Microsoft

The Definitive Guide to DAX

Business intelligence with Microsoft Power BI, SQL Server Analysis Services, and Excel

راهنمای جامع دکس

هوش تجاری با

Microsoft Power BI

SQL Server Analysis Services

و

Excel

SECOND EDITION

ویرایش دوم

Marco Russo and Alberto Ferrari

Sample files on the web

راهنمای جامع دکس: هوش تجاری با *Microsoft Power BI*، *SQL Server Analysis Services* و *Excel*

ویراش دوم

مارکو روسو و آلبرتو فراری

فهرست مطالب در یک نگاه

فهرست مطالب

پیشگفتار

ممکن ا ست شما نام ما را ندانید. ما روزهای عمر خود را صرف نو شتن کد نرم‌افزاری می‌کنیم که شما در کار روزانه خود اســـتفاده می‌کنید: ما بخشـــی از تیم توســـعه Power BI، SQL Server Analysis Services، و... هســـتیم، بله ما جزو نویسندگان زبان دکس و موتور VertiPaq هستیم.

زبانی که قرار است با استفاده از این کتاب فرا بگیرید، ساخته دست ماست. ما سال‌ها روی این زبان کار کرده‌ایم، موتور را بهینه‌سازی کرده‌ایم، راه‌هایی برای بهبود بهینه‌ساز یافته و سعی کرده‌ایم دکس را زبانی ساده، تمیز و سالم بسازیم تا زندگی شما به عنوان یک تحلیلگر داده آسان‌تر و سازنده‌تر باشد.

اما آهای، این قرار ا ست پیشگفتار یک کتاب با شد، پس دیگر در مورد ما نیست! چرا ما برای کتابی که مارکو و آلبرتو، بچه‌های SQLBI منتشر کرده‌اند، مقدمه بنویسیم؟ خوب، زیرا زمانی که شما شروع به یادگیری دکس می‌کنید، قبل از یافتن مقالات نوشته شده توسط آن‌ها، چند کلیک و جست‌وجو در وب لازم است. شما شروع به خواندن مقالات آن‌ها و یادگیری این زبان می‌کنید و امیدواریم که زحمات ما را ارزشمند بدانید. سال‌ها پیش با آن‌ها آشنا شده‌ایم، ما به خاطر دانش عمیق آن‌ها از SQL Server Analysis Services بسیار حیرت زده شدیم. هنگامی که ماجراجویی دکس شروع شد، آن‌ها جزو اولین کسانی بودند که این موتور و زبان جدید را یاد گرفته و پذیرفتند.

مقالات، اوراق و پست‌های وبلاگی که آن‌ها منتشر می‌کنند و در وب به اشتراک می‌گذارند منبع یادگیری برای هزاران نفر شده ا ست. ما کد را می‌نویسیم، اما زمان زیادی را صرف آموزش تو سعه‌دهندگان در مورد نحوه ا ستفاده از آن نمی‌کنیم؛ مارکو و آلبرتو کسانی هستند که دانش را در مورد دکس منتشر کرده‌اند.

کتاب‌های آلبرتو و مارکو جزو معدود کتاب‌های پرفروش در این زمینه هســـتند، و اکنون با این راهنمای جدید دکس، آن‌ها واقعاً یک کتاب بسیار مهم در مورد زبانی که ما نو شته‌ایم و آن را دو ست داریم خلق کرده‌اند. ما کد را می‌نویسیم، آن‌ها کتاب‌ها را ، و شما دکس را فرا می‌گیرید که قدرت تحلیلی بی‌سابقه‌ای را برای کسب‌وکار شما فراهم می‌کند. این چیزی است که ما دوست داریم: همه با هم به عنوان یک تیم—ما، آن‌ها و شما—کار کنیم تا بینش بهتری را از داده‌ها استخراج کنیم.

Marius Dumitru, Architect, Power BI CTO's Office
Cristian Petculescu, Chief Architect of Power BI
Jeffrey Wang, Principal Software Engineer Manager
Christian Wade, Senior Program Manager

قدردانی‌ها

نوشـــتن ویرایش دوم به یک ســـال تمام کار نیاز داشـــت، ســـه ماه بیشـــتر از ویرایش اول. این یک ســـفر طولانی و شگفت‌انگیزی بوده ا ست و با پیوند دادن مردم در سرا سر جهان در هر عرض جغرافیایی و منطقه زمانی توان ستیم نتیجه‌ای را که قصـــد خواندن آن را دارید، خلق کنیم. ما نام افراد زیادی را برای قدردانی بابت این کتاب داریم اما می‌دانیم نوشـــتن یک فهر ست کامل غیرممکن ا ست. بنابراین، از همه شما که در نو شتن این کتاب م شارکت کردید بسیار سپا سگزاریم—حتی اگر نمی‌دانســتید که این کار را انجام می‌دهید. نظرات وبلاگ، پســت‌های انجمن، بحث‌های ایمیلی، گفت‌وگو با شــرکت‌کنندگان و سخنرانان در کنفرانس‌های فنی، تجزیه‌وتحلیل سناریوهای مشتریان و بسیاری موارد دیگر برای ما مفید بوده است و بسیاری از افراد ایده‌های مهمی را در این کتاب ارائه کرده‌اند. علاوه بر این، از همه دان شجویان دوره‌های آموز شی خودمان ت شکر می‌کنیم: با آموزش به شما، ما بهتر شدیم!

گفتنی است، افرادی هستند که به دلیل مشارکت خاص آن‌ها باید شخصاً به نام آن‌ها اشاره کنیم.

می‌خواهیم با Edward Melomed شـروع کنیم: او الهام‌بخش ما بوده اسـت و احتمالاً سـفر خود را با زبان دکس بدون بحث پر شوری که چند سال پیش با او دا شتیم آغاز نمی‌کردیم و آن بحث با فهر ست مطالب اولین کتابمان در مورد Power Pivot به پایان رسید که روی دستمال نوشته شد.

می‌خواهیم از انتشارات مایکروسافت و افرادی که در این پروژه مشارکت داشتند تشکر کنیم: همه آن‌ها در روند نوشتن این کتاب به ما کمک کردند.

تنها کاری که طولانی‌تر از نوشـتن یک کتاب اسـت، مطالعه‌ای اسـت که باید برای آماده شـدن برای نوشـتن آن کتاب انجام دهید. گروهی از مردم که ما آن‌ها را (در کمال صـمیمیت) «خودی‌ها» می‌نامیم، به ما کمک کردند تا برای نوشـتن این کتاب آماده شـویم. چند نفر از مایکروسـافت نیز شـایسـته ذکر ویژه هسـتند، زیرا آن‌ها زمان گران‌بهای خود را صـرف آموزش مفاهیم مهم *Power BI* و دکس به ما کردند: آن‌ها Marius Dumitru، Jeffrey Wang، Akshai Mirchandani، Krystian Sakowski و Cristian Petculescu هستند. بچه‌ها! کمک شما بسیار باارزش بوده است.

همچنین می‌خواهیم از Amir Netz، Christian Wade، Ashvini Sharma، Kasper De Jonge و T. K. Anand برای مشارکت‌شان در بحث‌های زیادی که درباره محصول داشتیم تشکر کنیم. ما احساس می‌کنیم که آن‌ها در انتخاب‌های راهبردی که در این کتاب و در حرفه‌مان انجام دادیم، بسیار به ما کمک کردند.

ما می‌خواهیم به نام یک خانم ا شاره ویژه‌ای دا شته با شیم که در بهبود و تمیز کردن زبان انگلی سی ما کار باورنکردنی انجام داد. Claire Costa تمام د ست نو شته‌ها را ت صحیح و خواندن آن را بسیار آ سان کرد. Claire، کمک شما بسیار ارز شمند است—متشکریم!

آخرین اشـــاره ویژه به بازبین فنی ما می‌رســـد: Daniil Maslyuk هر یک از خط‌های کد، متن، مثال و مرجعی را که نو شته بودیم به دقت برر سی کرد. او هر نوع ا شتباهی را که از د ست ما در می‌رفت پیدا کرد. او به ندرت اظهارنظرهایی می‌کرد که نیاز به تغییر در کتاب ندا شت. نتیجه برای ما شگفت‌انگیز ا ست. اگر کتاب دارای خطاهای کمتری ن سبت به ن سخه خطی اولیه ما است، تنها به دلیل تلاش‌های Daniil می‌باشد. اگر باز هم دارای خطا باشد، البته که آن خطا تقصیر ماست.

خیلی ممنون، دوستان!

فهرست خطاها، به‌روزرسانی‌ها، و پشتیبانی از کتاب

ما تمام تلاش خود را برای اطمینان از صحت این کتاب و محتوای همراه آن انجام داده‌ایم. می‌توانید به به‌روزرسانی‌های این کتاب—در قالب فهرستی از خطاهای ارسالی و اصلاحات مربوط به آن‌ها—در آدرس https://MicrosoftPressStore.com/DefinitiveGuideDAX/errata دسترسی داشته باشید.

برای اطلاعات و پشتیبانی بیشتر از کتاب، لطفاً به https://MicrosoftPressStore.com/Support مراجعه کنید.

لطفاً توجه دا شته با شید که پ شتیبانی مح صول برای نرم‌افزار و سخت‌افزار Microsoft از طریق آدرس‌های قبلی ارائه نمی‌شود. برای راهنمایی در مورد نرم‌افزار یا سخت‌افزار Microsoft، به آدرس http://support.microsoft.com بروید.

در تماس باشید

اجازه دهید گفتگو را ادامه دهیم! ما در توییتر هستیم: http://twitter.com/MicrosoftPress.

مقدمه ویرایش دوم

وقتی تصمیم گرفتیم که زمان به‌روزرسانی این کتاب فرا رسیده است، فکر کردیم این کار آسانی است: از این گذشته، چیزهای زیادی در زبان دکس تغییر نکرده است و هسته نظری کتاب هنوز بسیار خوب بود. ما معتقد بودیم که تمرکز بیشتر ما بر روی به‌روزرسانی نماگرفت‌ها[1] از Excel به Power BI است، افزودن چند دست‌کاری اینجا و آنجا، و کار ما تمام می‌شود. چقدر اشتباه کردیم!

