مرجع آموزشی

C# 2010

(جلد 2)

مهندس نادر نبوی

انتشارات پندار پارس

سرشناسه : نبوی، نادر، ۱۳۴۰ –
عنوان و نام پدیدآور : مرجع آموزشی C# 2010 / نادر نبوی.
مشخصات نشر : تهران : پندار پارس: ۱۳۹۲ –
مشخصات ظاهری : ج۲. : مصور ،جدول .+ یک لوح فشرده.
شابک : دوره 978-600-6529-07-3 : ؛ ج.۱: 978-600-6529-05-9 : ۶۴۰ص. ج.۲: 978-600-6529-06-6 : ۵۲۸ ص: ۲۳۸۰۰۰ ریال
وضعیت فهرست نویسی : فیپا
یادداشت : ج.۲ (چاپ اول: ۱۳۹۱) (فیپا).
یادداشت : عنوان روی جلد: مرجع آموزشی برنامه‌نویسی به زبان C#2010 با نگاهی به تازه‌های C# 2012 .
عنوان روی جلد : مرجع آموزشی برنامه‌نویسی به زبان C#2010 با نگاهی به تازه‌های C# 2012 .
موضوع : سی شارپ (زبان برنامه‌نویسی کامپیوتر)
رده بندی کنگره : QA۷۶/۷۳ س۹۵/۱۳۹۰ ۲ن
رده بندی دیویی : ۱۳۳/۰۰۵
شماره کتابشناسی ملی : ۲۶۲۱۱۱۳

نام کتاب : مرجع آموزشی C# 2010 با نگاهی به تازه‌های C# 2012 (جلد۲)

ناشر : انتشارات پندار پارس

تالیف : نادر نبوی

چاپ نخست : بهار ۹۲

شمارگان : ۱۰۰۰ نسخه

طرح جلد : فرزانه روزبهانی

لیتوگرافی : ترام‌سنج

چاپ، صحافی : جاویدنو، خیام

قیمت : ۲۳۸۰۰ تومان با DVD رایگان شابک : ۶-۰۶-۶۵۲۹-۶۰۰-۹۷۸ دوره: ۳-۰۷-۶۵۲۹-۶۰۰-۹۷۸

پیش‌گفتار

کتابی که در دست دارید، جلد دوم کتابی است که با عنوان "مرجع آموزش برنامه‌نویسی به زبان C# 2010"، در زمستان سال 1390 به چاپ رسید. از آنجا که معتقدم برنامه‌نویسی درست، جز با بهره‌گیری از یک زمینه‌ی تئوریک قوی امکان‌پذیر نیست، جلد نخست کتاب، بیشتر بر مبانی تئوری این زبان تکیه داشت. بر عکس، در جلد دوم سعی شده است تمرکز بیشتری بر نکات عملی و بررسی چگونگی پیاده‌سازی پروژه‌ها، با استفاده از مفاهیم برگرفته از جلد نخست، داشته باشیم.

از همین روی در بیشتر فصل‌های این کتاب، با پروژه‌هایی برخورد می‌کنید که به تناسب مطالب گفته شده در آن فصل، به‌ویژه برای خواننده‌ای که مطالب گفته شده را نادیده گرفته و سعی بر فهم و اجرای یک‌باره‌ی برنامه داشته باشد، ممکن است قدری پیچیده به نظر برسند. بنابراین توصیه می‌کنم نظم منطقی فصل‌ها و بررسی‌های نخستین انجام گرفته در هر یک از آنها را، پیش از پرداختن به اجرای پروژه‌ی آن فصل، به درستی مد نظر قرار دهید.

فصل 15 به صورت دنباله‌ی منطقی فصل 14 جلد نخست، به بررسی لایه‌ی منفصل ADO.NET، که فهم آن برای کار با Entity Framework ضروری است، اختصاص پیدا کرده است. فصل‌های 16و17 به بررسی پروژه‌های ویندوز، اصول مربوط به کنترل‌ها، خاصیت‌ها و رویدادهای مهم آنها می‌پردازند. در این نقطه، خواننده‌ی گرامی توانایی آشنایی با مبحث مهم مقیدسازی کنترل‌ها در پروژه‌های ویندوز (و یا هر نوع دیگری از پروژه‌های NET.) را پیدا کرده است. بنابراین فصل 18 به استفاده از اشیاء Dataset (و دیگر اشیاء مربوط به آن) در پروژه‌های ویندوز، اختصاص یافته است.

فصل 19 بررسی LINQ بر SQL، با فرض آشنایی کامل خواننده با مفهوم کلی LINQ را، هدف اصلی خود قرار می‌دهد. در فصل‌های 20 تا 22 با ویژگی‌های مهم Entity Framework که ORM توصیه شده از سوی مایکروسافت است، آشنا می‌شوید. این ابزار نگاشت میان سیستم‌های رابطه‌ای و شیءگرا را می‌توانید در همه‌ی انواع پروژه‌ی NET.، از ویندوز و وب گرفته تا WPF و SilverLight به‌کار برید. و سرانجام در فصل 23 به مطالعه‌ی سیستم‌های مبتنی بر سرویس (SOA) با به‌کارگیری تکنولوژی WCF، به عنوان جمع‌بندی هر دو جلد کتاب، پرداخته شده است.

دانشجویی که هر دو جلد را با حوصله و دقت دنبال کرده و به این ترتیب دارای پایه‌هایی قوی از تئوری و برنامه‌نویسی عملی باشد، مشکلی در اجرای پروژه‌های NET. و فهم دیگر تکنولوژی‌های آن در ادامه مطالعاتش، نخواهد داشت.

در پایان از جناب آقای مهندس یعسوبی که کار نشر و امر پر زحمت ویرایش کتاب را بر عهده داشتند، صمیمانه سپاسگذارم.

همان‌گونه که در مقدمه جلد نخست هم گفته‌ام، متن کتاب هنوز هم جای ویرایش دارد. از خوانندگان محترم تقاضا می‌کنم پیشنهادات خود را با آدرس nabavijobmail@yahoo.com در میان بگذارند تا در ویرایش‌های بعدی به خواست خداوند، اصلاح گردند.

