

مرجع آموزشی

C# 2010

(جلد 2)

مهندس نادر نبوی

انتشارات پندار پارس

سرشناسه
عنوان و نام پدیدآور
مشخصات نشر
مشخصات ظاهری
شابک
وضعیت فهرست نویسی
یادداشت
یادداشت
عنوان روی جلد
موضوع
رده بندی کنگره
رده بندی دیویی
شماره کتابشناسی ملی

نبوی، نادر، ۱۳۴۰ -
مرجع آموزشی C# 2010 / نادر نبوی.
تهران: پندار پارس: ۱۳۹۲ -
ج ۲: مصور، جدول، ۴۰ یک لوح فشرده.
دوره 978-600-6529-07-3 : ج ۱: 978-600-6529-05-9 : ج ۲: 978-600-6529-06-6 : ص ۵۲۸: ۲۳۸۰۰۰ ریال

فیبا
ج ۲ (چاپ اول: ۱۳۹۱) (فیبا).
عنوان روی جلد: مرجع آموزشی برنامه نویسی به زبان C# 2010 با نگاهی به تازه های C# 2012.
مرجع آموزشی برنامه نویسی به زبان C# 2010 با نگاهی به تازه های C# 2012.
سی شارپ (زبان برنامه نویسی کامپیوتر)
س ۷۳/۷۶QA / س ۱۳۹۰۹۵ ۲ن
۱۳۳/۰۰۵
۲۶۲۱۱۱۳



نام کتاب : مرجع آموزشی C# 2010 با نگاهی به تازه های C# 2012 (جلد ۲)

ناشر : انتشارات پندار پارس

تالیف : نادر نبوی

چاپ نخست : بهار ۹۲

شمارگان : ۱۰۰۰ نسخه

طرح جلد : فرزانه روزبهانی

لیتوگرافی : ترام سنج

چاپ، صحافی : جاویدنو، خیام

قیمت : ۲۳۸۰۰ تومان با DVD رایگان شابک : 978-600-6529-06-6 دوره: 3-07-6529-600-978



* هرگونه کپی برداری، تکثیر و چاپ کاغذی یا الکترونیکی از این کتاب بدون اجازه ناشر تخلف بوده و پیگرد قانونی دارد *

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 16 و 17 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 نگاهت 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 سپاسگزارم.

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

639	فصل 15: ADO.NET بخش دوم، لایه‌ی غیرمتصل
640	15-1 درک کارکرد لایه‌ی غیر متصل ADO.NET
641	15-2 آشنایی با نقش DATASET
642	15-2-1 خاصیت‌های کلیدی DATASET
643	15-2-2 متدهای کلیدی شیء DATASET
644	15-2-3 ایجاد DATASET
645	15-3 کار با ستون‌ها DATACOLUMNS
647	15-3-1 ایجاد شیء DATACOLUMN
647	15-3-2 استفاده از ستون AUTOINCREMENT
648	15-3-3 اضافه کردن اشیاء DATACOLUMN به شیء DATATABLE
649	15-4 کار با سطرها DATAROWS
651	15-4-1 آشنایی با خاصیت ROWSTATE
653	15-4-2 آشنایی با خاصیت DATAROWVERSION
654	15-5 کار با شیء DATATABLE
656	15-5-1 افزودن اشیاء DATATABLE به DATASET
656	15-5-2 دسترسی به داده‌های جدول‌ها DATASET
658	15-5-3 پردازش DATATABLE توسط DATATBLEREADER
659	15-5-4 سریال‌سازی اشیاء DATATABLE و DATASET به‌صورت XML
661	15-5-5 سریال‌سازی اشیاء DATATABLE و DATASET به‌صورت باینری
662	15-6 پیاده‌سازی روش غیر متصل در کار با پایگاه داده
662	15-6-1 ایجاد پایگاه داده TRADE
664	15-6-2 تعریف پروژه‌ی خرید و فروش
665	15-6-3 ایجاد پروژه‌های برنامه‌ی TRADE
666	15-6-4 ایجاد متدهای عمومی لایه‌ی غیر متصل
668	15-6-5 ایجاد متدهای لازم برای دریافت گزارش‌ها
669	15-6-6 ایجاد متدهای لازم برای آگاهی از وضعیت جدول
669	15-6-7 ایجاد متدهای لازم برای انجام محاسبات در جدول
670	15-6-8 ایجاد متدهایی برای تغییر جدول
672	15-6-9 ایجاد متد جست‌وجو در جدول‌ها