به محض آغاز به‌روزرسانی فصل اول، به سرعت متوجه شدیم که می‌خواهیم تقریباً همه چیز را بازنویسی کنیم. ما نه تنها در فصل اول، بلکه در هر صفحه از کتاب چنین احساسی داشتیم. بنابراین، این واقعاً یک نسخه دوم نیست؛ این یک کتاب کاملاً جدید است.

علت این نیست که این زبان یا ابزارها به شدت تغییر کرده‌اند. علت این است که در این چند سال اخیر ما—به عنوان نویسنده و معلم—بسیار پیشرفت کرده‌ایم، امیدوارم این پیشرفت در راستای بهتر شدن بوده باشد. ما دکس را به هزاران کاربر و توسعه‌دهنده در سراسر جهان آموزش داده‌ایم؛ به سختی با دانش‌آموزان خود کار کرده و همیشه برای بهترین شیوه توضیح موضوعات پیچیده تلاش کرده‌ایم. در نهایت، راه‌های مختلفی را برای توصیف زبانی که ما آن را دوست داریم، یافته‌ایم.

ما تعداد مثال‌ها را برای این نسخه افزایش دادیم و استفاده‌های عملی از این قابلیت‌ها را پس از آموزش مبانی نظری دکس نشان دادیم. ما سعی کردیم از سبک ساده‌تری استفاده کنیم، بدون اینکه از دقت کاسته شود. ما با ویراستار در مورد افزایش تعداد صفحات جنگیدیم، زیرا این مورد برای پوشش همه موضوعاتی که می‌خواستیم به اشتراک بگذاریم مورد نیاز بود. با این وجود، ما اصل کتاب را تغییر ندادیم: فرض می‌کنیم که هیچ دانش قبلی از دکس وجود ندارد، حتی اگر این کتاب برای توسعه‌دهندگان معمولی دکس نباشد. این کتاب برای افرادی است که واقعاً می‌خواهند این زبان را یاد بگیرند و درک عمیقی از قدرت و پیچیدگی دکس به دست آورند.

آری، اگر می‌خواهید از قدرت واقعی دکس استفاده کنید، باید برای یک سفر طولانی با ما آماده باشید، کتاب را از اول تا آخر بخوانید، و سپس دوباره آن را بخوانید، و به دنبال جزئیات زیادی باشید که در نگاه اول مشخص نبودند.

[1] – *Screenshots*

مقدمه ویرایش اول

ما میزان قابل توجهی محتوا در خصـــوص دکس تولید کرده‌ایم: کتاب‌هایی درباره Power Pivot و SSAS Tabular، پست‌های وبلاگ، مقالات، گزارشات رسمی[1]، و در نهایت کتابی که به الگوهای دکس اختصاص دارد. پس چرا باید کتاب دیگری درباره دکس بنویسیم (و امیدواریم شما آن را بخوانید)؟ آیا واقعاً چیزهای زیادی برای یادگیری در مورد این زبان وجود دارد؟ البته، ما فکر می‌کنیم که پاسخ قطعی آری است.

وقتی کتابی می‌نویسید، اولین چیزی که ویراستار می‌خواهد بداند تعداد صفحات آن است. دلایل بسیار خوبی برای اهمیت این موضـــوع وجود دارد: قیمت، مدیریت، تخصـــیص منابع و غیره. در پایان، تقریباً همه چیز در یک کتاب به تعداد صفحات باز می‌گردد. به عنوان نویسنده، این تا حدودی ناامید کننده است. در واقع، هر زمان که کتابی را می‌نویسیم، باید با دقت فضا را به توضیحات محصول (چه Power Pivot برای Microsoft Excel یا SSAS Tabular) و به توضیحات زبان دکس اختصاص دهیم. این موضوع همیشه این احساس تلخ را برای ما به وجود آورده است که صفحات کافی برای توضیح تمام آنچه می‌خواســتیم در مورد دکس آموزش دهیم در اختیار نداریم. به هر حال، شــما نمی‌توانید ۱,۰۰۰ صـفحه درباره Power Pivot بنویسید؛ کتابی با چنین اندازه‌ای برای هر کسی ترسناک خواهد بود.

به هر طریق، ما سال‌ها درباره SSAS Tabular و Power Pivot نوشتیم و پروژه کتابی را که کاملاً به دکس اختصاص داده شده بود را در کشو نگه داشتیم. زمانی که کشو را گشوده و تصمیم گرفتیم از انتخاب مواردی که باید در کتاب بعدی بگنجانیم اجتناب کنیم: می‌خواســتیم همه چیز را در مورد دکس بدون هیچ مصـالحه‌ای توضـیح دهیم. نتیجه آن تصـمیم این کتاب است.

در اینجا توضیحی در مورد نحوه ایجاد یک ستون محاسبه‌شده، یا اینکه از کدام کادر محاوره‌ای برای تنظیم یک ویژگی اســتفاده کنید، پیدا نمی‌کنید. این یک کتاب گام‌به‌گام نیســت که نحوه اســتفاده از Microsoft Visual Studio، Power BI یا Power Pivot برای Excel را به شــما آموزش دهد. در عوض، این یک فرو رفتن عمیق در زبان دکس اســت که از ابتدا شــروع می‌شود و سپس به جزئیات بسیار فنی در مورد نحوه بهینه‌سازی کد و مدل شما می‌رسد.

ما هر صفحه از این کتاب را در حین نوشتن آن دوست داشتیم. آنقدر مطالب را مرور کردیم که آن را حفظ شدیم. هر زمان که فکر می‌کردیم چیز مهمی وجود دارد به اضافه کردن محتوا ادامه می‌دادیم، بنابراین تعداد صفحات را افزایش می‌دادیم و هرگز چیزی را به سـبب اینکه دیگر هیچ صـفحه‌ای باقی نمانده اسـت حذف نمی‌کردیم. با انجام این کار، ما بیشــتر در مورد دکس یاد گرفتیم و از هر لحظه‌ای که برای انجام این کار صرف کردیم لذت بردیم.

اما یک چیز دیگر وجود دارد. چرا باید کتابی درباره دکس بخوانید؟

به واقع اگر پس از اولین نســخه آزمایشــی Power Pivot یا Power BI شــما به این موضــوع فکر کرده‌اید. شــما تنها نیستید؛ اولین بار هم که ما آن را بررسی کردیم همین فکر را داشتیم. دکس بسیار آسان است! بسیار شبیه به Excel می‌باشد! علاوه بر این، اگر قبلاً زبان‌های برنامه نویســـی و/یا زبان کوئری دیگری را فرا گرفته‌اید، احتمالاً به یادگیری یک زبان جدید از طریق مشاهده نمونه‌هایی از ترکیب[2]، عادت کرده‌اید و الگوهایی را که پیدا می‌کنید با آن‌هایی که از قبل می شناسید مطابقت می‌دهید. ما مرتکب این اشتباه شدیم و مایلیم شما از انجام آن اجتناب کنید.

دکس یک زبان قدرتمند است که در تعداد فزاینده‌ای از ابزارهای تحلیلی استفاده می‌شود. آن بسیار قدرتمند است، اما شامل تعدادی مفهوم است که درک آن‌ها با استدلال استقرایی دشوار است. به عنوان مثال، زمینه ارزیابی موضوعی است که نیازمند یک رویکرد قیاسی است: شما با یک نظریه شروع می‌کنید و سپس چند نمونه را مشاهده می‌کنید که نشان می‌دهد

---

[1] - *White Papers*

[2] - *Syntax*

این نظریه چگونه کار می‌کند. استدلال قیاسی رویکرد این کتاب است. ما می‌دانیم که تعدادی از افراد از یادگیری به این روش خوششان نمی‌آید، زیرا آن‌ها رویکرد عملی‌تری را ترجیح می‌دهند—یاد می‌گیرند که چگونه مسائل خاص را حل کنند، و سپس با تجربه و تمرین، نظریه اصلی را با استدلال استقرایی درک می‌کنند. اگر به دنبال چنین رویکردی هستید، این کتاب برای شما مناسب نیست. ما کتابی در مورد الگوهای دکس نوشتیم، پر از مثال‌ها و بدون هیچ توضیحی در مورد اینکه چرا یک فرمول کار می‌کند یا چرا یک روش خاص برای کدنویسی بهتر است. آن کتاب منبع خوبی برای نسخه‌برداری و چسباندن فرمول‌های دکس است. هدف این کتاب در اینجا متفاوت است: قادر ساختن شما به تسلط و استادی دکس. تمام مثال‌ها رفتار دکس را نشان می‌دهند. آن‌ها مشکل خاصی را حل نمی‌کنند. اگر فرمول‌هایی یافتید که می‌توانید آن‌ها رادر مدل‌های خود دوباره استفاده کنید، خوش به حال شما. با این حال، همیشه به یاد داشته باشید که این فقط یک اثر جانبی است، نه هدف آن مثال. در نهایت، همیشه هر یادداشتی را بخوانید تا مطمئن شوید که در کدهای استفاده شده در مثال‌ها هیچ مشکلی وجود ندارد. در راستای اهداف آموزشی، ما اغلب از کدهایی استفاده کرده‌ایم که لزوماً روش بهینه[1] نبودند.

ما واقعاً امیدواریم که از گذراندن وقت با ما در این سفر زیبا برای یادگیری دکس لذت ببرید، حداقل به همان شکلی که ما از نوشتن آن لذت بردیم.