نادر نبوی

بهمن 1391

فهرست

فصل پانزدهم
ADO.NET بخش دوم، لایه غیر متصل

در فصل پیش، لایه‌ی متصل ADO.NET را مورد بررسی قرار دادیم که امکان ارائه‌ی جملات کوئری (در حالت کلی، جملات SQL) را به یک پایگاه داده با کمک اشیاء command، data reader و connection (اتصال) فراهم می‌کرد. در این فصل، به بررسی لایه‌ی غیر متصل می‌پردازیم. با استفاده از این جنبه‌ی ADO.NET و با به‌کارگیری اعضای فضای نامی System.Data، می‌توانید داده‌های پایگاه داده را در سمت برنامه‌ی فراخوان[1] مدل‌سازی کنید. با انجام این کار، این‌طور به‌نظر خواهد رسید که برنامه‌ی فراخوان به صورت پیوسته به منبعی از داده‌ها متصل است، در حالی که همه‌ی پردازش‌ها، در واقع بر روی داده‌های محلی[2] انجام می‌شوند. نکته در اینجاست که این داده‌های محلی، در حقیقت یک کپی از داده‌های اصلی موجود در پایگاه داده هستند.

گرچه کاملا امکان‌پذیر است که از لایه‌ی غیر متصل ADO.NET بدون هیچگونه ارجاعی به پایگاه‌های داده‌ی ارتباطی استفاده کنید، ولی در بیشتر موارد و کاربردها، شیء DataSet با داده‌های بر گرفته از جدول یا جدول‌های یک پایگاه داده پر می‌شود؛ و معمولا این کار به‌وسیله‌ی شیء data adapter، یک فراهم کننده‌ی داده‌ی ویژه، انجام می‌پذیرد.

همان‌طور که در بخش‌های آینده‌ی این فصل خواهید دید، شیء data adapter به‌عنوان میانجی و رابط میان شیء DataSet (در سمت برنامه‌ی کاربردی یا لایه‌ی مشتری) و عناصر (جدول‌ها، نماها یا روال‌های ذخیره شده) پایگاه داده عمل می‌کند. با استفاده از این اشیاء، برنامه‌ی کاربردی می‌تواند ذخیره‌سازی داده‌های پایگاه داده در اشیاء DataSet، دستکاری این داده‌ها و در آخر، برگرداندن داده‌های تغییر یافته به پایگاه داده (محل ذخیره‌سازی فیزیکی داده‌ها) را انجام دهد.

لازم است در همین‌جا، به این نکته اشاره کنیم که کار انتقال داده‌های جداول پایگاه داده به مکان ذخیره‌سازی محلی (local) یعنی DataSet را، اصطلاحا Fill و کار عکس آن، یعنی به‌روز رسانی پایگاه داده با داده‌های تغییر یافته را، Update کردن آن می‌گویند. همان‌طور که خواهید دید، شیء data adapter برای انجام این اعمال، دارای متدهایی به‌نام‌های ()Fill و ()Update است.

در فصل‌های شانزده و هفده، با ایجاد پروژه‌های ویندوز و کار با فرم‌ها و کنترل‌های ویندوز آشنا خواهید شد؛ بنابراین مبحث مقیدسازی کنترل‌ها و استفاده از فرم‌های ویندوز در نمایش داده‌های ذخیره شده در اشیاء DataSet را، در فصل هیجده خواهید دید.

---

[1] client tire
[2] local data

آخرین مطالب و البته نه کم اهمیت‌ترین آنها، آشنایی با LINQ بر DataSet و LINQ بر SQL خواهد بود که امکان اعمال کوئری‌های LINQ را بر داده‌های ذخیره شده در DataSetها فراهم می‌آورند. این مورد را نیز، به دلیل استفاده از پروژه‌های ویندوز و توانایی ترکیب مفاهیم آن با مقیدسازی کنترل‌ها، به فصل 19 می‌سپاریم.

15-1 درک کارکرد لایه‌ی غیر متصل ADO.NET

همان‌طور که در فصل گذشته دیدید، کار با لایه‌ی متصل، متضمن تعامل با اشیاء اتصال، command و data reader است. با استفاده از این کلاس‌ها قادر به انتخاب، افزودن، ویرایش و حذف مستقیم رکوردها از جدول‌های پایگاه داده بودید. با همه‌ی اینها، تا اینجای کار تنها با قسمتی از کل داستان ADO.NET آشنا شده‌اید. همان‌طور که تاکنون متوجه شده‌اید، با استفاده از اشیاء ADO.NET به‌صورتی غیرمستقیم و غیر متصل نیز می‌شود با داده‌های پایگاه داده کار کرد.

با این روش، می‌توان داده‌های یک پایگاه داده‌ی رابطه‌ای را در حافظه (درون DataSet) مدل‌سازی کرد که البته این امر، بسیار فراتر از مدل‌سازی جدولی از سطرها و ستون‌ها است؛ زیرا کلاس‌های فضای نامی System.Data امکان نمایش ارتباطات میان جدول‌ها، محدویت‌های مربوط به فیلدها[1]، کلیدهای اصلی[2]، نماها[3] و دیگر عناصر پایگاه داده را نیز فراهم می‌آورند.

در فصل‌های آینده، با کاربرد LINQ بر SQL و LINQ بر موجودیت‌ها آشنا خواهید شد و در آنجا درخواهید یافت که چگونه با استفاده از آن روش‌ها، می‌توان به یک مدل‌سازی کاملا شیءگرا از داده‌های یک پایگاه داده‌ی ارتباطی دست یافت. به هر روی، در روشی که اکنون مورد بحث است، یعنی روش DataSet، پس از مدل‌سازی داده‌ها درون حافظه، می‌توان فیلترهای مختلف به آنها اعمال کرد، کوئری‌های درون حافظه‌ای[4] را اجرا کرد و یا داده‌های موجود را به‌صورت فایل‌های XML یا باینری ذخیره و یا بارگذاری کرد. دلیل اینکه این روش را غیر متصل می‌نامند، این است که همه‌ی این کارها را بدون ایجاد ارتباط با پایگاه داده و استفاده از شیء اتصال و تنها با ایجاد یک DataSet (با استفاده از کدنویسی) و یا بارگذاری داده‌ها از یک فایل محلی XML، انجام خواهید داد.