673	15-6-10 ایجاد لایه میانی یا منطقی برنامه
675	15-6-11 ایجاد رابط کاربر پروژهی TRADE
683	فصل 16: ایجاد برنامه‌های کاربردی ویندوز، بخش نخست
684	16-1 فضاهای نامی فرم‌های ویندوز
685	16-2 ایجاد یک پروژه‌ی ساده‌ی ویندوز، کدنویسی محض
688	16-2-1 استفاده از خاصیت CONTROLS
691	16-2-2 نقش SYSTEM.EVENTHANDLER و SYSTEM.EVENTARGS
692	16-3 پروژه‌های فرم‌های ویندوز در ویژوال استدیوی 2010
693	16-3-1 عنصرهای تشکیل دهنده‌ی یک پروژه‌ی ویندوز
696	16-3-2 کالبد شکافی کد فرم
698	16-3-3 کالبد شکافی کلاس PROGRAM
698	16-3-4 طراحی بصری یک منو
701	16-4 کالبد شکافی فرم
703	16-4-1 بررسی کارکرد کلاس CONTROL
706	16-4-2 بررسی کارکرد کلاس FORM
709	16-4-3 بررسی چرخه‌ی عمر فرم
712	16-5 کار با کنترل‌ها در پروژه‌های ویندوز
712	16-5-1 بررسی دوباره‌ی خصوصیات مهم کنترل‌های فرم
713	16-5-2 چیدمان کنترل‌ها روی فرم
716	16-5-3 کنترل‌های TEXTBOX و LABEL و BUTTON
719	16-5-4 کنترل‌های PANEL و GROUPBOX
722	16-5-5 کنترل‌های RADIOBUTTON و CHECKBOX
732	16-5-6 کنترل PICTUREBOX
736	16-5-7 کنترل TOOLTIP
738	16-5-8 کنترل NUMERICUPDOWN
741	16-5-9 کار با رویدادهای ماوس
745	16-5-10 کار با رویدادهای صفحه کلید
748	16-6 ایجاد یک برنامه‌ی کامل ویندوز
748	16-6-1 ایجاد منوی اصلی برنامه
749	16-6-2 ایجاد کلاس SHAPEDATA
750	16-6-3 ایجاد نوع SHAPEPICKERDIALOG

752	16-6-4 افزودن انواع مورد نیاز کلاس اصلی برنامه
752	16-6-5 برنامه‌ی منوی TOOLS
754	16-6-6 ذخیره و بازیابی ترسیمات
756	جمع‌بندی فصل شانزدهم
757	فصل 17: ایجاد برنامه‌های کاربردی ویندوز، بخش دوم
757	17-1 منوها
768	17-2 کنترل LINKLABEL
771	17-3 کنترل LISTBOX
775	17-4 کنترل CHECKLISTBOX
779	17-5 کنترل COMBOBOX
783	17-6 کنترل TREEVIEW
790	17-7 کنترل LISTVIEW
797	17-8 کنترل TABCONTROL
802	17-9 ایجاد برنامه‌هایی با واسط چند سندی (MDI)
811	17-10 ایجاد کنترل‌های شخصی
811	17-10-1 متد ONPAINT()
812	17-10-2 ایجاد کنترل‌های جدید
813	17-10-3 ایجاد کنترل ساعت
815	17-10-4 استفاده از کنترل‌های شخصی، در پروژه‌های دیگر
816	جمع‌بندی فصل هفدهم
817	فصل 18: پروژه‌های ویندوز و لایه‌ی غیرمتصل ADO.NET
818	18-1 آماده‌سازی پایگاه داده
819	18-2 مقیدسازی داده‌ها
819	18-2-1 شیء CURRENCYMANAGER
820	18-3 مقیدسازی کنترل TEXTBOX
825	18-4 مقیدسازی کنترل DATAGRIDVIEW
825	18-4-1 مقیدسازی DATAGRIDVIEW به اشیاء
827	18-4-2 مقیدسازی DATAGRIDVIEW به DATATABLE
829	18-4-3 ایجاد فرم والد-فرزند به‌وسیله‌ی DATAGRIDVIEW
831	18-5 مقیدسازی کنترل‌های LISTBOX و COMBOBOX
835	18-6 پیاده‌سازی پروژه‌ی TRADE در محیط ویندوز