این کتاب مناسب چه کسانی است

اگر کاربر معمولی دکس هستید، این کتاب احتمالاً بهترین انتخاب برای شما نیست. بسیاری از کتاب‌ها مقدمه‌ای ساده در مورد ابزارهایی که دکس را پیاده‌سازی می‌کنند و خود زبان دکس ارائه می‌دهند، از ابتدا شروع می‌کنند و به سطح اولیه برنامه‌نویسی دکس می‌رسند. ما این را به خوبی می‌دانیم، زیرا برخی از آن کتاب‌ها را ما هم به رشته تحریر در آورده‌ایم!

از طرف دیگر، اگر در مورد دکس جدی هستید و واقعاً می‌خواهید تمام جزئیات این زبان زیبا را درک کنید، این کتاب شماست. ممکن است این کتاب اولین کتاب شما در مورد دکس باشد؛ در این صورت نباید انتظار داشته باشید که خیلی زود از پیشرفته‌ترین موضوعات بهره‌مند شوید. پیشنهاد می‌کنیم کتاب را از ابتدا تا انتها بخوانید و بعد از کسب تجربه، پیچیده‌ترین قسمت‌ها را دوباره بخوانید؛ این احتمال وجود دارد که برخی از مفاهیم در آن مرحله واضح‌تر شوند.

دکس برای افراد مختلف، برای اهداف مختلف مفید است: کاربران Power BI ممکن است نیاز به نوشتن فرمول‌های دکس در مدل‌های خود داشته باشند، کاربران Excel می‌توانند از دکس برای ایجاد مدل‌های داده Power Pivot استفاده کنند، متخصصان هوش تجاری[2] (BI) ممکن است نیاز به پیاده‌سازی کد دکس در راه‌حل‌های *BI* در هر اندازه‌ای داشته باشند. ما در این کتاب سعی کردیم اطلاعاتی را در اختیار همه این افراد مختلف قرار دهیم. برخی از محتواها (مخصوصاً بخش بهینه‌سازی) احتمالاً بیشتر از طرف متخصصان BI مورد هدف قرار می‌گیرند، زیرا دانش مورد نیاز برای بهینه‌سازی معیار دکس بسیار فنی است؛ اما ما معتقدیم که کاربران Power BI و Excel نیز باید دامنه عملکرد ممکن عبارات دکس را برای دستیابی به بهترین نتایج برای مدل‌های خود درک کنند.

در آخر ما می‌خواستیم کتابی برای مطالعه بنویسیم، نه فقط کتابی برای خواندن. در ابتدا سعی می‌کنیم آن را آسان نگه داریم و یک مسیر منطقی از صفر تا دکس را دنبال کنیم. با این حال، زمانی که مفاهیم یادگیری شروع به پیچیده‌تر شدن می‌کنند، از ساده بودن دست می‌کشیم و واقع‌بین می‌مانیم. دکس ساده است، اما آسان نیست. سال‌ها طول کشید تا ما به آن تسلط پیدا و تمام جزئیات موتور را درک کنیم. انتظار نداشته باشید که بتوانید در عرض چند روز و با مطالعه‌ای معمولی، همه این مطالب را یاد بگیرید. این کتاب در سطح بسیار بالایی توجه شما را می‌طلبد. در ازای آن، ما عمق بی سابقه‌ای از پوشش تمام جنبه‌های دکس را ارائه می‌دهیم که به شما این امکان را می‌دهد که به یک متخصص واقعی دکس تبدیل شوید.

---

[1] - *Best Practice*

[2] - *Business Intelligence*

فرضیات در مورد شما

ما انتظار داریم خواننده ما دانش اولیه Power BI و تجربه‌ای در تجزیه‌وتحلیل اعداد داشته باشد. اگر قبلاً با زبان دکس آشنا شده‌اید، این برای شما خوب است—قسمت اول را سریع‌تر می‌خوانید—البته که دانستن دکس ضروری نیست.

در سراسر کتاب ارجاعاتی به کدهای MDX و SQL وجود دارد؛ با این حال، شما واقعاً نیازی به دانستن این زبان‌ها ندارید زیرا آن‌ها فقط مقایسه بین روش‌های مختلف نوشتن عبارات را نشان می‌دهند. اگر آن خطوط کد را متوجه نشدید، خوب است؛ آن بدین معنی است که آن موضوع خاص برای شما مناسب نیست.

در پیشرفته‌ترین بخش‌های کتاب، ما در مورد موازی سازی، دسترسی به حافظه، استفاده از CPU، و دیگر موضوعات فوق‌العاده زمان‌بر که ممکن است همه با آن‌ها آشنا نباشند، بحث می‌کنیم. هر توسعه‌دهنده‌ای در آنجا احساس راحتی می‌کند، در حالی که کاربران Power BI و Excel ممکن است کمی ترسیده باشند. با این وجود، این اطلاعات برای بحث در مورد بهینه سازی دکس مورد نیاز است. در واقع، پیشرفته‌ترین بخش کتاب بیشتر به سمت توسعه‌دهندگان BI است تا کاربران Power BI و Excel. با این حال، ما فکر می‌کنیم که همه از خواندن آن سود می‌برند.

سازماندهی این کتاب

این کتاب به گونه‌ای طراحی شده است که از فصل‌های ابتدایی به فصل‌های پیچیده، به شیوه‌ای منطقی جریان یابد. هر فصل با این فرض نوشته شده است که محتوای قبلی کاملاً درک شده است؛ تقریباً هیچ تکراری از مفاهیمی که قبلاً توضیح داده شده وجود ندارد. به همین دلیل، اکیداً پیشنهاد می‌کنیم که کتاب را از ابتدا تا انتها بخوانید و از رفتن زودهنگام به سراغ فصل‌های پیشرفته‌تر خودداری کنید.

هنگامی که کتاب را برای اولین بار خواندید، آن تبدیل به یک مرجع مفید خواهد شد: برای مثال، اگر در مورد رفتار *ALLSELECTED* شک دارید، می‌توانید مستقیماً به آن بخش بروید و ذهن خود را در مورد آن روشن کنید. با این وجود، خواندن آن بخش بدون هضم محتوای قبلی ممکن است منجر به ناامیدی یا بدتر از آن، درک ناقص مفاهیم شود.

با این اوصاف، محتوای کتاب به طور خلاصه در اینجا ارائه شده است:

- فصل ۱ مقدمه مختصری در مورد دکس است که چند بخش آن به کاربرانی اختصاص داده شده است که از قبل دانش برخی از زبان‌های دیگر مانند SQL، Excel یا MDX را دارند. ما در اینجا هیچ مفهوم جدیدی را معرفی نمی‌کنیم؛ فقط چند نکته در مورد تفاوت‌های بین دکس و زبان‌های دیگر که ممکن است برای خواننده شناخته شده باشد ارائه می‌دهیم.
- فصل ۲ خود زبان دکس را معرفی می‌کند. ما مفاهیم اساسی مانند ستون‌های محاسبه‌شده، معیارها و توابع پوشش‌دهنده خطا را پوشش می‌دهیم؛ ما همچنین بسیاری از توابع اساسی زبان را فهرست می‌کنیم.
- فصل ۳ به توابع جدولی اصلی اختصاص دارد. بسیاری از توابع در دکس روی جداول کار می‌کنند و به عنوان نتیجه نیز جداول را برمی‌گردند. در این فصل به ابتدایی‌ترین توابع جدولی می‌پردازیم، در حالی که توابع جدولی پیشرفته را در فصل ۱۲ و ۱۳ پوشش می‌دهیم.
- فصل ۴ زمینه‌های ارزیابی را تشریح می‌کند. زمینه‌های ارزیابی پایه و اساس زبان دکس هستند، بنابراین این فصل، همراه با فصل بعد، احتمالاً مهم‌ترین فصول در کل کتاب به شمار می‌روند.
- فصل ۵ فقط دو تابع را پوشش می‌دهد: *CALCULATE* و *CALCULATETABLE*. این دو تابع مهم‌ترین توابع در دکس هستند و به شدت به یک درک خوب از زمینه‌های ارزیابی متکی می‌باشند.
- فصل ۶ متغیرها را توضیح می‌دهد. ما در بسیاری از مثال‌های کتاب از متغیرها استفاده می‌کنیم، اما فصل ۶ جایی است که ترکیب آن‌ها را معرفی می‌کنیم و نحوه استفاده از متغیرها را توضیح می‌دهیم. این فصل

زمانی که مثال‌های زیادی را با استفاده از متغیرها در فصل‌های بعدی مشاهده می‌کنید به عنوان یک مرجع مفید خواهد بود.

