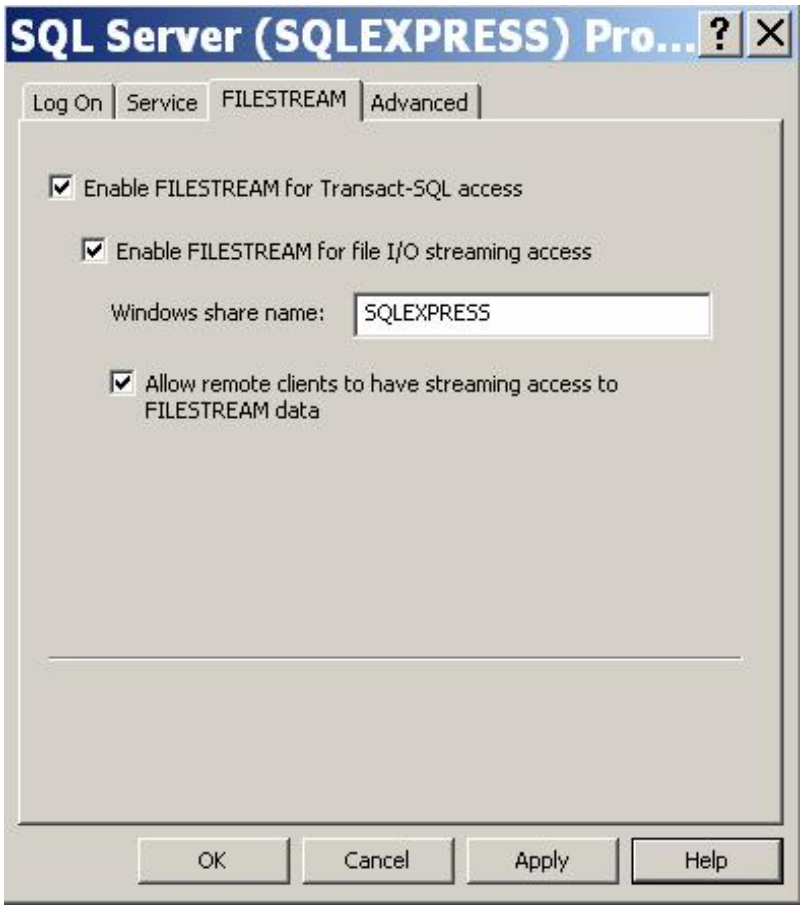
Microsoft SQL Server 2008\Configuration Tools\
SQL Server Configuration Manager

- في الشجرة اليسرى في نافذة تهيئة خادم سيكويل، انقر العنصر SQL Server Services مرتين بالفأرة.
- في القائمة اليمنى، حدد اسم خادم سيكويل الذي تتعامل معه.. في حالتنا هذه سيكون SQL Server(SQLEXPRESS)، وانقره مرتين بالفأرة لعرض خصائصه.

- في نافذة الخصائص، اضغط الشريط العلــوي Tab المســمى FILESTREAM لعرض صفحة خصائصه.
- ضع علامة الاختيار أمام الاختيار:

Enable FILESTREAM for Transact-SQL access

هذا سيتيح لك قراءة وكتابة الملفات الخارجية من خلال الاستعلامات.



- إذا وضعت علامة الاختيار أمام الاختيار:

Enable FILESTREAM for file I/O streaming access

فسيتيح هذا لك القراءة من الملفات الخارجية من خلال نظــام مشــاركة الملفــات Sharing عبر شبكات الويندوز.. أي أنك ستتطيع التعامل مع الملف مباشرة بدون استعلامات، كأنك تتعامل مع أي ملف عادي على الشبكة، وهو ما سنفعله باستخدام

الفئة SqlFileStream.. ويجب عليك أن تكتب في مربع النص اسم مجلد المشاركة الذي ستقرأ الملف من خلاله.. في الوضع الافتراضي يكون هذا الاسم هو SQLEXPRESS، لكن يمكنك تغييره إلى ما تشاء.