838	18-6-1 ایجاد فرم ورود اطلاعات مشتری
840	18-6-2 ایجاد فرم ورود اطلاعات محصولات
843	18-6-3 نمایش خریدهای مشتری
849	18-6-4 ایجاد فرم ویرایش کامل محصولات
852	18-6-5 ایجاد نماهای اصلی و فرعی
855	فصل 19: LINQ بر SQL
855	19-1 نقش کلاس‌های موجودیت‌ها
856	19-2 بررسی نقش نوع DATACONTEXT
857	19-2-1 مثال ساده‌ای از LINQ بر SQL
859	19-2-2 ارث‌بری از کلاس DATACONTEXT
861	19-2-3 بررسی صفات [TABLE] و [COLUMN]
862	19-3 ایجاد کلاس‌های موجودیت‌ها به وسیله‌ی ویژوال استدیو 2010
863	19-3-1 بررسی کد ایجاد شده به وسیله‌ی ویژوال استدیو
866	19-4 تعریف روابط میان موجودیت‌ها
867	19-5 بررسی کلاس مشتق شده از DATACONTEXT
869	19-6 مقیدسازی DATAGRIDVIEW با LINQ بر SQL
870	19-7 پیاده سازی پروژهی TRADE با LINQ بر SQL
871	19-8 ایجاد فرم مشاهده‌ی مشخصات محصولات
873	19-9 اجرای روال‌های نخیره شده با LINQ بر SQL
877	19-10 افزودن رکورد به وسیله‌ی LINQ بر SQL
880	19-11 ویرایش رکوردها به وسیله‌ی LINQ بر SQL
883	19-12 حذف رکوردها به وسیله‌ی LINQ بر SQL
885	جمع‌بندی فصل نوزدهم
887	فصل 20: کار با ENTITY FRAMEWORK، بخش نخست
887	20-1 بازنگری روش‌های دسترسی به داده‌ها
888	20-2 بررسی رویکردهای شیء‌گرا و رابطه‌ای
889	20-3 استفاده از کلاس‌ها در سازمان‌دهی داده‌ها
890	20-3-1 استفاده از کلاس‌ها در نمایش داده‌ها
895	20-3-2 حرکت از یک کلاس منفرد به مدل‌سازی اشیاء
896	20-4 بررسی ناهمگونی مدل‌سازی ارتباطی/شیء‌گرا
896	20-4-1 ناهمگونی انواع داده

897	20-4-2 ناهمگونی ارتباطها
899	20-4-3 ناهماهنگی تعداد عناصر
900	20-4-4 ناهمگونی ارث‌بری
902	20-5 ORM و ENTITY FRAMEWORK
902	20-5-1 کارکرد ORM و جایگاه EF
904	20-6 چگونگی دسترسی به داده‌ها به وسیله‌ی EF
905	20-6-1 بررسی دقیق‌تر EDM
907	20-7 مدل داده‌ی موجودیت‌ها EDM
908	20-7-1 ایجاد EDM
909	20-7-1-1 ایجاد EDM با روش آغاز از پایگاه داده
914	20-7-1-2 ایجاد EDM با روش آغاز از مدل
918	20-7-1-3 ایجاد پایگاه داده
921	20-7-2 مدیریت ارتباطها
922	20-8 محیط طراحی و EDM
922	20-8-1 محیط طراحی
924	20-8-2 پنجره‌ی جزئیات نگاشت
925	20-8-3 موجودیت‌ها
927	20-8-4 صفت‌های منفرد
928	20-8-5 انواع ترکیبی
928	20-8-6 ایجاد انواع ترکیبی
930	20-8-7 کلیدهای خارجی و ارتباطها
932	20-8-8 صفات حرکتی میان موجودیت‌ها
933	20-9 بررسی دقیق‌تر کد EDM، نوع <T> OBJECTSET
936	جمع‌بندی فصل بیستم
937	فصل 21: کار با ENTITY FRAMEWORK، بخش دوم
937	21-1 ساختار برنامه، مدل و پایگاه داده
938	21-1-1 پایگاه داده ORDERIT
941	21-1-2 ساختار برنامه
942	21-1-3 ساختار مدل
943	21-1-4 ایجاد نوع ترکیبی CONTACTINFO
945	21-2 کوئری در EF، نگاه نخست