- فصل ۷ پیمایشگرها و *CALCULATE* را پوشش می‌دهد: ازدواجی که در بهشت انجام شده است. یادگیری نحوه استفاده از پیمایشگرها، همراه با قدرت انتقال زمینه‌ای، بیشتر قدرت دکس را آزاد می‌کند. در این فصل، چند مثال را نشان می‌دهیم که برای درک چگونگی استفاده از این ابزار مفید است.
- فصل ۸ محاسبات هوش زمانی را در سطحی بسیار عمیق شرح می‌دهد. ابتدای سال تا حال، ابتدای ماه تا حال، مقادیر سال قبل، دوره‌های هفته‌ای، و تقویم‌های سفارشی برخی از محاسباتی هستند که در این فصل به آن‌ها پرداخته شده است.
- فصل ۹ به آخرین ویژگی معرفی شده در دکس اختصاص دارد: گروه‌های محاسباتی. گروه‌های محاسباتی به عنوان یک ابزار مدل‌سازی بسیار قدرتمند هستند. این فصل نحوه ایجاد و استفاده از گروه‌های محاسباتی را شرح می‌دهد، مفاهیم اولیه را معرفی می‌کند و چند مثال را نشان می‌دهد.
- فصل ۱۰ کاربردهای پیشرفته‌تری از زمینه فیلتری، شجره داده‌ها، بررسی زمینه فیلتری و سایر ابزارهای مفید برای محاسبه فرمول‌های پیشرفته را پوشش می‌دهد.
- فصل ۱۱ به شما نشان می‌دهد که چگونه محاسبات را روی سلسله‌مراتب انجام دهید و چگونه ساختارهای والد/فرزند را با استفاده از دکس مدیریت کنید.
- فصل‌های ۱۲ و ۱۳ توابع جدولی پیشرفته‌ای را پوشش می‌دهند که هم برای خلق کوئری‌ها و/یا برای محاسبه محاسبات پیشرفته مفید هستند.
- فصل ۱۴ دانش شما را در مورد زمینه ارزیابی یک گام جلوتر می‌برد و توابع پیچیده‌ای مانند *ALLSELECTED* و *KEEPFILTERS* را با کمک نظریه جداول گسترش‌یافته مورد بحث قرار می‌دهد. این یک فصل پیشرفته است که بیشتر اسرار عبارات پیچیده دکس را آشکار می‌کند.
- فصل ۱۵ در مورد مدیریت روابط در دکس است. در واقع، به لطف دکس، هر نوع رابطه‌ای را می‌توان در یک مدل داده تنظیم کرد. این فصل شامل توصیف بسیاری از انواع روابط است که در مدل داده‌های تحلیلی رایج هستند.
- فصل ۱۶ شامل چند مثال از محاسبات پیچیده حل شده در دکس است. این فصل آخر در مورد این زبان است که برای کشف راه‌حل‌ها و ایده‌های جدید مفید می‌باشد.
- فصل ۱۷ شامل شرح مفصلی از موتور VertiPaq است که رایج‌ترین موتور ذخیره سازی مورد استفاده در مدل‌هایی است که دکس را اجرا می‌کنند. درک آن برای یادگیری نحوه به دست آوردن بهترین عملکرد در دکس ضروری است.
- فصل ۱۸ از دانش فصل ۱۷ برای نشان دادن بهینه‌سازی‌های احتمالی استفاده می‌کند که می‌توانید در سطح مدل داده اعمال کنید. شما یاد می‌گیرید که چگونه تعدادعناصر ستون‌ها را کاهش دهید، چگونه ستون‌ها را برای وارد کردن انتخاب کنید، و چگونه با انتخاب انواع روابط مناسب و با کاهش مصرف حافظه در دکس، عملکرد را بهبود بخشید.
- فصل ۱۹ به شما می‌آموزد که چگونه یک طرح کوئری را بخوانید و چگونه عملکرد یک کوئری دکس را با کمک ابزارهایی مانند DAX Studio و SQL Server Profiler اندازه‌گیری کنید.
- فصل ۲۰ چند تکنیک بهینه‌سازی را بر اساس محتوای فصل‌های قبلی در مورد بهینه‌سازی نشان می‌دهد. ما بسیاری از عبارات دکس را نشان می‌دهیم، عملکرد آن‌ها را اندازه‌گیری می‌کنیم و سپس فرمول‌های بهینه‌شده را نمایش و توضیح می‌دهیم.

کنوانسیون‌ها

در این کتاب از قراردادهای زیر استفاده شده است:

- از حالت **ضخیم** برای نشان دادن متنی که شما می‌نویسید استفاده می‌شود.
- از حالت **کج** برای نشان دادن عبارات جدید، نام معیارها، ستون‌های محاسبه‌شده، جداول و پایگاه‌داده استفاده می‌شود.
- حروف اول نام کادرهای گفت‌وگو، عناصر کادر گفت‌وگو و دستورات با حروف بزرگ نوشته می‌شوند. به عنوان مثال، کادر گفت‌وگوی *Save As*.
- نام سربرگ‌های ریبونی همگی با حرف بزرگ آورده شده‌اند.
- میانبرهای صفحه کلید با علامت مثبت (+) نشان داده می‌شوند که نام کلیدها را از هم جدا می‌کند. برای مثال، Ctrl+Alt+Delete بدین معناست که کلیدهای Ctrl، Alt و Delete را هم‌زمان فشار دهید.

درباره محتوای همراه

ما محتوای همراه را برای غنی سازی تجربه یادگیری شما گنجانده‌ایم. محتوای همراه این کتاب را می‌توانید از صفحه زیر دریافت کنید:

MicrosoftPressStore.com/DefinitiveGuideDAX/downloads

محتوای همراه شامل موارد زیر است:

- یک نسخه پشتیبان SQL Server از پایگاه‌داده Contoso Retail DW که می‌توانید برای ساختن نمونه‌های خودتان استفاده کنید. این یک پایگاه‌داده آزمایشی استاندارد ارائه شده توسط Microsoft است که ما آن را با چند نما توسعه داده و غنی کرده‌ایم تا ایجاد یک مدل داده بر مبنای آن آسان‌تر شود.
- یک مدل Power BI Desktop جداگانه برای هر شکل از کتاب. هر شکل فایل مخصوص به خود را دارد. مدل داده تقریباً همیشه یکسان است، اما می‌توانید از این فایل‌ها برای پیگیری دقیق مراحل ذکر شده در کتاب استفاده کنید.

فصل ۱ دکس چیست؟

دکس، که سرنام عبارت Data Analysis eXpressions می‌باشد، زبان برنامه‌نویسی Microsoft Power BI, Microsoft Analysis Services و Microsoft Power Pivot برای Excel می‌باشد. دکس در سال ۲۰۱۰ همراه با اولین عرضه PowerPivot برای Microsoft Excel 2010 ایجاد شد. در سال ۲۰۱۰، PowerPivot بدون فاصله تلفظ می شد. فاصله در نام Power Pivot در سال ۲۰۱۳ معرفی شد. از آن زمان تا کنون دکس هم در جامعه Excel، که از دکس برای ایجاد مدل داده power pivot در Excel استفاده می‌کنند و هم در جامعه هوش تجاری (BI)، که از دکس برای ایجاد مدل در Power BI و Analysis Services استفاده می‌کنند، محبوبیت پیدا کرد. دکس در تعداد زیادی ابزارهای متفاوت ارائه شده است، که همگی آن‌ها از موتور داخلی یکسانی استفاده می‌کنند که *Tabular* نامیده می‌شود. به همین دلیل، اغلب اوقات که ما به *مدل‌های Tabular* اشاره می‌کنیم، شامل تمام این ابزارهای متفاوت در تنها یک کلمه می‌شود.

دکس زبان ساده‌ای است. با این وجود، دکس از اغلب زبان‌های برنامه‌نویسی متفاوت می‌باشد، بنابراین آشنا شدن با برخی از مفاهیم جدید آن شاید کمی زمان‌بر باشد. طبق تجربه ما، بر اساس آموزش دکس به هزاران نفر، یادگیری مبانی دکس آسان می‌باشد: شما قادر خواهید بود در چند ساعت از آن استفاده کنید. وقتی زمان آن فرا می‌رسد که مفاهیم پیشرفته همانند زمینه‌های ارزیابی، پیمایش، و انتقال زمینه‌ای را بشناسید، هر چیزی پیچیده به نظر خواهد رسید. منصرف نشوید! صبور باشید. زمانی که مغز شما شروع به هضم این مفاهیم می‌کند، شما متوجه خواهید شد که دکس در حقیقت، زبان ساده‌ای است. فقط کمی زمان می‌برد تا به آن عادت کنید.

این فصل در ابتدا با این خلاصه شروع می‌شود که مدل داده بر حسب جداول و روابط چیست. ما به خوانندگان در هر سطحی از تجربه توصیه می‌کنیم این فصل را مطالعه کنند تا با اصطلاحات استفاده شده در سرتاسر این کتاب هنگام اشاره به جداول، مدل‌ها، و انواع دیگر ارتباطات آشنا شوند.

در بخش‌های بعدی، ما توصیه‌هایی را به خوانندگان ارائه می‌دهیم که تجربه کار با زبان‌های برنامه‌نویسی همانند Microsoft Excel، SQL و MDX را دارند. هر بخش بر یک زبان خاص تمرکز دارد، تا خوانندگانی که کنجکاو هستند دکس را به طور خلاصه با آن زبان مقایسه کنند. اگر مقایسه برای شما سودمند می‌باشد بر زبان‌هایی که آن‌ها را می شناسید تمرکز کنید؛ سپس بخش پایانی را مطالعه بفرمایید، «دکس برای کاربران Power BI» و بعد به سراغ فصل بعدی بروید جایی که به واقع سفر ما به درون زبان دکس از آنجا آغاز می‌شود.

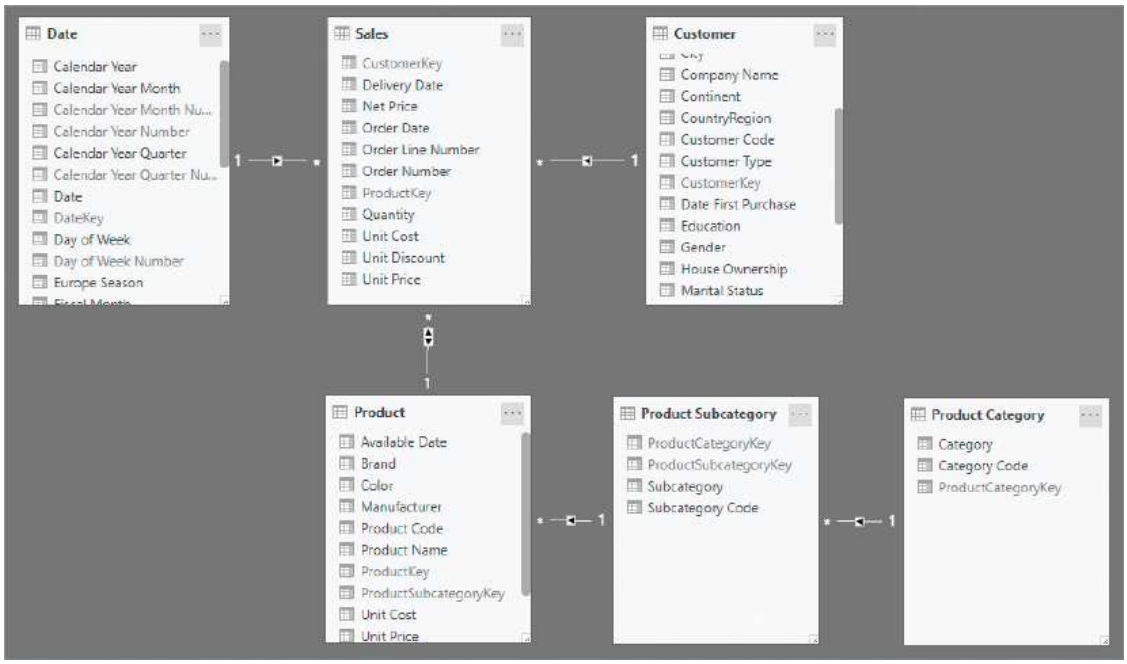
شناخت مدل داده

دکس زبانی است که به طور خاص برای محاسبه فرمول‌های کسب‌وکار بر روی یک مدل داده طراحی شده است. شاید خوانندگان از قبل بدانند که یک مدل داده چیست. با وجود این، ما با یک توضیح درباره مدل‌های داده و روابط شروع می‌کنیم تا پایه‌ای ایجاد کنیم که بر روی آن دانش دکس خوانندگان ساخته شود.