- إذا وضعت علامة الاختيار أمام الاختيار:

Allow remote clients to have streaming access to FILESTREAM

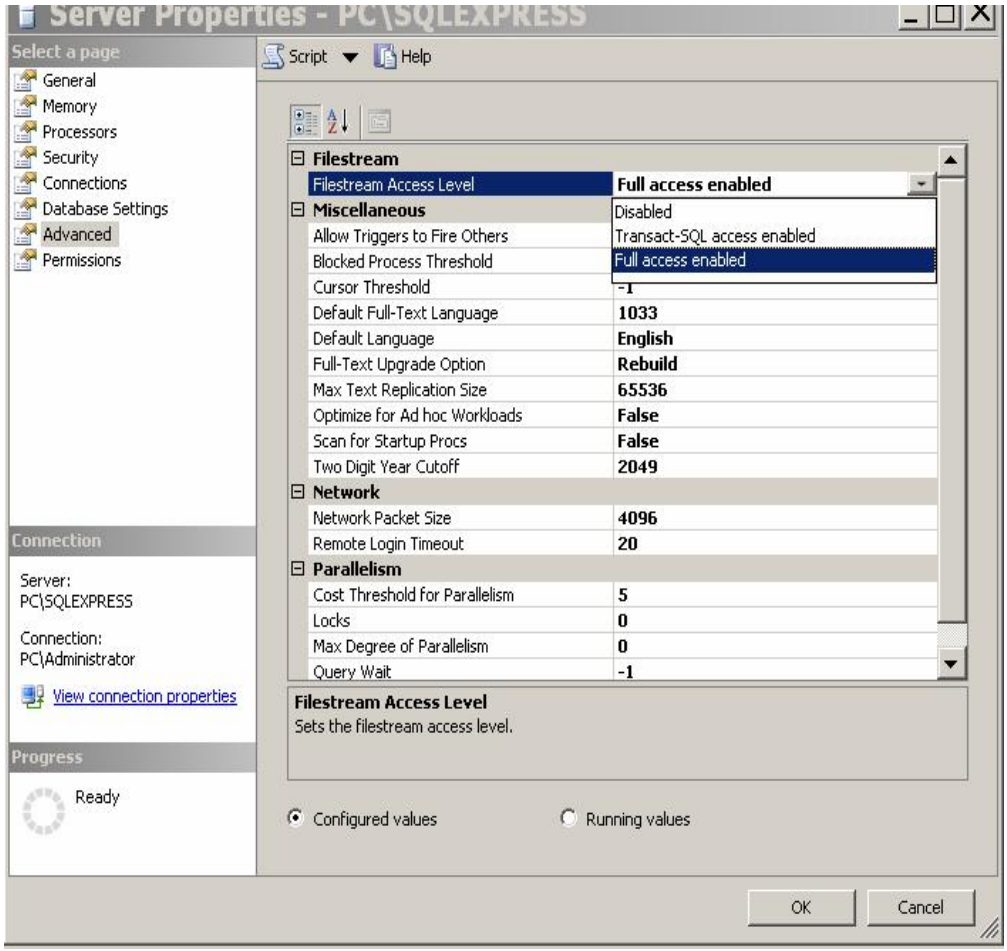
فسيسمح هذا للمستخدمين من خارج الشبكة المحلية Remote Users بقراءة بيانات الملف عبر نظام مشاركة الملفات.

ملحوظة هامة:
يشكو كثير من المستخدمين عند استخدام الفئة SqlFileStream من أن رسالة خطأ تظهر لهم تخبرهم بأن مسار الملف غير موجود على الشبكة.. يعود هذا السبب في الغالب إلى تعطيلهم لإمكانية مشاركة الملفات Sharing الخاصة بالويندوز، لهذا عليك التأكد من تفعيلها قبل تفعيل الاختيار: Enable FILESTREAM for file I/O streaming access ولفعل هذا، اتبع الخطوات التالية: - افتح متصفح الويندوز Windows Explorer ومن القائمة الرئيسية Tools اضغط Folder Options. - في نافذة خيارات المجلدات، اضغط الشريط العلوي View، وتأكد من وضع علامة الاختيار أمام الخيار الأخير في قائمة الخيارات: Use simple file sharing. - اضغط Ok لإغلاق النافذة. وإذا كنت فعّلت خيارات مجرى البيانات FileStream الخاصة بخادم سيكويل قبل تفعيل المشاركة، فقم بتعطيل الخيارات FileStream، وأعد تشغيل خادم سيكويل Restart، ثم أعد تفيعل خيارات مجرى البيانات.. بهذه الطريقة ستضمن وصول برنامجك إلى ملفات المشاركة الخاصة بسيكويل سيرفر، والتي يحفظ فيها الملفات الخارجية.

- اضغط Ok لإغلاق النافذة وحفظ هذه التغييرات.