945	21-2-1 روش بهره‌گیری از عبارات کوئری
950	21-2-2 مروری دوباره بر CONTEXT
950	21-2-3 استفاده از روش مبتنی بر متد (عبارات لاندا)
952	21-2-3 انتخاب زبان کوئری ESQL یا LINQ
954	21-3 مروری بر کلاس OBJECTCONTEXT
955	21-3-1 کلاس OBJECTSTATEENTRY
956	21-3-2 ذخیره‌ی تغییرات موجودیت‌ها
957	21-4 ویرایش موجودیت‌ها
960	21-5 افزودن موجودیت‌ها
962	21-5-1 افزودن موجودیت‌ها، روش دوم
963	21-6 حذف موجودیت‌ها
965	21-6-1 حذف موجودیت‌ها، کلاس ENTITYKEY
966	21-7 بررسی عملگرهای LINQ بر موجودیت‌ها
966	21-7-1 فیلتر کردن داده‌ها، عملگر WHERE
971	21-7-2 انعکاس نتایج
974	21-7-3 گروه‌بندی داده‌ها
982	21-7-4 مرتب‌سازی نتایج
985	21-8 کاربرد توابع
985	21-8-1 توابع استاندارد
986	21-8-2 توابع پایگاه داده
987	21-8-3 کاربرد مستقیم عبارات SQL
989	21-8-4 کار با پارامترهای کوئری
990	21-9 واکنشی موجودیت‌ها
990	21-9-1 بارگذاری بی‌درنگ
993	21-9-2 بارگذاری تدریجی
994	21-10 نگاشت روال‌های ذخیره شده در EDM
995	21-10-1 وارد کردن روال ذخیره شده به EDM
998	21-10-2 واکنشی نتایج روال‌های ذخیره شده
999	21-10-2-1 روال‌های ذخیره شده با خروجی همسان با موجودیت
1003	21-10-2-2 روال‌های ذخیره شده با خروجی ناسازگار با موجودیت‌ها
1007	21-10-2-3 روال‌های ذخیره شده با خروجی اسکالر

1010	21-11 کار با نماها در EF
1011	21-11-1 کار با نما مانند یک جدول
1013	21-11-2 ایجاد کوئری معرف
1016	21-12 کار با توابع تک مقداری در پایگاه داده
1019	21-13 کار با توابع دارای خروجی جدول
1021	21-14 به‌روزرسانی داده‌ها به‌وسیله‌ی روال‌های ذخیره شده
1021	21-14-1 به‌روز رسانی یک موجودیت به‌وسیله‌ی روال ذخیره شده
1029	فصل 22: ENTITY SQL, مسئله‌ی همزمانی و تراکنش‌ها
1029	22-1 ساختار برنامه، مدل و پایگاه داده
1030	22-1-1 پایگاه داده PRIVATECLASSESDATABASE
1031	22-1-2 ساختار برنامه
1032	22-1-3 ساختار مدل
1033	22-2 آشنایی با کوئری پایه در ESQL
1035	22-3 فیلترکردن داده‌ها
1036	22-4 کار با جدول‌ها/موجودیت‌های مرتبط
1038	22-5 صفحه‌بندی نتایج
1039	22-6 کاربرد انعکاس در ESQL
1041	22-7 گروه‌بندی داده‌ها در ESQL
1044	22-7-1 فیلترکردن داده‌های گروه‌بندی شده
1045	22-8 مرتب‌سازی داده‌ها
1045	22-8-1 مرتب‌سازی بر اساس ارتباطها
1046	22-9 استفاده از متدهای کوئری‌ساز
1048	22-9-1 فرآیند زنجیره‌ای
1049	22-9-2 متدهای کوئری‌ساز و متدهای LINQ بر موجودیت‌ها
1050	22-9-3 استفاده از پارامترها برای جلوگیری از تزریق SQL
1051	21-10 مدیریت همزمانی
1052	22-10-1 راه حل بدبینانه‌ی کنترل همزمانی
1053	22-10-2 راه‌حل خوش‌بینانه‌ی کنترل همزمانی
1054	22-10-3 مدیریت خوش‌بینانه‌ی همزمانی در EF
1055	22-10-3-1 سناریوی متصل
1056	22-10-3-2 سناریوی منفصل