مدل داده یک مجموعه از جداول می‌باشد که به‌وسیله روابط به هم مرتبط شده‌اند.

ما همه می‌دانیم که جدول چیســـت: یک مجموعه از ردیف‌ها که دربردارنده داده می‌باشـــند، که در آن هر ردیف به ستون‌هایی تقسیم شده است. هر ستون یک نوع داده دارد و دربردارنده یک قسمت از اطلاعات می‌باشد. به طور معمول ما به یک ردیف در یک جدول به‌صـورت یک رکورد[۱] اشـاره می‌کنیم. جداول مسـیر مطمئنی برای سـازماندهی داده می‌باشـند. یک جدول خود یک مدل داده اسـت، اگرچه در سـاده‌ترین شـکل آن. بنابراین، زمانی که ما اسـامی و اعداد را در کارپوشـه Excel می‌نویسیم، در حال ساخت یک مدل داده می‌باشیم.

اگر یک مدل داده حاوی تعداد زیادی جدول باشـد، بسـیار محتمل اسـت که آن‌ها از طریق روابط به یکدیگر مرتبط شـوند. یک رابطه پیوندی بین دو جدول می‌باشـد. وقتی دو جدول با یک رابطه به یکدیگر گره خورده‌اند، می‌گوییم که آن‌ها مرتبط هستند. از نظر گرافیکی یک رابطه بوسیله یک خط که دو جدول را به هم وصل می‌کند مشخص می شود. شکل ۱-۱ یک نمونه از یک مدل داده را نشان می‌دهد.



شکل ۱-۱ این مدل داده از شش جدول ساخته شده است

در ادامه تعدادی از جنبه‌های مهم روابط عنوان می‌شود:

- دو جدول در یک رابطه نقش مشـابهی ندارند. آن‌ها *طرف‌تکی*[۲] رابطه و *طرف‌چندی*[۳] رابطه نامیده می‌شـوند، که به ترتیب با ۱ و علامت * نمایش داده می‌شـوند. در شـکل ۱-۱ بر رابطه بین *Product* و *Product Subcategory* تمرکز کنید. یک زیردسته[۴] منفرد حاوی محصولات زیادی می‌باشد، درحالی‌که یک محصول منفرد فقط یک زیردسـته دارد. بنابراین، *Product Subcategory* که سـمت تکی این رابطه می‌باشـد، یک زیردسته دارد، در حالی که *Product* که سمت چندی این رابطه می‌باشد محصولات زیادی دارد.
- انواع خاص روابط عبارتند از: رابطه ۱:۱ (یک‌به‌یک) و روابط ضعیف[۵]. در روابط ۱:۱ هر دو جدول در سمت تکی می‌باشند، درحالی‌که در روابط ضعیف، هر دو جدول می‌توانند در سمت چندی رابطه باشند. این انواع خاص روابط، رایج نمی‌باشند؛ ما آن‌ها را با جزئیات بیشتری در فصل ۱۵ «روابط پیشرفته» مورد بحث قرار

---

۱ - *Record*

۲ - *One-Side*

۳ - *Many-Side*

۴ - *Subcategory*

۵ - *Weak Relationships*

خواهیم داد.

- ستون‌های استفاده شده برای ایجاد رابطه، که به طور معمول نام مشابهی در هر دو جدول دارند، کلید رابطه نامیده می‌شوند. در سمت تکی رابطه این ستون نیازمند داشتن مقادیر منحصربه‌فرد برای هر ردیف می‌باشد و نمی‌تواند حاوی مقادیر خالی باشد. در سمت چندی رابطه یک مقدار مشابه می‌تواند در ردیف‌های متفاوت بسیاری تکرار شود، و بیشتر اوقات هم به همین شکل می‌باشد. هنگامی که یک ستون یک مقدار منحصربه‌فرد برای هر ردیف دارد، آن ستون یک کلید برای آن جدول نامیده می‌شود.
- روابط می‌توانند یک زنجیره تشکیل دهند. هر محصولی یک زیرد سته دارد و هر زیرد سته نیز دارای یک دسته[1] می‌باشد. بنابراین، هر محصولی یک دسته دارد. به منظور بازیابی دسته یک محصول، شخص به پیمودن زنجیره حاصل از دو رابطه نیاز خواهد داشت. شکل ۱-۱ شامل مثالی از یک زنجیره ایجاد شده از سه ارتباط می‌باشد، که از *Sales* شروع می‌شود و تا *Product Category* ادامه می‌یابد.
- در هر رابطه، یک یا دو پیکان کوچک می‌تواند *جهت فیلتر متقاطع*[2] را مشخص کند. شکل ۱-۱ دو پیکان را در رابطه بین *Sales* و *Product* نشان می‌دهد، در حالی که تمام دیگر روابط تنها یک پیکان دارند این پیکان نشان‌دهنده جهت فیلتر خودکار رابطه (*فیلتر متقاطع*) می‌باشند. از آنجایی که تعیین جهت صحیح فیلترها یکی از مهم‌ترین مهارت‌هایی است که می‌بایست یاد بگیرید، ما در مورد این موضوع با جزئیات بیشتری در فصول بعدی بحث خواهیم کرد. به طور معمول ما افراد را تشویق می‌کنیم از فیلترهای دوطرفه استفاده نکنند، همان‌طور که در فصل ۱۵ توضیح داده شده است. آن‌ها تنها در راستای اهداف آموزشی در این مدل ارائه شده‌اند.

شناخت جهت یک رابطه

هر رابطه‌ای می‌تواند یک فیلتر متقاطع یک‌طرفه یا دوطرفه داشته باشد. عمل فیلتر همیشه از سمت تکی رابطه به طرف‌چندی رابطه می‌باشد. اگر فیلتر متقاطع دوطرفه باشد—یعنی اگر دو پیکان روی آن باشد—در نتیجه عمل فیلتر کردن می‌تواند از سمت چندی‌رابطه به سمت تکی نیز رخ دهد.

یک مثال شاید به درک این رفتار کمک کند. اگر یک گزارش بر اساس مدل داده نشان داده شده در شکل ۱-۱ باشد، به این شکل که *Calendar Years* در منطقه ردیف‌ها و *Quantity* و *Count of Product Name* در منطقه مقادیر باشند، نتیجه‌ای را ایجاد می‌کند که در شکل ۱-۲ نشان داده شده است.

Calendar Year	Quantity	Count of Product Name
CY 2007	44,310	1258
CY 2008	40,226	1478
CY 2009	55,644	1513
Total	**140,180**	**2517**

شکل ۱-۲ این گزارش تاثیر فیلتر متقاطع در مورد جداول متعدد را نشان می‌دهد.

Calendar Year ستونی است که متعلق به جدول *Date* می‌باشد. از آنجا که *Date* در سمت تکی رابطه با *Sales* می‌باشد، موتور جدول Sales را بر حسب سال فیلتر می‌کند. این دلیلی است که مقدار نمایش داده شده به وسیله سال فیلتر شده است.

---

[1] - *Category*

[2] - *Cross Filter Direction*

در خصوص *products*، این سناریو تا حدودی متفاوت می‌باشد. عمل فیلتر به این دلیل اتفاق می‌افتد که رابطه بین جداول *Sales* و *Product* دوطرفه می‌باشد. زمانی که ما *Count of Product Name* را در گزارش قرار می‌دهیم، تعداد *products* فروش رفته در هر سال را به د ست می‌آوریم چرا که فیلتر بر روی سال از طریق Sales به *product* تسری می‌یابد. همان‌طور که در بخش‌های بعدی توضیح می‌دهیم، اگر ارتباط بین *Sales* و *product* یک‌طرفه باشد، نتیجه متفاوت می‌باشد.

اگر ما این گزارش را از طریق قرار دادن *Color* در منطقه ردیف‌ها و اضافه کردن *Count of Date* در منطقه مقادیر تغییر دهیم، نتیجه متفاوت خواهد شد، همان‌طورکه در شکل ۱-۳ نشان داده شده است.

Color	Quantity	Count of Product Name	Count of Date
Azure	546	14	2556
Black	33,618	602	2556
Blue	8,859	200	2556
Brown	2,570	77	2556
Gold	1,393	50	2556
Green	3,020	74	2556
Grey	11,900	283	2556
Orange	2,203	55	2556
Pink	4,921	84	2556
Purple	102	6	2556
Red	8,079	99	2556
Silver	27,551	417	2556
Silver Grey	959	14	2556
Transparent	1,251	1	2556
White	30,543	505	2556
Yellow	2,665	36	2556
Total	**140,180**	**2517**	**2556**

شکل ۱-۳ این گزارش نشان می‌دهد که اگر فیلتر دوطرفه فعال نباشد، جداول فیلتر نمی‌شود.