٢- تفعيل استخدام مجرى البيانات FILESTREAM في خادم سيكيول:

لفعل هذا، افتح مدير سيكويل SQL Server Management Studio، وفي متصفح الكائنات Object Browser حدد العنصر الرئيسي في الشجرة (الذي يحمل اسم خادم سيكويل SQLEXPRESS)، واضغطه بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر Properties.. سيعرض هذا نافذة خصائص خادم سيكيويل.. اضغط العنصر Advanced من القائمة اليسرى، لعرض الخصائص المتقدمة، كما هو موضح في الصورة:

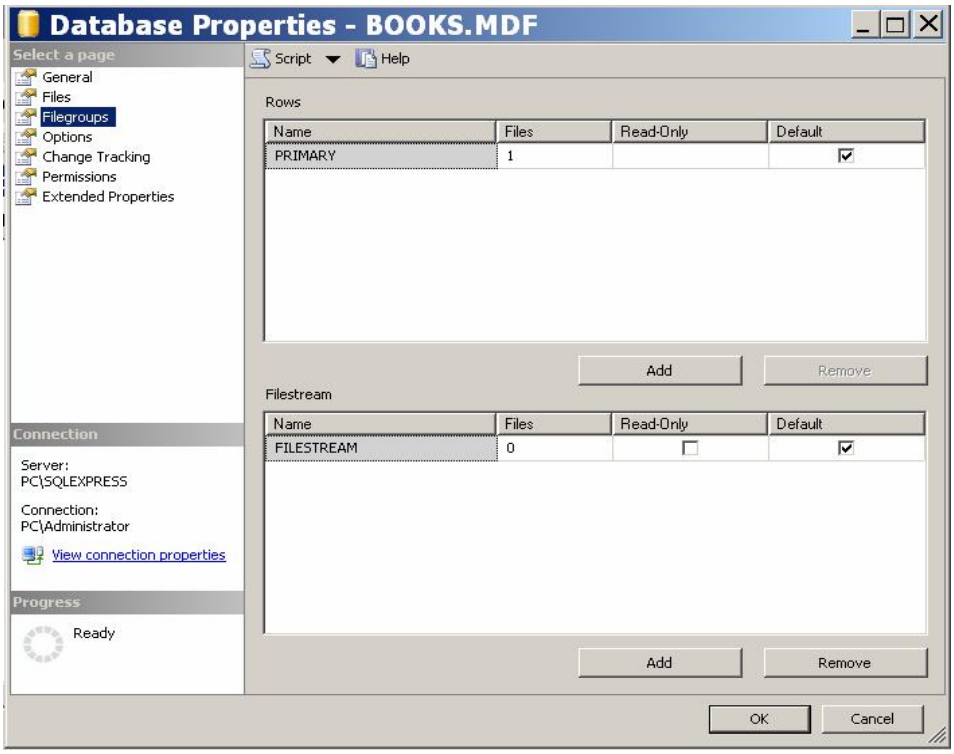


ستجد أول خاصية فيها هي Filestream Access Level، التي توضح كيفية التعامل مع الملفات الخارجية.. ستجد قيمة هذه الخاصية Disabled أي أن التعامل مع الملفات الخارجية ممنوع!.. اضغط القائمة المنسدلة، واختر التعامل الكامل Full Access Enabled.. هذا يسمح بالتعامل مع الملفات مباشرة، أو من خلال الاستعلامات.. أما إذا أردت قصر التعامل مع الملفات من خلال استعلامات SQL فقط، فاختر الاختيار الثاني: Transact-SQL Access Enabled.. اضغط الزر Ok لإغلاق النافذة.. ستظهر لك رسالة تخبرك بأن بعض التغييرات لن تحدث إلا إذا أوقفت خادم سيكويل عن العمل وأعدت تشغيله Restart.. يمكنك فعل هذا من مدير تهيئة خادم سيكويل SQL Server Configuration Manager كما تعلمنا من قبل.

٣- تفعيل استخدام مجرى البيانات FILESTREAM في قاعدة البيانات:

يمكنك فعل هذا عند إنشاء قاعدة البيانات بجمل T-SQL وذلك باستخدام السمة FILESTREAM (هذا خارج نطاق هذا الكتاب).. كما يمكنك تعديل قاعدة البيانات بعد إنشائها لتفعيل هذه الخاصية، وذلك باستخدام الواجهة المرئية لمدير سيكويل SQL Server Management Studio.. افترض أننا نتعامل مع قاعدة بيانات الكتب كمثال.. اتبع الخطوات التالية لتفعيل حفظ البيانات خارجها:

- افتح متصفح الكائنات Object Explorer في مدير سيكويل.
- ألحق Attach قاعدة البيانات C:\Books.mdf إن لم تكن موجودة.
- اضغط قاعدة بيانات الكتب بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط Properties.



- في نافذة الخصائص، اضغط العنصر FileGroups من القائمة اليسرى.. ستجد الجزء الأيمن مقسوما إلى نصفين:

أ. النصف العلوي يعرض مجموعات الملفات العادية (ملفات قاعدة البيانات الأساسية).. ويمكنك إضافة مجموعة أخرى لو أردت تقسيم قاعدة البيانات على أكثر من ملف.