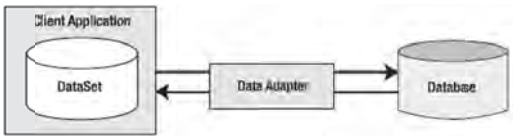
اکنون این پرسش مطرح می‌شود که DataSet و یا فایل XML، از ابتدا چگونه با داده‌های مورد نظر پر می‌شوند. پاسخ این پرسش، در کاربرد شیء ویژه‌ای به‌نام data adapter نهفته است (که از کلاس مجرد DbDataAdapter، مشتق شده است). شیء data adapter، مسئول اخذ اطلاعات از پایگاه داده و به‌روز رسانی آن است. بر خلاف آنچه که در روش متصل دیدید، داده‌های تهیه شده به‌وسیله‌ی یک data adapter از

[1] column constraints
[2] primary keys
[3] views
[4] in-memory queries

طریق شیء data reader پردازش نمی‌شوند، بلکه این اشیاء از شیء دیگری به‌نام DataSet برای انتقال داده‌ها میان برنامه‌ی فراخوان و منبع داده‌ها استفاده می‌کنند.

یک شیء DataSet می‌تواند در بر گیرنده‌ی هر تعداد اشیائی از نوع DataTable (جداول داخل DataSet) باشد، و این اشیاء هم به نوبه‌ی خود می‌توانند دارای هر تعداد اشیائی از نوع DataRow (سطرهای جداول) و DataColumn (ستون‌های جداول) باشند. به بیانی دیگر، یک DataSet کلکسیونی از سطرها (DataRow) و ستون‌ها (DataColumn) است.

مدیریت اتصال با پایگاه داده، به‌صورت خودکار با شیء data adapter است؛ و برای بالا بردن کارآیی برنامه، مدت زمان اتصال تا جای ممکن به حداقل کاهش می‌یابد. پس از اینکه برنامه‌ی فراخوان، محتویات Dataset را به‌دست آورد، اتصال آن به‌طور کامل با پایگاه داده قطع می‌شود و پس از آن، تنها به کپی محلی داده‌ها دسترسی خواهد داشت. در این حالت، برنامه‌ی فراخوان آزاد است که هر تعداد رکورد جدید به داده‌های موجود بیفزاید، آنها را تغییر دهد و یا ویرایش کند، ولی داده‌های اصلی در پایگاه داده بدون تغییر باقی می‌مانند. هنگامی که برنامه‌ی مشتری بخواهد تغییرها را در جدول‌های اصلی پایگاه داده اعمال کند، باید به‌صورت صریح DataSet را برای به‌روز رسانی پایگاه به شیء data adapter (معمولا همان data adapterی که مسئول پر کردن DataSet بوده است)، ارسال کند. شکل 1-15، این فرآیند را روشن‌تر بیان می‌کند.

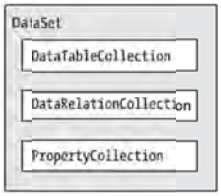


شکل 1-15

با توجه به اینکه کلاس DataSet هسته‌ی مرکزی لایه‌ی غیر متصل را تشکیل می‌دهد، نخستین قدم در این فصل، آشنایی با چگونگی ایجاد یک شیء DataSet، البته به‌صورت دستی (در این مرحله، کاری با پایگاه داده و جداول آن نداریم) خواهد بود. پس از به‌دست آوردن دانش این کار، دست‌کاری و پردازش DataSetی که با یک شیء data adapter پر شده باشد، آسان خواهد بود.

15-2 آشنایی با نقش DataSet

در آغاز یادآور می‌شویم که DataSet نمایشی از داده‌های فیزیکی یک پایگاه داده‌ی رابطه‌ای، درون حافظه‌ی کامپیوتر برنامه‌ی فراخوان (یا مشتری client) است. به‌صورت دقیق‌تر، DataSet کلاسی است دارای سه کلکسیون اصلی (شکل 2-15).



شکل 15-2

با استفاده از خاصیت Tables شیء DataSet، می‌توانید به کلکسیون DataTableCollection دارای DataTableها دسترسی پیدا کنید. با توجه به اینکه DataSet نمایشی از داده‌های جدول‌های یک پایگاه داده‌ی رابطه‌ای است، کلکسیون DataRelationCollection نگاه‌دارنده‌ی ارتباط‌های میان DataTableها است. برای مثال می‌توانید با استفاده از کلاس DataRelation، ارتباطی میان دو جدول درون DataSet ایجاد کنید (به بیان SQL Server، این امر برابر اعمال محدودیتی از نوع کلید خارجی به جداول است)؛ سپس می‌توانید ارتباط ایجاد شده (در واقع همان شیء DataRelation) را، به‌وسیله‌ی خاصیت Relations مربوط به شیء DataRelationCollection به آن بیفزایید. در این مرحله می‌توانید با استفاده از ارتباط ایجاد شده، میان دو جدول مرتبط حرکت کنید و به جست‌وجوی داده‌ها بپردازید (برای نمونه، جست‌وجوی رکوردهایی در جدول سمت چند که دارای شرط خاصی در جدول سمت یک باشند، یا برعکس). البته در این فصل، با همه‌ی این اعمال، آشنا خواهید شد.