1058	22-10-4 مدیریت استثنای همزمانی
1061	22-11 تراکنش‌ها و EF
1064	22-12 ترکیب ESQL و LINQ بر موجودیت‌ها
1067	فصل 23: آشنایی با WCF
1067	23-1 آشنایی با معماری مبتنی بر سرویس
1068	23-1-2 WCF و معماری مبتنی بر سرویس
1070	23-2 نقش سرویس‌های وب WEB SERVICES
1072	23-2-1 مثالی از یک وب‌سرویس در .NET
1074	23-3 نقش WCF
1074	23-3-1 بررسی ویژگی‌های WCF
1075	23-3-2 بررسی اسمبلی‌های کلیدی WCF
1076	23-4 ایجاد پروژه‌ی WCF در ویژوال استدیو
1077	23-5 اجزای اصلی برنامه‌ی کاربردی WCF
1079	23-6 اصول ABC در WCF
1079	23-6-1 قراردادهای WCF
1080	23-6-2 قیدهای WCF
1081	23-6-2-1 قیود مبتنی بر HTTP
1082	23-6-2-2 قیود مبتنی بر TCP
1083	23-6-3 آدرس‌ها در WCF
1085	23-7 ایجاد یک سرویس WCF
1086	23-7-1 صفت [SERVICECONTRACT]
1087	23-7-2 صفت [OPERATIONCONTRACT]
1088	23-7-3 کلاس‌ها به‌عنوان قرارداد سرویس
1089	23-8 میزبانی سرویس WCF
1090	23-8-1 تنظیمات ABC در فایل پیکربندی
1091	23-8-2 به‌کارگیری نوع SERVICEHOST در برنامه‌ی میزبان
1091	23-8-3 تعیین آدرس‌های پایه
1093	23-8-4 بررسی کلاس SERVICEHOST
1095	23-8-5 بررسی عنصر <SYSTEM.SERVICEMODEL>
1097	23-8-6 امکان تبادل فراداده
1099	23-9 ایجاد برنامه‌ی مشتری WCF

1101	23-10	پیگر بندی بر اساس پروتکل TCP
1103	23-11	نقاط پایانی پیش فرض در WCF 4.0 به بعد
1104	23-12	ارائه‌ی یک سرویس WCF با چندین قید
1105	23-13	پیگر بندی پیش فرض رفتار MEX در WCF 4.0
1108	23-14	آشنایی با پروژه‌های WCF SERVICE LIBRARY
1109	23-14-1	آزمایش سرویس با WCFTESTCLIENT.EXE
1109	23-14-2	ویرایش فایل پیگر بندی توسط SVCCONFIGEDITOR.EXE
1115	23-15	طراحی قراردادهای داده در WCF
1116	23-15-1	استفاده از پروژه‌ی WCF SERVICE APPLICATION
1124	23-16	نقش فایل *.SVC
1125	23-17	نکاتی در مورد فایل WEB.CONFIG
1125	23-18	افزودن سرویس‌ها، یک سرویس با چند قرارداد
1128	23-18-1	افزودن سرویس‌ها، یک قرارداد با چندین سرویس
1132	23-19	میزبانی سرویس در IIS
1137	23-20	استفاده از سرویس میزبانی شده در IIS
1139	23-21	آشنایی با سرویس‌های داده در WCF (WDS)
1139	23-21-1	پروژه‌ی سرویس‌های داده‌ی WCF
1141	23-21-2	ایجاد سرویس داده
1142	23-21-3	اجرای کوئری بر سرویس داده
1144	23-21-4	امنیت در WDS
1144	23-21-5	برنامه‌ی مشتری، ایجاد کلاس پروکسی
1145	23-21-5-1	افزودن آیتم جدید با استفاده از کلاس‌های پروکسی
1145	23-21-5-2	ویرایش یک آیتم با استفاده از کلاس‌های پروکسی
1146	23-21-5-3	حذف یک آیتم با استفاده از کلاس‌های پروکسی

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 فراخوان¹ 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 محلی² انجام می‌شوند. نکته در اینجا است که این داده‌های محلی، در حقیقت یک کپی از داده‌های اصلی موجود در پایگاه داده هستند.