ب. النصف السفلي يتعامل مع الملفات الخارجية Filestream.. هذا هو النصف الذي يعنينا.. اضغط الزر Add لإضافة صف جديد، وفي خانة الاسم اكتب FILESTREAM، وضع علامة الاختيار في الخانة Default كما هو موضح في الصورة.

- من القائمة اليسرى اضغط العنصر Files لعرض ملفات قاعدة البيانات، وفي الجهة اليمنى اضغط الزر Add لإضافة صف جديد.

- اكتب في خانة الاسم BooksFiles.. سيكون هذا هو اسم المجلد الذي سيتم حفظ الملفات التابعة لقاعدة البيانات فيه.
- في الخانة File Type أسدل قائمة العناصر واختر النوع FILESTREAM.. هذا سيجعل الخانة FileGroup تحتوي على القيمة FILESTREAM أيضا.
- استخدم الزر الموجود في الخانة Path لاختيار موضع حفظ المجلد.. هناك شرط إجباري عليك الالتزام به، وهو حتمية اختيار مسار على جزء من القرص الصلب مهيأ بتنسيق NTFS وليس FAT32.
- اضغط Ok لتنفيذ كل ما فعلناه.. ستجد أن مجلدا اسمه BooksFiles قد تم إنشاؤه على المسار الذي اخترته.. ويسمى هذا المجلد بحاوية الملفات Data Container، أو بمجموعة الملفات Filegroup.. لاحظ ما يلي:

أ- لا يمكنك إنشاء مجموعات ملفات متداخلة.

ب- لا يمكنك حفظ مجموعة الملفات على قسم مضغوط Compressed Volume من القرص الصلب.

ج- لا يتم تشفير البيانات المحفوظة في الملفات الخارجية، حتى لو كانت قاعدة البيانات تقوم بتشفير بياناتها.. لهذا لو كان التشفير مهما بالنسبة لك، فأرسل البيانات مشفرة منذ البداية إلى قاعدة البيانات.

د- يمكنك استخدام أوامر التحديث Update والحذف Delete والإدراج Insert الخاصة بلغة SQL للتعامل مع العمود الذي يحفظ بياناته في ملف خارجي، بنفس الطريقة التي تتعامل بها مع أي عمود عادي، حيث ستقوم قاعدة البيانات بإجراء هذه التغييرات على الملف الخارجي دون أن تشغل ذهنك بهذا.

هـ- لا يمكنك استخدام الأمر UPDATE .Write لكتابة البيانات مجزأة في عمود يحفظ بياناته في ملف خارجي.. وبدلا من هذا عليك استخدام الفئة SqlFileStream التي سنتعرف عليها لاحقا، لكتابة أجزاء البيانات في الملف مباشرة.

٤- تفعيل استخدام مجرى البيانات FILESTREAM في العمود:

الآن يمكنك إنشاء عمود في جدول الناشرين (على سبيل المثال) يحفظ بياناته في ملفات خارجية.. لكن للاسف، لا يحتوي مصمم الجدول على خاصية تتيح لنا القيام بهذا بطريقة مرئية، لهذا ليس أمامنا سوى استخدام استعلام T-SQL.. لفعل هذا اتبع ما يلي:

- في متصفح الكائنات، اضغط اسم قاعدة البيانات BOOKS.MDF بزر الفأرة الأيمن، ومن القائمة الموضعية اضغط الأمر New Query.
- في نافذة الاستعلامات، اكتب الاستعلام التالي:

```
ALTER TABLE Publishers
ADD Logo3 VARBINARY(MAX) FILESTREAM  NULL,
RowGuid UNIQUEIDENTIFIER
NOT NULL ROWGUIDCOL
UNIQUE DEFAULT NEWID( )
GO
```

هذا الاستعلام يضيف إلى الجدول Publishers عمودين جديدين:

أ. العمود Logo3، وهو من النوع VARBINARY(MAX) وسيحفظ بياناته في ملف خارجي FILESTREAM، ويقبل القيمة NULL.

ب. والعمود RowGuid، وهو معرف متفرد للجدول UNIQUEIDENTIFIER، ولا يقبل العدم NOT NULL، وهو يعمل كمعرف لصفوف الجدول ROWGUIDCOL.. لاحظ أن وجود هذا العمود إجباري إذا أردت استخدام الملفات الخارجية، لأنه يستخدم في ربط الملف بالصف الذي يحفظ بياناته، لهذا لا يمكن أن يقبل هذا العمود القيمة Null.