15-2-1 خاصیت‌های کلیدی DataSet

اجازه دهید پیش از اینکه وارد جزئیات بیشتری در مورد کدنویسی شویم، نگاهی به برخی از خصوصیات مهم کلاس DataSet داشته باشیم. جدول 15-1، شماری از این خصوصیات را (به غیر از خصوصیات اصلی Tables و Relations و ExtendedProperties) فهرست کرده است:

جدول 15-1

خصوصیت	معنی و مفهوم
CaseSensitive	این خاصیت مشخص می‌کند که آیا مقایسه مقادیر رشته‌ای در اشیاء DataTable درون DataSet، به حروف کوچک و بزرگ انگلیسی حساس باشد یا نباشد. مقدار پیش‌فرض این خاصیت، false است (مقایسه‌های رشته‌ای، به کوچکی و بزرگی حروف حساس نیستند).
DataSetName	نام (قابل ارجاع در برنامه) DataSet را مشخص می‌کند. معمولا این مقدار را، به‌عنوان پارامتر سازنده مشخص می‌کنید.
EnforceConstraints	خاصیتی منطقی (قابل خواندن و نوشتن)، مشخص کننده‌ی اینکه آیا قوانین پایگاه

خصوصیت	معنی و مفهوم
	داده در مورد حفظ جامعیت، هنگام به‌روز رسانی رکوردها رعایت شود یا نه (مقدار پیش‌فرض، true است).
HasErrors	مقداری را به‌عنوان اینکه آیا خطایی در سطرهای جداول (DataTables) درون DataSet موجود هست یا نه، باز می‌گرداند.
RemotingFormat	به‌وسیله‌ی این خاصیت، می‌توان چگونگی تثبیت (سریال سازی) محتوای DataSet را مشخص کرد (به‌صورت باینری یا XML که مقدار پیش‌فرض برای فایل خروجی، XML است).

15-2-2 متدهای کلیدی شیء dataSet

متدهای DataSet، در ارتباط با خواص اشاره شده در جدول 15-1 کار می‌کنند. افزون بر امکان کار با فایل‌های متنی، DataSet دارای متدهایی برای کپی محتویات جدول‌هایش، حرکت میان جدول‌های داخلی و تعیین نقاط آغاز و پایان به‌روز رسانی‌های دسته جمعی[1] است. برخی از متدهای مهم، در جدول 15-2 لیست شده‌اند.

جدول 15-2

متد	معنی و عملکرد
()AcceptChanges	همه‌ی تغییرات انجام شده را، از زمان بار شدن DataSet به حافظه و یا پس از آخرین اجرای متد ()AcceptChanges، ذخیره و تثبیت می‌کند (معادل commit در پایگاه داده).
()Clear	DataSet را با حذف همه‌ی سطرهای جدول‌های آن، کاملا خالی می‌کند.
()Clone	یک کپی از ساختار DataSet (بدون داده‌ها)، به‌همراه همه‌ی جدول‌ها و ارتباطات آنها ایجاد می‌کند.
()Copy	ساختار یک DataSet را به‌همراه داده‌های آن کپی می‌کند.
()GetChanges	یک کپی از DataSet را شامل همه‌ی تغییرهای انجام شده در آن، از زمان بار شدن به حافظه و یا آخرین اجرای متد ()AcceptChanges باز می‌گرداند. این متد به شکل‌های مختلفی سربارگذاری شده است، بنابراین امکان می‌دهد تنها سطرهای

[1] batch updates

متد	معنی و عملکرد
	افزوده شده، سطرهای تغییر یافته و یا تنها سطرهای حذف شده را به‌وسیله‌ی آن استخراج کنید.
()HasChanges	مقداری منطقی را، نشان‌دهنده‌ی اینکه آیا DataSet دچار تغییراتی شده است یا نه، باز می‌گرداند. این تغییرات می‌تواند مربوط به افزوده شدن سطرهای جدید، ویرایش سطرهای موجود و یا حذف سطرها باشد.
()Merge	DataSet مورد نظر را، با DataSet دیگری ادغام می‌کند.
()ReadXml	امکان تشکیل ساختار یک DataSet و پر شدن جدول‌های آن را بر اساس داده‌های یک سند XML فراهم می‌کند.
()RejectChanges	همه‌ی تغییرات انجام شده بر روی داده‌های DataSet را از زمان ایجاد آن و یا آخرین اجرای متد ()AcceptChanges به حالت ابتدایی باز می‌گرداند (معادل roll back در پایگاه داده).
()WriteXml	امکان نوشتن محتوای DataSet را به یک سند XML فراهم می‌کند.

15-2-3 ایجاد DataSet

اکنون که درک بهتری از یک DataSet و چگونگی کارکرد آن به‌دست آورده‌اید، پروژه‌ی کنسول جدیدی به‌نام SimpleDataSet ایجاد و فضای نامی System.Data را به آن وارد کنید. درون متد ()Main، یک DataSet ایجاد خواهیم کرد که (تنها برای مثال) دارای سه خاصیت تعمیم یافته، نشان دهنده‌ی زمان، یک شناسه‌ی عددی منحصر به‌فرد و نام شرکت باشد:

```
static void Main(string[] args)
{
  Console.WriteLine("***** Fun with DataSets *****\n");

  // Create the DataSet object and add a few properties.
  DataSet carsInventoryDS = new DataSet("Car Inventory");
  carsInventoryDS.ExtendedProperties["TimeStamp"] = DateTime.Now;
  carsInventoryDS.ExtendedProperties["DataSetID"] = Guid.NewGuid();
  carsInventoryDS.ExtendedProperties["Company"] =
    "Mikko's Hot Tub Super Store";
  Console.ReadLine();
}
```

در مثال بالا، از ساختار System.Guid برای ایجاد شناسه‌ی یکتا استفاده شده است. متد ()NewGuid از ساختار بالا، یک کد عددی[1] یکتا (128 بیتی) تولید می‌کند.

به هر روی، کار با یک DataSet تا زمانی که دارای جدولی نباشد جالب نخواهد بود. بنابراین قدم بعدی، بررسی چگونگی کارکرد شیء DataTable خواهد بود. برای این کار، از شیء DataColumn آغاز خواهیم کرد.