فیلتری که بر روی ردیف‌ها می‌باشد، ستون *Color* در جدول *Product* است. از آنجایی که *Product* در سمت تکی رابطه با Sales می‌باشد، *Quantity* به درستی فیلتر شده است. *Count of Product Name* فیلتر شده است چرا که آن مقادیر را از جدولی محاسبه می‌کند که در ردیف‌ها قرار دارد، یعنی *Product*. عدد غیر منتظره *Count of Date* می‌باشد. در حقیقت، آن همیشه یک عدد را برای تمام ردیف‌ها نشان می‌دهد—یعنی، تعداد کل ردیف‌ها در جدول *Date*.

فیلتر ناشی از ستون *Color* به *Date* تسری نمی‌یابد چرا که ارتباط بین *Date* و *Sales* یک‌طرفه می‌باشد. بنابراین، اگرچه، *Sales* فیلتر فعالی بر روی آن دارد، این فیلتر نمی‌تواند به *Date* تسری یابد چرا که نوع این رابطه مانع آن می‌شود.

اگر ما با فعال کردن فیلتر متقاطع دوطرفه ارتباط بین *Date* و *Sales* را تغییر دهیم، نتیجه چیزی است که در شکل ۱-۴ نشان داده شده است.

این اعداد هم‌اکنون تعداد روزهایی را نشان می‌دهند که حداقل یک محصول مربوط به رنگ مورد نظر فروش رفته باشد. در اولین نگاه، شاید به نظر برسد که می‌بایست تمام ارتباطات به شکل دوطرفه تعریف شوند، برای اینکه اجازه داده شود فیلتر در هر جهتی تسری یابد و همیشه نتایجی بازگرداند که با معنی می‌باشند. همان‌طور که در ادامه این کتاب یاد خواهید گرفت، طراحی یک مدل داده به این شیوه تقریباً هرگز مناسب نمی‌باشد. در حقیقت، بسته به سناریویی که شما با آن کار می‌کنید، تسری صحیح روابط را انتخاب خواهید کرد. اگر شما از پیشنهادات ما پیروی کنید، تا آنجایی که می‌توانید از فیلتر دوطرفه خودداری خواهید کرد.

Color	Quantity	Count of Product Name	Count of Date
Azure	546	14	41
Black	33,618	602	811
Blue	8,859	200	408
Brown	2,570	77	169
Gold	1,393	50	106
Green	3,020	74	188
Grey	11,900	283	499
Orange	2,203	55	142
Pink	4,921	84	226
Purple	102	6	11
Red	8,079	99	286
Silver	27,551	417	722
Silver Grey	959	14	63
Transparent	1,251	1	14
White	30,543	505	750
Yellow	2,665	36	110
Total	**140,180**	**2517**	**2556**

شکل ۱-۴ اگر ما فیلتر دوطرفه را فعال کنیم، جدول *Date* با استفاده از ستون *Color* فیلتر می‌شود

دکس برای کاربران Excel

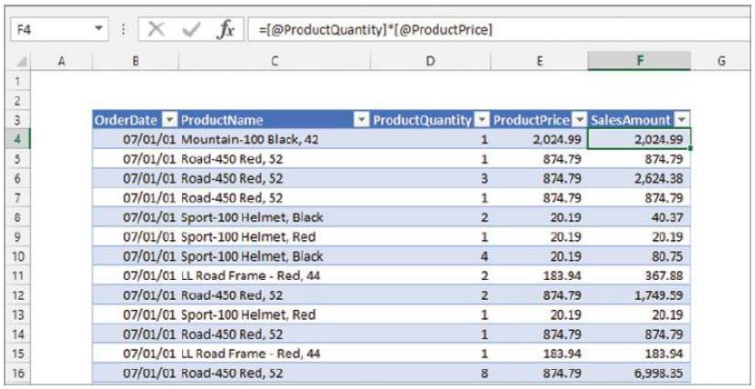
به احتمال زیاد شما زبان فرمول Excel را می‌شناسید که دکس تا حدودی شبیه آن می‌باشد. از این گذشته، ریشه‌های دکس در Power Pivot for Excel قرار دارد و تیم توسعه سعی داشت این دو زبان را شبیه به هم نگه دارد. این شباهت انتقال به این زبان جدید را آسان‌تر می‌کند. با این حال، چند تفاوت مهم وجود دارد.

سلول‌ها در مقابل جداول

Excel محاسبات را روی سلول‌ها انجام می‌دهد. یک سلول با استفاده از مختصات آن ارجاع داده می شود. بنابراین، می‌توانیم فرمول‌ها را به‌صورت زیر بنویسیم:

```
= (A1 * 1.25) - B2
```

در دکس مفهوم سلول و مختصات آن وجود ندارد. دکس روی جداول و ستون‌ها کار می‌کند نه سلول‌ها. در نتیجه، عبارات دکس به جداول و ستون‌ها اشاره دارد و این به معنای نوشتن کد به شکل متفاوتی است. مفاهیم جداول و ستون‌ها در Excel جدید نیستند. در واقع، اگر با استفاده از تابع *Format as Table* یک محدوده Excel را به عنوان جدول تعریف کنیم، می‌توانیم فرمول‌هایی را در Excel بنویسیم که به جداول و ستون‌ها اشاره می‌کنند. در شکل ۱-۵، ستون *SalesAmount* عبارتی را ارزیابی می‌کند که به ستون‌های موجود در همان جدول به جای سلول‌های کارپوشه[۱] اشاره می‌کند.



OrderDate	ProductName	ProductQuantity	ProductPrice	SalesAmount
07/01/01	Mountain-100 Black, 42	1	2,024.99	2,024.99
07/01/01	Road-450 Red, 52	1	874.79	874.79
07/01/01	Road-450 Red, 52	3	874.79	2,624.38
07/01/01	Road-450 Red, 52	1	874.79	874.79
07/01/01	Sport-100 Helmet, Black	2	20.19	40.37
07/01/01	Sport-100 Helmet, Red	1	20.19	20.19
07/01/01	Sport-100 Helmet, Black	4	20.19	80.75
07/01/01	LL Road Frame - Red, 44	2	183.94	367.88
07/01/01	Road-450 Red, 52	2	874.79	1,749.59
07/01/01	Sport-100 Helmet, Red	1	20.19	20.19
07/01/01	Road-450 Red, 52	1	874.79	874.79
07/01/01	LL Road Frame - Red, 44	1	183.94	183.94
07/01/01	Road-450 Red, 52	8	874.79	6,998.35

شکل ۱-۵ *Excel* می‌تواند به نام ستون‌ها در جداول ارجاع دهد.

[۱] - *Workbook*

با استفاده از Excel، به ستون‌های موجود در جدول با غالب [*@ColumnName*] اشاره می‌کنیم. *ColumnName* نام ستون مورد استفاده است و نماد @ به معنای «مقدار را در مورد ردیف فعلی در نظر بگیرید». اگرچه این ترکیب شهودی نیست، معمولاً ما این عبارات را نمی‌نویسیم. وقتی روی یک سلول کلیک می‌کنیم ظاهر می‌شوند و Excel وظیفه دارد کد مناسب را برای ما درج کند.

ممکن است تصور کنید Excel دارای دو روش مختلف برای انجام محاسبات است. می‌توانیم از ارجاعات استاندارد سلولی ا ستفاده کنیم، در این صورت فرمول سلول *F4* ممکن ا ست *E4*D4* با شد یا می‌توانیم از ارجاعات ستونی در داخل جدول ا ستفاده کنیم. ا ستفاده از ارجاعات ستونی این مزیت را دارد که می‌توانیم از یک عبارت در تمام سلول‌های یک ستون استفاده کنیم و Excel فرمول را با مقدار متفاوتی برای هر ردیف محاسبه می‌کند.

برخلاف Excel، دکس فقط روی جداول کار می‌کند. همه فرمول‌ها باید به ستون‌های داخل جداول اشاره کنند. به عنوان مثال، در دکس ضرب قبلی را به این صورت می‌نویسیم:

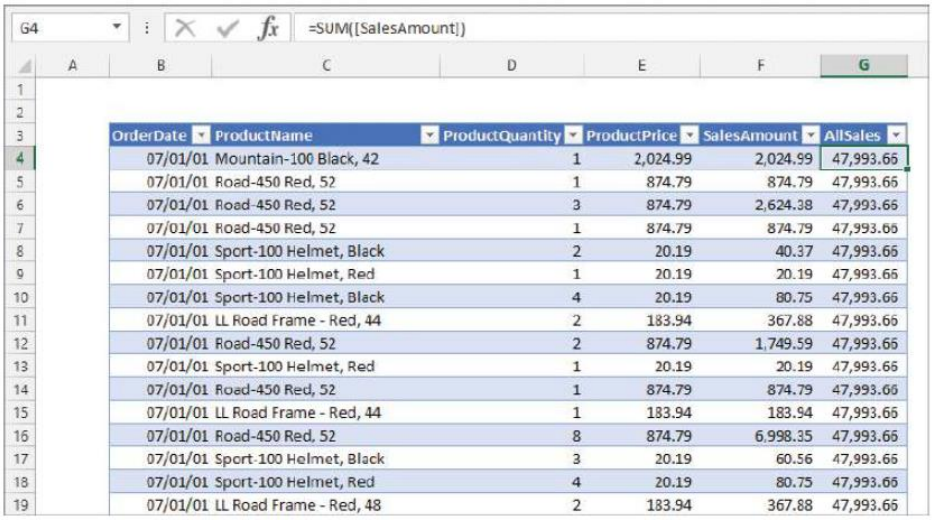
```
Sales[SalesAmount] = Sales[ProductPrice] * Sales[ProductQuantity]
```

همان‌طور که می‌بینید، هر ستون با نام جدول خود پیشونددار شده است. در Excel، ما نام جدول را ارائه نمی‌دهیم زیرا فرمول‌های Excel درون یک جدول کار می‌کنند. با این حال، دکس بر روی یک مدل داده کار می‌کند که شامل جداول زیادی است. در نتیجه، ما باید نام جدول را مشخص کنیم زیرا دو ستون در جداول مختلف ممکن است نام یکسانی داشته باشند.