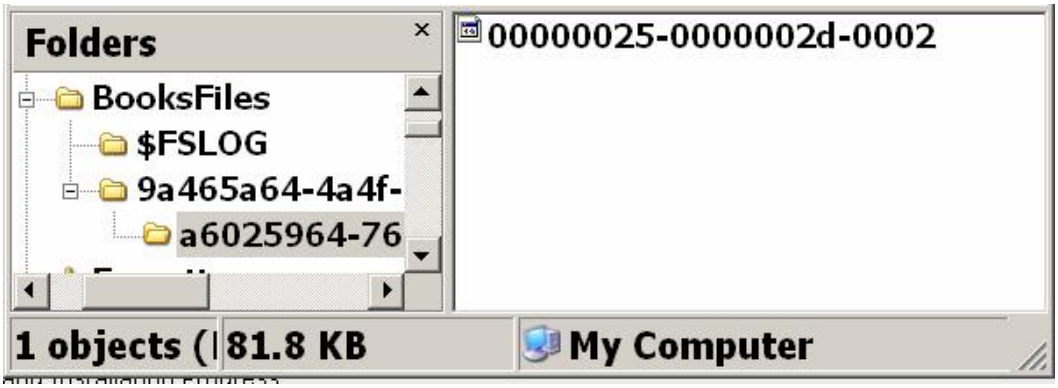
ولتنفيذ هذا الاستعلام، اضغط في أي موضع من النافذة بزر الفأرة الأيمن، واضغط الأمر Execute.. سيظهر في الجزء السفلي من النافذة رسالة تخبرك بنجاح أو فشل التنفيذ.

ويمكنك الاسترشاد بهذا الاستعلام في المواقف المماثلة، فكل ما عليك هو تغيير اسم الجدول Publishers واسم العمود Logo3 ليصير الاستعلام مناسبا لاحتياجاتك.

الآن فقط أنجزنا المهمة كاملة، وبإمكانك أن تحفظ صورة في الحقل Logo3 الخاص بالناشر الأول، باستعلام عادي كالتالي:

```
UPDATE Publishers
SET Logo3 = @Logo
WHERE ID = 1
```

ويمكنك تنفيذ هذا الاستعلام بضغط الزر Write to FileStream في المشروع WriteLargeData، حيث سيظهر لك مربع حوار اختيار صورة.. اختر أي صورة تريدها واضغط OK.. الآن لو فتحت المجلد BooksFiles فستجد ملفا جديدا قد أضيف إليه.. هذا الملف سيحمل اسما عجيبا لضمان عدم تشابه أسماء الملفات، لكنك لو أخذت نسخة منه وغيرت امتدادها إلى امتداد صورة (bmp. مثلا) فستجد أنها نفس الصورة التي اخترتها.



ولو عرضت بيانات جدول الناشرين، فستجد في الخانة التي حفظنا فيها الصورة أرقاما سداسية عشرية، تشير إلى موضع ملف الصورة في المجلد BooksFiles.

ويمكنك قراءة بيانات هذه الصورة باستعلام عادي كالتالي:

```
SELECT ID, Logo3
FROM Publishers
```

حيث يمكنك قراءة بيانات الصورة كاملة أو بطريقة تتابعية Sequential باستخدام قارئ البيانات DataReader كما سنرى لاحقا.. وهذا هو نفس ما يمكنك فعله مع البيانات الثنائية المحفوظة في قاعدة البيانات مثل image أو varbinary(MAX).. ويمكنك ضغط الزر Read FileStream في المشروع ReadLargeData لتجربة قراءة الصورة التى حفظتها خارج قاعدة البيانات، حيث سيتم حفظها على المحرك C:\ بالاسم Logo1.bmp.

وهكذا نكون قد تعاملنا مع الملف الخارجي كأنه جزء من قاعدة البيانات.. يتبقى غذن أن نعرف كيف نتعامل مع هذا الملف مباشرة.. هذا هو دور الفئة SqlFileStream.. فلنتعرف عليها.

فئة مجرى بيانات سيكويل SqlFileStream Class

هذه الفئة ترث الفئة Stream، مما يعني أنها تملك خصائص ووسائل القراءة من الملفات والكتابة فيها (راجع فصل الملفات في كتاب إطار العمل).. لكن هذه الفئة مخصصة للتعامل مع الملفات المحفوظة خارج قاعدة البيانات من خلال السمة FILESTREAM.