15-3 کار با ستون‌ها DataColumns

نوع DataColumn نشان دهنده‌ی یک ستون، درون شیئی از نوع DataTable است. به بیانی کلی‌تر، اطلاعات مربوط به مجموعه‌ی همه‌ی DataColumnهای یک DataTable، مشخص کننده‌ی طرح جدول[2] آن می‌باشد. به‌عنوان مثال اگر بخواهید جدول Inventory از پایگاه داده‌ی AutoLot (که در جلد نخست کتاب ایجاد کردید) را به این‌صورت مدل‌سازی کند، برای هر یک از فیلدها (CarID و Color و PetName)، یک DataColumn با همان مشخصه‌های فیلد مربوط ایجاد خواهید کرد. در مرحله‌ی بعد، پس از ایجاد اشیاء DataColumn، آنها را به کلکسیون ستون‌های DataTable (به‌وسیله‌ی خاصیت Columns) اضافه خواهید کرد.

حتما با مفهوم محدودیت‌هایی[3] که می‌توان به ستون‌های جداول پایگاه داده اعمال کرد، آشنایی دارید. این محدودیت‌ها می‌توانند مربوط به کلید اصلی، مقدار پیش فرض ستون و بسیاری موارد دیگر باشند. همچنین می‌دانید که هر ستون در یک جدول، باید دارای نوع داده‌ی مشخصی باشد. به‌عنوان مثال، در جدول Inventory، نوع فیلد CarID[4] به‌صورت عدد صحیح (int) است در حالی که فیلدهای Make و Color و PetName از نوع رشته‌ای هستند. کلاس DataColumn نیز دارای خاصیت‌هایی برای تعیین چنین مواردی است. جدول 3-15، برخی از خاصیت‌های مهم تعریف شده در این کلاس را لیست کرده است.

جدول 3-15

خصوصیت	معنی و مفهوم
AllowDBNull	از این خاصیت، برای تعیین اینکه ستون، قابلیت پذیرش مقادیر null را دارد یا ندارد، استفاده می‌کنید.
AutoIncrement AutoIncrementSeed AutoIncrementStep	از این خاصیت، برای تنظیم رفتار افزایش خودکار ستون استفاده می‌کنید. کاربرد این خصوصیت، در زمانی که می‌خواهید مقادیر ستون یکتا و غیر

[1] globally unique identifier

[2] table schema information نام، نوع و دیگر مشخصات ستون‌های جدول، طرح آن را مشخص می‌کنند و جدول بر اساس این اطلاعات ایجاد می‌شود.

[3] constraints

[4] توجه دارید که در بحث حاضر، لغات فیلد و ستون را به‌صورت مترادف به‌کار می‌بریم.

خصوصیت	معنی و مفهوم
	تکراری (مانند مقادیر کلید اصلی جدول) باشند، مفید خواهد بود.
Caption	این خاصیت، امکان تعیین عنوانی را برای ستون فراهم می‌آورد. کاربرد آن، به‌خصوص در جایی‌که بخواهید عنوانی ملموس به نام فیلدی از جدول پایگاه داده بدهید، مفید خواهد بود.
ColumnMapping	این خصوصیت، تعیین کننده‌ی چگونگی نمایان شدن داده‌های ستون در فایل XML است (زمانی که DataSet به‌وسیله‌ی متد ()WriteXml در یک سند XML درج شود). با استفاده از آن، می‌توان فرم درج ستون را به‌صورت یک عنصر (element)، یک صفت (attribute) یا محتوای متنی تعیین کرد و یا معین کنیم که محتوای ستون اصلا در فایل XML درج نشود و از آن صرف نظر گردد.
ColumnName	این خصوصیت نوشتنی و خواندنی، مشخص کننده‌ی نام ستون در DataSet است. اگر مقداری برای آن مشخص نکنید، ستون‌ها به‌ترتیب شماره گذاری می‌شوند (Column1 و Column2 و تا آخر).
DataType	این خاصیت، تعیین کننده‌ی نوع داده‌ی ستون است.
DefaultValue	این خاصیت خواندنی و نوشتنی، تعیین کننده‌ی مقدار پیش فرض ستون در هنگام افزودن سطر جدید است.
Expression	این خصوصیت نوشتنی و خواندنی، عبارتی را برای فیلتر کردن سطرها، محاسبه‌ی مقدار ستون (ستون محاسباتی) و یا ایجاد یک ستون تجمعی (میانگین، کمینه، بیشینه و دیگر) مشخص می‌کند.
Ordinal	این خاصیت، تعیین کننده‌ی موقعیت عددی ستون در کلکسیون ستون‌های DataTable است.
ReadOnly	این خاصیت، تعیین می‌کند که آیا محتوای ستون پس از اضافه شدن یک سطر به جدول، تنها خواندنی باشد و یا نباشد. مقدار پیش فرض آن، false است.
Table	جدولی (DataTable) را که ستون متعلق به آن است باز می‌گرداند.
Unique	این خاصیت خواندنی و نوشتنی، تعیین کننده‌ی یکتایی و یا قابل تکرار بودن مقادیر محتوای ستون است. اگر به ستونی محدودیت کلید اصلی را اعمال کنید، مقدار این خاصیت نیز باید به true تنظیم شود.

15-3-1 ایجاد شیء DataColumn

در ادامه‌ی پروژه‌ی SimpleDataSet (و البته برای آشنایی بیشتر با DataColumn)، فرض کنید می‌خواهیم جدول Inventory را مدل‌سازی کنیم. با توجه به اینکه فیلد CarID، کلید اصلی جدول است، باید شیء DataColumn مربوط به آن، به‌صورت فقط خواندنی، یکتا و غیر تهی (non-null) تنظیم شود. در مرحله‌ی بعد، متد ()FillDataSet را به کلاس Program بیفزایید. از این متد، برای ایجاد چهار ستون استفاده خواهیم کرد. توجه کنید که این متد، یک شیء DataSet را به‌عنوان تنها پارامتر ورودی خود می‌پذیرد:

```
static void FillDataSet(DataSet ds)
{
  // Create data columns that map to the
  // 'real' columns in the Inventory table
  // of the AutoLot database.
  DataColumn carIDColumn = new DataColumn("CarID", typeof(int));
  carIDColumn.Caption = "Car ID";
  carIDColumn.ReadOnly = true;
  carIDColumn.AllowDBNull = false;
  carIDColumn.Unique = true;
  DataColumn carMakeColumn = new DataColumn("Make", typeof(string));
  DataColumn carColorColumn = new DataColumn("Color", typeof(string));
  DataColumn carPetNameColumn = new DataColumn("PetName", typeof(string));
  carPetNameColumn.Caption = "Pet Name";
}
```