بسیاری از توابع در دکس مانند توابع معادل در Excel کار می‌کنند. به عنوان مثال، تابع *IF* در دکس و Excel به همین صورت خوانده می‌شود:

```
Excel IF ( [@SalesAmount] > 10, 1, 0)
DAX IF ( Sales[SalesAmount] > 10, 1, 0)
```

یکی از جنبه‌های مهمی که در آن ترکیب Excel و دکس متفاوت ا ست، نحوه ارجاع به کل یک ستون ا ست. در واقع، در [*@ProductQuantity*]، علامت @ به معنای «مقدار در ردیف فعلی» ا ست. در دکس، نیازی به تعیین این مو ضوع نی ست که یک مقدار باید از ردیف فعلی باشد، زیرا این رفتار پیش‌فرض این زبان است. در Excel، می‌توانیم با حذف نماد @ به کل ستون ارجاع دهیم، یعنی تمام ردیف‌های آن ستون. این وضعیت را می‌توانید در شکل ۱-۶ مشاهده کنید.



	A	B	C	D	E	F	G
1							
2							
3		OrderDate	ProductName	ProductQuantity	ProductPrice	SalesAmount	AllSales
4		07/01/01	Mountain-100 Black, 42	1	2,024.99	2,024.99	47,993.66
5		07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.66
6		07/01/01	Road-450 Red, 52	3	874.79	2,624.38	47,993.66
7		07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.66
8		07/01/01	Sport-100 Helmet, Black	2	20.19	40.37	47,993.66
9		07/01/01	Sport-100 Helmet, Red	1	20.19	20.19	47,993.66
10		07/01/01	Sport-100 Helmet, Black	4	20.19	80.75	47,993.66
11		07/01/01	LL Road Frame - Red, 44	2	183.94	367.88	47,993.66
12		07/01/01	Road-450 Red, 52	2	874.79	1,749.59	47,993.66
13		07/01/01	Sport-100 Helmet, Red	1	20.19	20.19	47,993.66
14		07/01/01	Road-450 Red, 52	1	874.79	874.79	47,993.66
15		07/01/01	LL Road Frame - Red, 44	1	183.94	183.94	47,993.66
16		07/01/01	Road-450 Red, 52	8	874.79	6,998.35	47,993.66
17		07/01/01	Sport-100 Helmet, Black	3	20.19	60.56	47,993.66
18		07/01/01	Sport-100 Helmet, Red	4	20.19	80.75	47,993.66
19		07/01/01	LL Road Frame - Red, 48	2	183.94	367.88	47,993.66

شکل ۱-۶ در *Excel*، می‌توانید با حذف علامت @ قبل از نام ستون، به کل ستون ارجاع دهید.

مقدار ستون *AllSales* در تمام ردیف‌ها یکسان است زیرا مجموع کل ستون *SalesAmount* است. به عبارت دیگر،

تفاوت ترکیبی[۱] بین مقدار یک ستون در ردیف فعلی و مقدار کل ستون وجود دارد.

این در دکس متفاوت است. در دکس این حالتی است که شما عبارت *AllSales* شکل ۱-۶ را می‌نویسید:

```
AllSales := SUM ( Sales[SalesAmount] )
```

هیچ تفاوت ترکیبی بین بازیابی مقدار یک ستون برای یک ردیف خاص و استفاده از ستون به عنوان یک کل وجود ندارد. دکس می‌داند که ما می‌خواهیم تمام مقادیر ستون را جمع کنیم، زیرا از نام ستون در یک تجمیع‌گر (در این مورد تابع *SUM*) استفاده می‌کنیم، که نیاز به یک نام ستون به عنوان پارامتر دارد. بنابراین، اگرچه *Excel* برای تمایز بین دو نوع داده برای بازیابی به یک ترکیب صریح نیاز دارد، دکس ابهام‌زدایی را به‌طور خودکار انجام می‌دهد. این تمایز ممکن است حداقل در ابتدا گیج‌کننده باشد.

Excel و دکس: دو زبان تابعی

یکی از جنبه‌های مشابه این دو زبان این است که هر دو زبان Excel و دکس زبان‌های تابعی[۲] هستند. یک زبان تابعی از عباراتی تشکیل شده است که اساساً فراخوان‌های تابع می‌باشند. در Excel و دکس، مفاهیم مربوط به دستورات[۳]، حلقه‌ها[۴] و پرش‌ها[۵] وجود ندارند، اگرچه در بسیاری از زبان‌های برنامه‌نویسی متداول هستند. در دکس، همه چیز یک عبارت[۶] است. این جنبه از این زبان اغلب برای برنامه‌نویسان از زبان‌های مختلف دیگر یک چالش به حساب می‌آید، اما برای کاربران Excel اصلاً جای تعجب ندارد.

پیمایشگرها در دکس

مفهومی که ممکن است برای شما جدید باشد، مفهوم پیمایشگرها[۷] است. هنگام کار در Excel، محاسبات را یک مرحله و در یک زمان انجام می‌دهید. مثال قبل نشان داد که برای محاسبه کل فروش، یک ستون حاوی قیمت ضرب در مقدار ایجاد می‌کنیم. سپس به عنوان مرحله دوم، آن را جمع می‌کنیم تا کل فروش را محاسبه کنیم. این عدد سپس برای مثال به عنوان مخرج برای محاسبه درصد فروش هر محصول مفید می‌باشد.

با استفاده از دکس می‌توانید همان عملیات را در یک مرحله با استفاده از پیمایشگرها انجام دهید. یک پیمایشگر دقیقاً همان کاری را انجام می‌دهد که نامش نشان می‌دهد: روی یک جدول پیمایش می‌کند و یک محاسبه را روی هر ردیف از جدول انجام می‌دهد و نتیجه را جمع می‌کند تا مقدار منحصربه‌فرد درخواستی را ایجاد کند.

با استفاده از مثال قبلی، اکنون می‌توانیم مجموع همه فروش‌ها را با استفاده از پیمایش‌گر *SUMX* محاسبه کنیم:

```
AllSales :=
SUMX (
    Sales,
    Sales[ProductQuantity] * Sales[ProductPrice]
)
```

---

۱ – *Syntactical*

۲ - *Functional*

۳ - *Statement*

۴ - *Loop*

۵ - *Jump*

۶ – *Expression*

۷ - *Iterators*

این رویکرد هم یک نقطه قوت و هم یک نقطه ضعف را آشکار می‌کند. نقطه قوت این است که ما می‌توانیم بسیاری از محاسبات پیچیده را در یک مرحله انجام دهیم بدون اینکه نگران اضافه کردن ستون‌هایی باشیم که در نهایت فقط برای فرمول‌های خاصی مفید هستند. نقطه ضعف این است که برنامه‌نویسی با دکس نسبت به برنامه‌نویسی با Excel کمتر بصری است. در واقع، شما ستونی را که قیمت را در مقدار ضرب می‌کند، نمی‌بینید؛ آن فقط در طول زمان محاسبه وجود دارد.

همان‌طور که در ادامه توضیح خواهیم داد، می‌توانیم یک ستون محاسبه شده ایجاد کنیم که ضرب قیمت در مقدار را محاسبه می‌کند. با این وجود، انجام این کار به ندرت عمل خوبی است زیرا از حافظه استفاده می‌کند و ممکن است منجر به کندی محاسبات شود، مگر اینکه از DirectQuery و Aggregations استفاده کنید، همان‌طور که در فصل ۱۸، «بهینه سازی VertiPaq» توضیح دادیم.

دکس به نظریه نیاز دارد

اجازه بدهید این واقعیت را واضح بگوییم: دکس نیاز دارد فرد در ابتدا نظریه را مطالعه کند، تفاوتی بین زبان‌های برنامه‌نویسی وجود ندارد. این یک تفاوت در طرز فکر است. احتمالاً به جست‌وجو در وب برای فرمول‌های پیچیده و الگوهای راه‌حل برای سناریوهایی که سعی در حل آن‌ها دارید عادت کرده‌اید. هنگامی که از Excel استفاده می‌کنید، به احتمال زیاد فرمولی پیدا خواهید کرد که تقریباً آنچه را که نیاز دارید انجام می‌دهد. می‌توانید فرمول را کپی کنید، آن را مطابق با نیازهای خود سفارشی کنید، و سپس بدون نگرانی در مورد نحوه عملکرد از آن استفاده کنید.

با این حال، این رویکرد که در Excel کار می‌کند، در مورد دکس کارگر نیست. قبل از اینکه بتوانید کد دکس خوب بنویسید، باید نظریه دکس را مطالعه کنید و به طور کامل درک کنید که زمینه‌های ارزیابی چگونه کار می‌کنند. اگر پایه نظری مناسبی نداشته باشید، متوجه خواهید شد که دکس یا مقادیری را مانند جادو محاسبه می‌کند یا اعداد عجیب‌وغریب را محاسبه می‌کند که معنایی ندارند. مشکل دکس نیست، بلکه این واقعیت است که شما هنوز دقیقاً متوجه نشده‌اید که دکس چگونه کار می‌کند.

خوشبختانه، نظریه پشت دکس به چند مفهوم مهم محدود می‌شود که در فصل ۴، «درک زمینه‌های ارزیابی» توضیح می‌دهیم. وقتی به آن فصل رسیدید، برای یادگیری شدید و مهیج آماده باشید. پس از تسلط بر آن محتوا، دکس هیچ رازی برای شما نخواهد داشت و یادگیری دکس عمدتاً موضوعی در خصوص کسب تجربه است. به یاد داشته باشید: دانستن نیمی از نبرد است. بنابراین تا زمانی که تا حدودی در مورد زمینه‌های ارزیابی مهارت کسب نکرده‌اید، سعی نکنید جلوتر بروید.