ولحدث إنشاء هذه الفئة صيغتان:

١- الصيغة الأولى تستقبل ثلاثة معاملات، هي بالترتيب:

- نص يمثل مسار الملف.. ويمكنك الحصول على مسار الملف من خلال استعلام SQL، باستخدام الوسيلة ( )PathName. التي تستخدم مع اسم العمود الذي يحفظ بياناته في ملفات خارجية.
- مصفوفة وحدات ثنائية Byte Array تحتوي على محتوى التعامل Transaction Context.. ويمكنك الحصول عليها من خلال استعلام SQL، باستخدام الوسيلة:

GET_FILESTREAM_TRANSACTION_CONTEXT

لاحظ أنك لا تستطيع إرسال القيمة null إلى هذا المعامل.. هذا معناه أن عليك التعامل مع الملف من خلال تعامل Transaction.. هذا يلفت انتباهك إلى أن سيكويل سيرفر ما زال ينظم عملية الكتابة في الملف، وإن حدث أي خطأ فسيلغي كل التغييرات، ولن يتم حفظ هذه التغييرات إلا إذا قمت بإجراء التعاملات Commit Transactions، كما سنعرف لاحقا

- إحدى قيم المرقم FileAccess التي توضح الطريقة التي تريد التعامل بها مع الملف: للقراءة Read، للكتابة Write، أم للقراءة والكتابة معا ReadWrite.

٢- الصيغة الثانية تزيد على الصيغة الأولى بمعاملين إضافيين:

- معامل من نوع المرقم FileOptions، يحدد خيارات التمعامل مع الملف.. وقد تعرفنا على هذا المرقم في فصل الملفات في كتاب برمجة إطار العمل، لكن ما يهمنا من قيمه هنا هو القيمة SequentialScan، فهي تتيح لنا قراءة

أجزاء من الملف على التوالي، والقيمة RandomAccess التي تتيح لنا القراءة من أي موضع في الملف دون ترتيب.

- عدد صحيح يستقبل الحجم المبدئي الذي تريد حجزه للملف عند إنشائه.. وإذا أردت استخدام القيمة الافتراضية الخاصة بدوت نت، فأرسال إلى هذا المعامل القيمة صفر.

وإضافة إلى ما ترثه من الفئة الأم من وسائل وخصائص، تمتلك هذه الفئة الخاصيتين التاليتين:

الاسم Name:

تعيد مسار الملف المرسل إلى المعامل الأول في حدث الإنشاء.

محتوى التعامل TransactionContext:

تعيد مصفوفة محتوى التعامل المرسلة إلى المعامل الثاني في حدث الإنشاء.

وستجد مثالا على استخدام هذه الفئة في الزر SqlFileStream.Read في المشروع ReadLargeData.. في هذا الزر نستخدم الاستعلام التالي:

```
SELECT ID, Logo3.PathName( ),
GET_FILESTREAM_TRANSACTION_CONTEXT ( )
From Publishers
```

هذا الاستعلام يعيد ثلاثة حقول، هي بالترتيب: رقم الناشر، مسار ملف الصورة، مصفوفة محتوى التعامل Transaction Context الخاصة بالملف.. ويستخدم الكود هذه الحقول لفتح الملف وقراءة محتوياته، وحفظها في ملف جديد على المسار :C.

وستجد مثالا على استخدام هذه الفئة لحفظ صورة في الخانة Logo3 الخاصة بالناشر الثاني، وذلك في الزر SqlFileStream.Write في المشروع WriteLargeData.. في هذا الزر استخدمنا استعلاما شبيها بالاستعلام السابق، لكننا لجأنا أولا إلى حيلة صغيرة، فقد

استخدمنا استعلاما آخر لوضع القيمة Null في الملف، وفصلنا الاستعلامين بالفاصلة المنقوطة:

```
UPDATE Publishers
SET Logo3 = 0x0
Where ID = 2;
SELECT Logo3.PathName( ),
GET_FILESTREAM_TRANSACTION_CONTEXT ( )
From Publishers
Where ID = 2
```

الحكمة من وراء هذا، هو أن خانة الصورة لو كانت فارغة فلن يكون هناك ملف مرتبط بها، وستعيد الوسيلة PathName القيمة Null وبالتالي لن نستطيع الكتابة في الملف بالفئة SqlFileStream.. لهذا سنضع في الخانة القيمة 0x0.. هذا يجعلها تنشئ ملفا وتتركه فارغا، لكن ما يعنينا هنا هو أننا نستطيع معرفة مساره للكتابة فيه.. أما لو كان هناك ملف فعلا وبه بيانات، فسيتم محوها، وهذا يناسبنا فعلا، لأننا سنكتب بيانات جديدة.

أما لو أردت إضافة بيانات إلى نهاية ملف موجود، فستحتاج أولا إلى استخدام الوسيلة SqlFileStream.Seek للوصول إلى نهاية هذا الملف قبل بدء الكتابة فيه، بنفس الطريقة التقليدية التي تتعامل بها مع الملفات العادية:

```
SqlFileStream.Seek(0, SeekOrigin.End);
```

طبعا في هذه الحالة لن تستخدم الاستعلام الذي يضع القيمة 0x0 في الخانة، لأنك تريد المحافظة على بيانات الملف الموجود.