شرح برنامه

توجه کنید که هنگام پیکربندی شیء carIDColumn، مقدار Car ID به خاصیت Caption نسبت داده شده است. استفاده از این خاصیت، در مواردی می‌تواند مفید باشد که می‌خواهید عنوانی غیر از عنوان اصلی فیلد در پایگاه داده را برای آن نمایش دهید؛ زیرا اسامی فیلدها در جدول‌های پایگاه داده (مانند au_fname) معمولا برای برنامه‌نویسی مناسب‌تر هستند، تا نمایش (مثلا به‌صورت قابل فهم‌تر Author First Name). در اینجا هم به‌همین منظور خاصیت Caption را برای فیلد PetName مقداردهی می‌کنید، زیرا Pet Name برای کاربر نهایی واضح‌تر است تا PetName.

15-3-2 استفاده از ستون Autoincrement

یکی از ویژگی‌هایی که می‌توان برای شیء DataColumn در نظر گرفت، توانایی آن در افزایش خودکار است. دلیل کاربرد چنین خاصیتی، اطمینان پیدا کردن از این امر است که با اضافه شدن سطر جدید، مقدار ستون بر

اساس گام‌هایی که برای افزایش آن مشخص شده است، به‌صورت خودکار تغییر کند. به‌کار بردن این خاصیت، در مواردی مفید است که محتویات ستون نباید تکراری باشد (مثلا در مورد کلید اصلی یک جدول).

این رفتار را می‌توان با خاصیت‌های AutoIncrement،AutoIncrementSeed و AutoIncrementStep کنترل کرد. از seed، به‌عنوان مقدار آغاز افزایش و از step، به‌عنوان تعیین کننده‌ی گام‌های افزایش ستون استفاده می‌کنید. نگارش دیگری از ایجاد ستون carIDColumn را، در زیر می‌بینید:

```
static void FillDataSet(DataSet ds)
{
  DataColumn carIDColumn = new DataColumn("CarID", typeof(int));
  carIDColumn.ReadOnly = true;
  carIDColumn.Caption = "Car ID";
  carIDColumn.AllowDBNull = false;
  carIDColumn.Unique = true;
  carIDColumn.AutoIncrement = true;
  carIDColumn.AutoIncrementSeed = 0;
  carIDColumn.AutoIncrementStep = 1;
  ...
}
```

با توجه به کد بالا، محتوای این ستون در هر بار اضافه شدن یک سطر جدید، به‌صورت 0,1,2,3,... مقدار می‌گیرد.

15-3-3 اضافه کردن اشیاء DataColumn به شیء DataTable

معمولا اشیاء DataColumn، به‌صورت مستقل کاربردی ندارد و باید به شیئی از نوع DataTable اضافه شوند. در مثال زیر، شیئی از نوع DataTable ایجاد و پس از آن، هر یک از DataColumnها به‌وسیله‌ی خاصیت Columns به کلکسیون ستون‌های آن (کلکسیون Columns)، اضافه می‌شوند:

```
static void FillDataSet(DataSet ds):
{
  ...
  // Now add DataColumns to a DataTable.
  DataTable inventoryTable = new DataTable("Inventory");
  inventoryTable.Columns.AddRange(new DataColumn[]
  { carIDColumn, carMakeColumn, carColorColumn, carPetNameColumn });
}
```

تا اینجای کار، شیء DataTable مثالمان (inventoryTable)، دارای چهار شیء DataColumn است که طرح جدول Inventory را در حافظه تشکیل داده‌اند. با این حال شیء DataTable بالا، هنوز خالی از داده‌ها

می‌باشد و افزون بر آن، به کلکسیون جدول‌ها (کلکسیون Tables) یک DataSet نیز اضافه نشده است. برای رفع این ایرادها، نخست جدول را با اشیاء DataRow پر خواهیم کرد.

15-4 کار با سطرها DataRows

همان‌گونه که دیدید، اشیاء DataColumn نماینده‌ی طرح یک DataTable هستند. در نقطه‌ی مقابل، اشیاء DataRow نماینده‌ی داده‌های جدول هستند. بنابراین اگر جدول Inventory در پایگاه داده‌ی AutoLot دارای 20 سطر باشد، برای نمایش آنها از 20 شیء DataRow استفاده خواهیم کرد. جدول 4-15، برخی از اعضای مهم کلاس DataRow را لیست کرده است.

جدول 4-15

خصوصیت	معنی و مفهوم
HasErrors GetColumnsInError() GetColumnError() ClearErrors() RowError	خاصیت HasErrors، مقداری منطقی بازمی‌گرداند که نشان دهنده‌ی وجود یا عدم وجود خطا در DataRow است. اگر خطایی وجود داشته باشد، می‌توانید از متد ()GetColumnsInError برای دست‌یابی به ستون‌های واجد خطا و از متد ()GetColumnError برای دریافت شرح خطاهای رخداده، استفاده کنید. به‌همین ترتیب با استفاده از متد ()ClearErrors می‌توانید هر یک از خطاهای لیست شده برای سطر را حذف کنید. خاصیت RowError، امکان تعریف شرحی (به‌صورت متن) برای خطایی مشخص را فراهم می‌کند.
ItemArray	از این خاصیت می‌توان برای به‌دست آوردن و یا تعیین مقادیر همه‌ی ستون‌های سطر، به‌صورت آرایه‌ای از اشیاء استفاده کرد.
RowState	از این خاصیت برای دسترسی به وضعیت حاضر یک سطر، به‌وسیله‌ی یکی از مقادیر نوع شمارشی RowState، استفاده می‌کنید (یک سطر می‌تواند به‌صورت new، modified، unchanged یا deleted علامت‌گذاری شده باشد).
Table	با استفاده از این خاصیت، ارجاعی به شیء DataTable در بر گیرنده‌ی سطر جاری، به‌دست می‌آورید.
AcceptChanges() RejectChanges()	این خاصیت‌ها برای پذیرش یا لغو همه‌ی تغییرات انجام شده بر سطر جاری، از زمان آخرین اجرای متد ()AcceptChanges به بعد، به‌کار می‌روند.
BeginEdit() EndEdit()	این متدها برای آغاز، پایان بخشیدن یا لغو همه‌ی عملیات ویرایشی روی یک

خصوصیت	معنی و مفهوم
CancelEdit()	سطر به‌کار می‌روند.
Delete()	این متد، سطری را به‌عنوان حذف شده علامت‌گذاری می‌کند. در این حالت، کاربرد متد AcceptChanges() سبب حذف کامل سطر خواهد شد.
IsNull()	خروجی منطقی این متد، نشان دهنده‌ی null یا غیر null بودن ستونی است که مشخص می‌گردد.