گرچه کاملاً 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 برگرفته از جدول یا جدول‌های یک پایگاه داده پر می‌شود؛ و معمولاً این کار به‌وسیله‌ی شیء `data adapter`، یک فراهم‌کننده‌ی 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، اصطلاحاً `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 DataSet فراهم می‌آورند. این مورد را نیز، به دلیل استفاده از پروژهای 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 اینجا 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، محدودیت‌های مربوط به فیلدها¹، کلیدهای اصلی²، نماها³ 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 کاملاً 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 حافظه‌ای⁴ 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

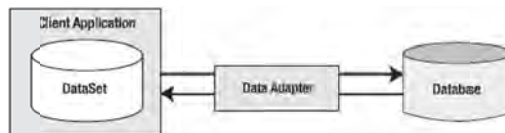
³ views

⁴ in-memory queries

طریق شیء data reader پردازش نمی‌شوند، بلکه این اشیاء از شیء دیگری به نام 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، به صورت خودکار با شیء data adapter است؛ و برای بالا بردن کارایی برنامه، مدت زمان اتصال تا جای ممکن به حداقل کاهش می‌یابد. پس از اینکه برنامه‌ی فراخوان، محتویات DataSet را به دست آورد، اتصال آن به طور 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 به صورت صریح DataSet را برای به روز رسانی پایگاه به شیء data adapter (معمولاً همان data adapter 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 به صورت دستی (در این مرحله، کاری با پایگاه داده و جداول آن نداریم) خواهد بود. پس از به دست 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. به صورت دقیق‌تر، DataSet کلاسی است دارای سه کلکسیون اصلی (شکل 2-15).



شکل 2-15

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 چپ که دارای شرط خاصی در جدول سمت یک باشند، یا برعکس). البته در این فصل، با همه‌ی این اعمال، آشنا خواهید شد.

15-2-1-1 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. معمولاً این مقدار را، به‌عنوان پارامتر سازنده مشخص می‌کنید.
EnforceConstraints	خاصیتی منطقی (قابل خواندن و نوشتن)، مشخص‌کننده‌ی اینکه آیا قوانین پایگاه

معنی و مفهوم	خصوصیت
داده در مورد حفظ جامعیت، هنگام به روز رسانی رکوردها رعایت شود یا نه (مقدار پیش فرض، true است).	
مقداری را به عنوان اینکه آیا خطایی در سطرهای جداول (DataTables) درون DataSet موجود هست یا نه، باز می گرداند.	HasErrors
به وسیله ی این خاصیت، می توان چگونگی تثبیت (سریال سازی) محتوای DataSet را مشخص کرد (به صورت باینری یا XML که مقدار پیش فرض برای فایل خروجی، XML است).	RemotingFormat

15-2-2 متدهای کلیدی شیء DataSet

متدهای DataSet، در ارتباط با خواص اشاره شده در جدول 15-1 کار می کنند. افزون بر امکان کار با فایل های متنی، DataSet دارای متدهایی برای کپی محتویات جدول هایش، حرکت میان جدول های داخلی و تعیین نقاط آغاز و پایان به روز رسانی های دسته جمعی¹ است. برخی از متدهای مهم، در جدول 15-2 لیست شده اند.

جدول 15-2

معنی و عملکرد	متد
همه ی تغییرات انجام شده را، از زمان بار شدن DataSet به حافظه و یا پس از آخرین اجرای متد AcceptChanges()، ذخیره و تثبیت می کند (معادل commit در پایگاه داده).	AcceptChanges()
DataSet را با حذف همه ی سطرهای جدول های آن، کاملاً خالی می کند.	Clear()
یک کپی از ساختار DataSet (بدون داده ها)، به همراه همه ی جدول ها و ارتباطات آنها ایجاد می کند.	Clone()
ساختار یک DataSet را به همراه داده های آن کپی می کند.	Copy()
یک کپی از DataSet را شامل همه ی تغییرهای انجام شده در آن، از زمان بار شدن به حافظه و یا آخرین اجرای متد AcceptChanges() باز می گرداند. این متد به شکل های مختلفی سرپارگذاری شده است، بنابراین امکان می دهد تنها سطرهای	GetChanges()

¹ batch updates

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

15-2-3 ایجاد DataSet

اکنون که درک بهتری از یک DataSet و چگونگی کارکرد آن به دست آورده اید، پروژه ی کنسول جدیدی به نام 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 شناسه ی عددی منحصر به فرد 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 عددی¹ 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.