ملحق: ٣
إعداد تطبيق قواعد البيانات على جهاز العميل

بعد أن تنتهي من كتابة مشروع قواعد البيانات، ستحتاج إلى تشغيله على جهاز العميل.. لفعل هذا اتبع الخطوات التالية:

١- قم بإعداد إطار العمل Net Frame work. على جهاز العميل:

استخدم إصدار إطار العمل الذي أنشأت البرنامج عليه في دوت نت.. ستجد ملف إعداد إطار العمل على القرص الخاص بدوت نت حيث سيبدأ اسمه بالحروف: dotNetFx.. أو يمكنك تحميله من موقع ميكروسوفت.

٢- قم بإعداد قاعدة البيانات:

إذا كنت تتعامل مع قاعدة بيانات سيكويل سيرفر، فيجب إعداد نسخة مناسبة من تطبيق SQL Server على جهاز العميل (إن كانت قاعدة البيانات محلية Local)، أو إعداده على الخادم Server إن كانت قاعدة البيانات ستخدم أكثر من مستخدم، وفي هذه الحالة عليك ضبط إعدادات الاتصال بالخادم والـ IP الذي سيتيح الاتصال به من الأجهزة الأخرى (في الغالب هذه مسئولية مدير الشبكة).. وفي كلتا الحالتين، يجب أن تضع قاعدة البيانات في العنوان الذي يتوقع برنامجك أن يجدها فيه (كما حددته في نص الاتصال Connection String)، أو الأفضل من هذا أن يسمح برنامجك للمستخدم باختيار موضع قاعدة البيانات من على الجهاز أو تسمح له بكتابة عنوان الخادم IP في نافذة مخصصة لهذا الغرض.

وإذا كنت تستخدم نسخة SQL Server Express فانظر المقطع الخاص به في نهاية هذا الملحق.

٣- قم بإعداد عارض التقارير:

إذا كان برنامجك يعرض بعض التقارير باستخدام Report Viewer أو Crystal Report، فعليك بإعداد المكتبات اللازمة لهذه التقارير على جهاز العملي.. على سبيل المثال، تحتاج التقارير التي تستخدم الأداة Report Viewer إلى برنامج إعداد اسمه:

Microsoft Report Viewer 2012 Runtime

وهو بدوره يحتاج لوجود إعدادات مسبقة لأنواع سيكويل سيرفر المدارة على جهاز المستخدم، فإن لم تكن موجودة، فيلزمها برنامج إعداد اسمه SQLSysClrTypes.. وكلاهما يمكن تحميله من موقع ميكروسوفت.

٤- قم بإعداد برنامجك:

لو نفذت الخطوات السابقة، ففي الغالب سيعمل الملف التنفيذي الخاص بك على جهاز العميل بدون الحاجة لأي إعدادات أخرى.. كل ما عليك فعله هو وضع كل الصور والملفات اللازمة لعمل برنامجك في مجلد واحد مع الملف التنفيذي (مع مراعاة كتابة الكود منذ البداية ليقرأ هذه الملفات من نفس مجلد الملف التنفيذي)، ثم نسخه إلى جهاز العميل.

أما إذا كنت تتعامل مع أدوات خاصة تحتاج لإعداد ووضع قيم في مسجل الويندوز Registry، ففي هذه الحالة عليك إنشاء برنامج حزم وتوزيع Setup Package يقوم بإعداد برنامجك على جهاز المستخدم.. وقد شرحت هذا الموضوع بالتفصيل في الفضل الأخير من مرجع "من الصفر إلى الاحتراف برمجة نماذج الوندوز".

ملحوظة:

الشرح السابق يخص تطبيقات قواعد البيانات الخاصة بشركة أو عميل محدد، لأنك هنا تعد البرنامج مرة واحدة فقط لعميل واحد، وبالتالي تستطيع أن تريه هذه الخطوات عمليا وتدربه

عليها ليقوم بها بنفسه بعد ذلك، مع كتابة ملف تعليمات يشرح له هذه الخطوات، بحيث لا يحتاج إليك بعد هذا.

لكن الأمر سيختلف مع البرامج التي تباع في السوق لمستخدمين كثر، ففي هذه الحالة عليك أن تكتب برنامج إعداد يقوم بكل أو معظم الخطوات السابقة آليا (أي أنك ستجمع كل الخطوات في الخطوة رقم ٤ في الشرح السابق).. وفي الغالب سيستثنى من الأمر إعداد سيكويل سيرفر لأنه تطبيق مستقل بترخيص، كما أن إعداده على خادم يعمل على شبكة أمر يخص المسئولين عن إدارة هذه الشبكة وتأمينها ومنح الصلاحيات لمستخدميها.