تفاوت اصلی کار با کلاس DataRow نسبت به کلاس DataColumn در این است که این کلاس دارای سازنده‌ای عمومی نیست، بنابراین نمی‌توان نمونه‌ی مستقیمی از روی آن ایجاد کرد.

```
// Error! No public constructor!
DataRow r = new DataRow();
```

با توجه به این نکته، بهترین روش این است که شیء جدید DataRow را از یک جدول (DataTable) آماده به دست آورید. در واقع کلاس DataTable، به‌همین منظور، دارای متدی به‌نام NewRow() است که نخستین سطر خالی جدول را باز می‌گرداند؛ سپس می‌توان ستون‌های این سطر خالی را با استفاده از اندیس ستون، با داده‌های مورد نظر پر کرد. برای انجام این کار، هم می‌توانید از نام ستون (به‌صورت یک مقدار رشته‌ای) و هم از اندیس عددی، به‌عنوان موقعیت ستون در سطر استفاده کنید.

```
static void FillDataSet(DataSet ds)
{
  ...
  // Now add some rows to the Inventory Table.
  DataRow carRow = inventoryTable.NewRow();
  carRow["Make"] = "BMW";
  carRow["Color"] = "Black";
  carRow["PetName"] = "Hamlet";
  inventoryTable.Rows.Add(carRow);
  carRow = inventoryTable.NewRow();

  // Column 0 is the autoincremented ID field,
  // so start at 1.
  carRow[1] = "Saab";
  carRow[2] = "Red";
  carRow[3] = "Sea Breeze";
  inventoryTable.Rows.Add(carRow);
}
```

شرح برنامه

همان‌طور که در مثال بالا می‌بینید، در این روش ابتدا یک سطر خالی از جدول گرفته می‌شود، ستون‌های این سطر با داده‌هایی پر می‌شود و در مرحله‌ی آخر، باید دوباره این سطر (که اکنون دارای داده‌های مورد نظر است) را به جدول اصلی افزود. مرحله‌ی آخر، به‌وسیله‌ی متد ()Add از کلکسیون Rows مربوط به شیء DataTable انجام می‌شود.

تا اینجای کار دارای یک DataTable به‌نام InventoryTable که شامل دو سطر است، هستیم. البته می‌توانید همین روش عمومی را ادامه دهید و هر تعداد DataTable که مایل باشید، متفاوت با طرح‌ها[1] و محتویات دلخواه ایجاد کنید. با این حال در این مرحله، پیش از پیوست شیء InventoryTable به DataSet، باید با کاربرد خاصیت مهم RowState آشنا شوید.

15-4-1 آشنایی با خاصیت RowState

خاصیت RowState می‌تواند در شناسایی سطرهایی که به نوعی تغییر کرده‌اند، از جمله سطرهایی که ویرایش و یا به جدول اضافه شده‌اند، مفید باشد. می‌توان به این خاصیت، هر یک از مقادیر نوع شمارشی DataRowState را به همان صورتی که در جدول 15-5 آمده است، نسبت داد.

جدول 15-5

مقدار نوع شمارشی DataRowState	معنی و مفهوم
Added	سطر مزبور به کلکسیون Rows اضافه می‌گردد، ولی متد ()AcceptChanges هنوز فراخوانی و اجرا نشده است.
Deleted	سطر مورد نظر در اثر اجرای متد ()Delete از کلاس DataRow برای حذف علامت خورده، ولی متد ()AcceptChanges هنوز فراخوانی و اجرا نشده است.
Detached	سطر، ایجاد شده ولی هنوز به هیچ کلکسیون سطری (Rows) پیوست نشده است. به‌همین صورت اگر سطری از جدولی حذف شود نیز، دارای این وضعیت خواهد بود.
Modified	سطر مورد نظر، ویرایش شده ولی متد ()AcceptChanges هنوز اجرا نشده است.
Unchanged	سطر مورد آزمایش، از زمان آخرین اجرای متد ()AcceptChanges تغییری نکرده است.

[1] schema

هنگام دستکاری سطرهای یک DataTable به‌وسیله‌ی کدنویسی، خاصیت RowState به‌صورت خودکار مقداردهی می‌شود. برای آزمایش گفته‌ی بالا، متد جدیدی به کلاس Program اضافه می‌کنیم که تغییراتی را به‌صورت یک شیء محلی[1] (یعنی در درون متد ایجاد می‌شود) بر روی سطری می‌دهد و سپس وضعیت آن را نمایش می‌دهد:

```
private static void ManipulateDataRowState()
{
  // Create a temp DataTable for testing.
  DataTable temp = new DataTable("Temp");
  temp.Columns.Add(new DataColumn("TempColumn", typeof(int)));

  // RowState = Detached (i.e. not part of a DataTable yet)
  DataRow row = temp.NewRow();
  Console.WriteLine("After calling NewRow(): {0}", row.RowState);

  // RowState = Added.
  temp.Rows.Add(row);
  Console.WriteLine("After calling Rows.Add(): {0}", row.RowState);

  // RowState = Added.
  row["TempColumn"] = 10;
  Console.WriteLine("After first assignment: {0}", row.RowState);

  // RowState = Unchanged.
  temp.AcceptChanges();
  Console.WriteLine("After calling AcceptChanges: {0}", row.RowState);

  // RowState = Modified.
  row["TempColumn"] = 11;
  Console.WriteLine("After first assignment: {0}", row.RowState);

  // RowState = Deleted.
  temp.Rows[0].Delete();
  Console.WriteLine("After calling Delete: {0}", row.RowState);
}
```