3-15 کار با ستون‌ها DataColumn

نوع DataColumn نشان دهنده‌ی یک ستون، درون شیئی از نوع DataTable است. به بیانی کلی‌تر، اطلاعات مربوط به مجموعه‌ی همه‌ی DataColumn‌های یک DataTable، مشخص‌کننده‌ی طرح جدول² آن می‌باشد. به‌عنوان مثال اگر بخواهید جدول Inventory از پایگاه داده‌ی AutoLot (که در جلد نخست کتاب ایجاد کردید) را به این‌صورت مدل‌سازی کنید، 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 محدودیت‌هایی³ 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 فیلد⁴ CarID 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

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

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.

معنی و مفهوم	خصوصیت
تکراری (مانند مقادیر کلید اصلی جدول) باشند، مفید خواهد بود.	
این خاصیت، امکان تعیین عنوانی را برای ستون فراهم می‌آورد. کاربرد آن، به‌خصوص در جایی که بخواهید عنوانی ملموس به نام فیلدی از جدول پایگاه داده بدهید، مفید خواهد بود.	Caption
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 اصلاً در فایل XML درج نشود و از آن صرف نظر گردد.	ColumnMapping
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).	ColumnName
این خاصیت، تعیین کننده‌ی نوع داده‌ی ستون است.	DataType
این خاصیت خواندنی و نوشتنی، تعیین کننده‌ی مقدار پیش فرض ستون در هنگام افزودن سطر جدید است.	DefaultValue
این خصوصیت نوشتنی و خواندنی، عبارتی را برای فیلتر کردن سطرها، محاسبه‌ی مقدار ستون (ستون محاسباتی) و یا ایجاد یک ستون جمععی (میانگین، کمینه، بیشینه و دیگر) مشخص می‌کند.	Expression
این خاصیت، تعیین کننده‌ی موقعیت عددی ستون در کلکسیون ستون‌های DataTable است.	Ordinal
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.	ReadOnly
جدولی (DataTable) را که ستون متعلق به آن است باز می‌گرداند.	Table
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.	Unique

15-3-1 ایجاد شیء DataColumn

در ادامه ی پروژه ی SimpleDataSet (و البته برای آشنایی بیشتر با DataColumn)، فرض کنید می خواهیم جدول Inventory را مدل سازی کنیم. با توجه به اینکه فیلد CarID، کلید اصلی جدول است، باید شیء DataColumn مربوط به آن، به صورت فقط خواندنی، یکتا و غیر تهی (non-null) تنظیم شود. در مرحله ی 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 به عنوان تنها پارامتر ورودی خود می پذیرد:

```
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 می تواند مفید باشد که می خواهید عنوانی غیر از عنوان اصلی فیلد در پایگاه داده را برای آن نمایش دهید؛ زیرا اسامی فیلدها در جدول های پایگاه داده (مانند au_fname) معمولاً برای برنامه نویسی مناسب تر هستند، تا نمایش (مثلاً به صورت قابل فهم تر Author First Name). در اینجا هم به همین منظور خاصیت Caption را برای فیلد PetName مقداردهی می کنید، زیرا Pet Name برای کاربر نهایی واضح تر است تا PetName.

15-3-2 استفاده از ستون Autoincrement

یکی از ویژگی هایی که می توان 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، به صورت خودکار تغییر کند. به کار 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`، به عنوان مقدار آغاز افزایش و از `step`، به عنوان 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، به صورت 0,1,2,3,... مقدار می‌گیرد.

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

معمولا اشیاء `DataColumn`، به صورت 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 `DataColumn`‌ها به وسیله‌ی خاصیت `Columns` به کلکسیون ستون‌های آن (کلکسیون `Columns`)، اضافه می‌شوند:

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

تا اینجا 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 DataRow

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 15-4، برخی از اعضای مهم کلاس DataRow را لیست کرده است.

جدول 15-4

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

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

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 اینجا 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 طرح‌ها¹ 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.

1-4-15 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 5-15 آمده است، نسبت داد.

جدول 5-15

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

¹ schema

هنگام دستکاری سطرهای یک DataTable به وسیله‌ی کدنویسی، خاصیت RowState به صورت خودکار مقداره‌ی می‌شود. برای آزمایش گفته‌ی بالا، متد جدیدی به کلاس Program اضافه می‌کنیم که تغییراتی را به صورت یک شیء محلی¹ (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.

2-4-15 آشنایی با خاصیت DataRowVersion

صرف نظر از اینکه وضعیت جاری یک سطر را می‌توان با خاصیت RowState به دست 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 به وجود 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، به دست 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

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