هل يمكن الاعتماد على نسخة SQL Server Express عند توزيع البرنامج؟

SQL Server Express هي النسخة المجانية من سيكويل سيرفر، وهي تسمح لك بالتعامل مع قاعدة بيانات يصل حجمها الأقصى إلى ١٠ جيجا بايت، وعمليات متزامنة تستهلك ١ جيجا من الذاكرة بحد أقصى.. ورغم أن هذه القيود تبدو فقيرة للغاية بجوار ما يمكن أن تفعله النسخة الكاملة من سيكويل سيرفر، تظلّ النسخة المجانية مناسبة جدا للتطبيقات التي تعمل على جهاز شخصي أو شبكة صغيرة أو موقع إنترنت صغير، فملء قاعدة بيانات بـ١٠ جيجا من البيانات أمر صعب، ما لم تكن تتعامل مع شركة عملاقة تخدم آلاف العملاء يوميا، وتحفظ بعض الصور والملفات كبيرة الحجم.

لكن.. ماذا لو زاد الضغط على قاعدة البيانات وامتلأت فعلا وتوقف التعامل معها (فلنقل بعد عامين أو ثلاثة مثلا)؟!

في الحقيقة، أنا لست من أنصار ترك قاعدة البيانات تتضخم بلا حد حتى لو كنا نتعامل مع النسخة الكاملة من سيكويل سيرفر.. فزيادة حجم قاعدة البيانات يسبب سلبيات كثيرة منها:

- وقت أطول في البحث والفهرسة والتحديث والضغط والإصلاح.
- مشاكل أكثر عند حفظ نسخ احتياطية من قاعدة البيانات، تتعلق بمساحة التخزين وبطء النقل.
- وجود بيانات كثيرة قديمة قد تكون الحاجة إليها قد انتهت أو قلت، تظل تدخل في عمليات حسابية أو نقدية أو إحصائيات أو نتائج بحث أو غير ذلك.

لهذا يوجد حلّ عملي بسيط، يُستخدم حتى خارج النظم الرقمية في أي تعاملات ورقية حكومية أو تجارية، وهو إغلاق قاعدة البيانات في نهاية كل عام وحفظها كنسخة أرشيفية، وإنشاء قاعدة بيانات جديدة بتاريخ السنة التالية.. وهكذا يحتفظ البرنامج بقاعدة بيانات مغلقة لكل سنة مضت، ويتعامل فقط مع قاعدة بيانات السنة الحالية، مع تقديم إمكانية للمستخدم لفتح قواعد بيانات السنوات الماضية للبحث فيها (بصلاحية مستخدم) أو تعديلها (بصلاحية المدير).

نصيحة: في النظم التي يملك فيها أكثر من مستخدم صلاحية التعديل، احفظ في كل سجل، بيانات المستخدم الذي أدخله وآخر مستخدم عدّله، لمنع أي تلاعبات.

لو خطّطت برنامجك بهذه الطريقة، فستكون نسخة SQL Server Express كافية للأعمال الصغيرة والمتوسطة التي لا يزيد حجم البيانات الذي تحفظه سنويا عن ١٠ جيجا بايت.
أما الشركات الكبيرة التي تحتاج أكثر من هذا، فلا أظن أنها ستمانع شراء نسخة سيكويل سيرفر مرخصة كاملة الإمكانيات ☺

ويمكنك تحميل نسخة Microsoft® SQL Server® Express من موقع ميكروسوفت (على حسب الإصدار الذي تريده)، كما أوضحنا في بداية هذا الكتاب.

اللهم ارحم أبي واغفر له وكفر عنه سيئاته

ملحوظة:

تم نشر الفصول ١٥ و ١٦ و ١٧ والملحق ١ في كتاب مستقل بعناوان:

جدول عرض البيانات DataGridView

يمكنم تحميله من هنا:

https://drive.google.com/file/d/1tm27L0yGX1RA__vxXng0t5ny3XuJUFv_/view?fbclid=IwAR0IM7zdX9dqkWPXttQyQU6s-FPna2m0hshLq9itiog9SS6PISFdes7Gm_0

أخي الفاضل:

إذا كنت قد استفدت ببعض أو كل ما قرأته في هذا الكتاب، فلا تنسني من دعائك بالهداية والتوفيق والسداد والرزق وحسن الخاتمة.

وادع لأبي رحمه الله بالرحمة، فقد نشرت هذا الكتاب مجانا كصدقة جارية له.. وإذا كنت في الحرم أو على مقربة منه، فلا تبخل بعمل عمرة سريعة له والدعاء له في الحرم المكي والحرم النبوي الشريفين.