نکته‌ی قابل توجه این است که شیء DataRow، برای به‌یاد آوردن همه‌ی تغییراتی که رخ داده است به اندازه‌ی کافی هوشمندانه کار می‌کند. با توجه به این مطلب، شیء DataTable در برگیرنده‌ی DataRow می‌تواند

[1] local DataRow object

سطرهایی را که اضافه یا ویرایش شده‌اند یا سطرهایی را که برای حذف علامت خورده‌اند، شناسایی کند؛ و این امر، ویژگی اصلی DataSet است؛ زیرا بر این اساس، تنها داده‌های ویرایش شده برای به‌روزرسانی پایگاه داده، بازپس فرستاده خواهند شد.

15-4-2 آشنایی با خاصیت DataRowVersion

صرف نظر از اینکه وضعیت جاری یک سطر را می‌توان با خاصیت RowState به‌دست آورد، شیء DataRow ممکن است دارای سه نگارش متفاوت از داده باشد، که با استفاده از خاصیت DataRowVersion قابل دست‌یابی است. هنگامی که یک DataRow برای بار نخست ایجاد می‌شود، تنها دارای یک کپی از داده است که با نگارش current (جاری) مشخص می‌شود. با این حال پس از دست‌کاری داده‌های سطر (با استفاده از فراخوانی متدهای مختلف)، نگارش‌های دیگری از داده‌ها به‌وجود می‌آیند. خاصیت DataRowVersion می‌تواند به‌طور مشخص بر اساس هر یک از مقادیر نوع شمارشی DataRowVersion که در جدول 6-15 آمده، تنظیم شود.

جدول 6-15

مقدار نوع شمارشی DataRowVersion	معنی و مفهوم
Current	نشان‌دهنده‌ی مقدار جاری یک سطر، حتی پس از اعمال تغییرات است.
Default	نگارش پیش‌فرض وضعیت سطر (DataRowState). برای وضعیت‌هایی مانند Added یا Modified یا Deleted، نگارش پیش‌فرض Current است. برای مقداری از وضعیت سطر برابر با Detached، نگارش برابر با Proposed خواهد بود.
Original	می‌تواند نشان‌دهنده‌ی مقداری که برای بار نخست در DataRow وارد شده است و یا مقدار کنونی DataRow، پس از آخرین فراخوانی متد ()AcceptChanges باشد.
Proposed	مقدار سطری را، که هم‌اکنون بر اثر فراخوانی متد ()BeginEdit در حال ویرایش است، به‌دست می‌دهد.

همان‌طور که در جدول 6-15 می‌بینید، مقدار خاصیت DataRowVersion در بسیاری از حالات وابسته به مقدار خاصیت DataRowState است. در حقیقت با فراخوانی متدهایی بر شیء DataRow (و در برخی حالات DataTable)، خاصیت DataRowVersion به‌صورت خودکار در پشت پرده تغییر می‌کند. در ادامه، فهرستی از متدهایی را، که می‌توانند بر روی این خاصیت مؤثر واقع شوند، می‌بینید:

- **اگر متد** DataRow.BeginEdit() **را فراخوانی کنید و مقدار سطر را تغییر دهید، مقادیر** Current **و** Proposed **قابل دسترسی خواهند بود.**
- **با فراخوانی متد** DataRow.CancelEdit()**، مقدار** Proposed **پاک می‌شود.**
- **پس از فراخوانی** DataRow.EndEdit()**، مقدار** Proposed **برابر با مقدار** Current **می‌شود.**
- **پس از فراخوانی متد** DataRow.AcceptChanges()**، مقدار** Original **و** Current **یکسان می‌شوند. همین حالت، با فراخوانی متد** DataTable.AcceptChanges() **نیز رخ می‌دهد.**
- **پس از فراخوانی متد** DataRow.RejectChanges()**، مقدار** Proposed **نادیده گرفته می‌شود و نگارش** Current **مورد ارجاع قرار می‌گیرد.**

مطالب گفته شده در مورد این خاصیت، به‌ویژه با درنظر گرفتن اینکه یک DataRow ممکن است در هر لحظه دارای همه‌ی نگارش‌های یاد شده بوده و یا فاقد آنها باشد، امکان دارد قدری پیچیده به نظر بیایند. با همه‌ی اینها، با توجه به اینکه یک DataRow در هر لحظه سه کپی از داده‌ها را نگهداری می‌کند، ایجاد واسط کاربری که کاربر نهایی به‌وسیله‌ی آن بتواند بازدید و ویرایش داده‌ها را انجام دهد، و تصمیم خود را عوض کند و تغییرات را لغو نماید (roll back) و یا آنها را به‌صورت دائم اعمال کند (commit)، امر ساده‌ای است. در ادامه‌ی این فصل و همین‌طور در فصل‌های آینده‌ی کتاب، مثال‌های گوناگونی از پیاده‌سازی روش‌های بالا خواهید دید.

15-5 کار با شیء DataTable

کلاس DataTable دارای اعضای گوناگونی است که بسیاری از آنها، هم از نظر نام و هم از نظر کارکرد، مانند اعضای کلاس DataSet هستند. جدول 15-7 بسیاری از اعضای اصلی این کلاس، به غیر از اعضای مربوط به سطرها و ستون‌ها (DataRows & DataColumns) را به‌صورت خلاصه تشریح کرده است.

جدول 15-7

عضو کلاس DataTable	معنی و عملکرد
CaseSensitive	مشخص کننده‌ی این است که آیا مقایسه‌های رشته‌ای بر روی محتوای DataTable، به حروف بزرگ و کوچک حساس باشند یا نباشد. مقدار پیش‌فرض، false است.
ChildRelations	کلکسیونی از ارتباطات با جدول‌های فرزند، از DataTable جاری را باز می‌گرداند (در صورت موجود بودن).