دکس برای توسعه دهندگان SQL

اگر به زبان SQL عادت دارید، قبلاً با جداول زیادی کار و برای تنظیم روابط بین ستون‌ها اتصال ایجاد کرده‌اید. از این منظر، احتمالاً در دنیای دکس احساس راحتی خواهید کرد. در واقع، محاسبات در دکس عبارت است از کوئری زدن از مجموعه‌ای از جداول که توسط روابط و مقادیر تجمیعی به یکدیگر متصل شده‌اند.

مدیریت رابطه

اولین تفاوت بین *SQL* و دکس در نحوه عملکرد روابط در مدل است. در SQL، ما می‌توانیم کلیدهای خارجی را بین جداول تنظیم کنیم تا روابط را اعلام کنند، اما این موتور هرگز از این کلیدهای خارجی در کوئری‌ها استفاده نمی‌کند، مگر اینکه در مورد آن‌ها صریح باشیم. به عنوان مثال، اگر یک جدول *Customers* و یک جدول *Sales* داشته باشیم، که در آن *CustomerKey* یک کلید اولیه در *Customers* و یک کلید خارجی در *Sales* است، می‌توانیم کوئری زیر را بنویسیم:

```
SELECT
   Customers.CustomerName,
   SUM ( Sales.SalesAmount ) AS SumOfSales
FROM
   Sales
   INNER JOIN Customers
    ON Sales.CustomerKey = Customers.CustomerKey
GROUP BY
   Customers.CustomerName
```

اگرچه ما رابطه را در مدل با استفاده از کلیدهای خارجی اعلام می‌کنیم، هنوز باید صریح باشیم و شرط اتصال را در کوئری بیان کنیم. اگرچه این رویکرد کوئری‌ها را طولانی‌تر می‌کند، اما مفید است زیرا می‌توانید از شرایط اتصال مختلف در کوئری‌های مختلف استفاده کنید و آزادی زیادی در نحوه بیان کوئری‌ها به شما می‌دهد.

در دکس، روابط بخشی از مدل هستند و همه آن‌ها *LEFT OUTER JOIN* هستند. هنگامی که آن‌ها در مدل تعریف می‌شوند، دیگر نیازی به تعیین نوع اتصال در کوئری ندارید: هر زمان که از ستون‌های مربوط به جدول اصلی استفاده می‌کنید، دکس از یک *LEFT OUTER JOIN* خودکار در کوئری استفاده می‌کند. بنابراین، در دکس شما می‌توانید کوئری قبلی SQL را به‌صورت زیر بنویسید:

```
EVALUATE
SUMMARIZECOLUMNS (
    Customers[CustomerName],
    "SumOfSales", SUM ( Sales[SalesAmount] )
)
```

از آنجایی که دکس رابطه موجود بین *Sales* و *Customers* را می‌داند، اتصال را به‌طور خودکار به دنبال مدل انجام می‌دهد. در نهایت، تابع *SUMMARIZECOLUMNS* باید یک گروه‌بندی را بر اساس *Customers[CustomerName]* انجام دهد، اما ما یک کلمه کلیدی برای آن نداریم: *SUMMARIZECOLUMNS* به طور خودکار داده‌ها را بر اساس ستون‌های انتخاب‌شده گروه‌بندی می‌کند.

دکس یک زبان تابعی است

SQL یک زبان اعلامی[۱] است. شما با اعلام مجموعه‌داده‌هایی که می‌خواهید بازیابی کنید با استفاده از دستورات *SELECT*، آنچه را که نیاز دارید، بدون نگرانی در مورد اینکه موتور واقعاً اطلاعات را چگونه بازیابی می‌کند، تعریف می‌کنید.

از طرف دیگر دکس یک زبان تابعی است. در دکس، هر عبارتی یک فراخوان تابع است. پارامترهای تابع به نوبه خود می‌توانند فراخوان‌های تابع دیگری باشند. ارزیابی پارامترها ممکن است به طرح‌های کوئری پیچیده‌ای منجر شود که دکس برای محاسبه نتیجه اجرا می‌کند.

به عنوان مثال، اگر بخواهیم فقط مشتریانی را که در اروپا زندگی می‌کنند بازیابی کنیم، می‌توانیم این کوئری را در SQL بنویسیم:

[۱] - *Declarative*

```
SELECT
   Customers.CustomerName,
   SUM ( Sales.SalesAmount ) AS SumOfSales
FROM
   Sales
   INNER JOIN Customers
    ON Sales.CustomerKey = Customers.CustomerKey
WHERE
   Customers.Continent = 'Europe'
GROUP BY
   Customers.CustomerName
```

با اســتفاده از دکس، شــرط *WHERE* را در کوئری اعلام نمی‌کنیم. در عوض، از یک تابع خاص (*FILTER*) برای فیلتر کردن نتیجه استفاده می‌کنیم:

```
EVALUATE
SUMMARIZECOLUMNS (
    Customers[CustomerName],
    FILTER (
        Customers,
        Customers[Continent] = "Europe"
    ),
    "SumOfSales", SUM ( Sales[SalesAmount] )
)
```

می‌توانید ببینید که *FILTER* یک تابع اســت: فقط مشــتریان ســاکن اروپا را برمی‌گرداند و نتیجه مورد انتظار را ایجاد می‌کند. ترتیبی که در آن ما این توابع را تودرتو می‌کنیم و انواع توابعی که اســتفاده می‌کنیم تأثیر زیادی بر روی نتیجه و عملکرد این موتور دارد. این در SQL نیز اتفاق می‌افتد، اگرچه در SQL ما به بهینه‌ســاز کوئری برای یافتن طرح کوئری بهینه اعتماد داریم. در دکس، اگرچه بهینه‌ســاز کوئری کار بســیار خوبی انجام می‌دهد، اما شــما به عنوان برنامه‌نویس مســئولیت بیشتری در نوشتن کد خوب دارید.

دکس به عنوان یک زبان برنامه‌نویسی و کوئری‌نویسی

در SQL، تمایز واضــحی بین زبان کوئری و زبان برنامه‌نویســی وجود دارد—یعنی مجموعه‌ای از دســتورالعمل‌ها برای ایجاد رویه‌های ذخیره‌شده، نماها و سایر قطعات کد در پایگاه‌داده استفاده می‌شود. هر گویش[۱] SQL عباراتی دارد که به برنامه‌نویسان اجازه می‌دهد مدل داده را با کد غنی کنند. با این حال، دکس عملاً هیچ تمایزی بین کوئری‌نویســی و برنامه‌نویســی قائل نمی‌شود. مجموعه‌ای غنی از توابع، جداول را دست‌کاری می‌کنند و به نوبه خود می‌توانند جداول را برگرداند. تابع *FILTER* در کوئری قبلی مثال خوبی برای این موضوع است.

از این منظر، به نظر می‌رسد دکس ساده‌تر از SQL است. هنگامی که آن را به عنوان یک زبان برنامه‌نویسی—کاربرد اصلی آن—یاد می‌گیرید، همه چیزهایی را که برای استفاده از آن به عنوان زبان کوئری نیز لازم است، خواهید دانست.

زیرکوئری‌ها و شروط در دکس و SQL

یکی از قدرتمندترین ویژگی‌های SQL به عنوان یک زبان کوئری، امکان اســتفاده از زیرکوئری‌ها اســت. دکس دارای مفاهیم مشابهی است. اما در مورد زیرکوئری‌های دکس، آن‌ها به هر حال از ماهیت تابعی این زبان سرچشمه می‌گیرند.

---

[۱] – *Dialect*

به عنوان مثال، برای بازیابی مشـــتریان و کل فروش به طور خاص برای مشـــتریانی که بیش از ۱۰۰ دلار آمریکا خرید کرده‌اند، می‌توانیم این کوئری را در SQL بنویسیم:

```
SELECT
   CustomerName,
   SumOfSales
FROM (
   SELECT
    Customers.CustomerName,
    SUM ( Sales.SalesAmount ) AS SumOfSales
   FROM
    Sales
    INNER JOIN Customers
     ON Sales.CustomerKey = Customers.CustomerKey
   GROUP BY
    Customers.CustomerName
   ) AS SubQuery
WHERE
   SubQuery.SumOfSales > 100
```

می‌توانیم با تودرتو کردن فراخوانی‌های تابع، همان نتیجه را در دکس به دست آوریم:

```
EVALUATE
FILTER (
    SUMMARIZECOLUMNS (
        Customers[CustomerName],
        "SumOfSales", SUM ( Sales[SalesAmount] )
    ),
    [SumOfSales] > 100
)
```

در این کد، زیرکوئری که *CustomerName* و *SumOfSales* را بازیابی می‌کند، بعداً به یک تابع *FILTER* وارد می‌شـود که فقط ردیف‌هایی را که *SumOfSales* بیشـتر از ۱۰۰ اسـت، حفظ می‌کند. در حال حاضـر، ممکن اسـت این کد برای شـما ناخوانا به نظر برسد. با این حال، به محض شروع یادگیری دکس، متوجه خواهید شد که استفاده از زیرکوئری‌ها بسیار ساده‌تر از SQL است و به طور طبیعی جریان دارد زیرا دکس یک زبان تابعی است.

دکس برای توسعه‌دهندگان MDX

بسیاری از متخصصان هوش تجاری شروع به یادگیری دکس می‌کنند زیرا این زبان جدید Tabular است. در گذشته، آن‌ها از زبان MDX برای سـاخت و کوئری مدل‌های چندبعدی خدمات تحلیلی[۱] اسـتفاده می‌کردند. اگر در میان آن‌ها هسـتید، برای یادگیری یک زبان کاملاً جدید آماده باشید: دکس و MDX اشتراک زیادی ندارند. بدتر از آن، برخی از مفاهیم در دکس شما را به یاد مفاهیم مشابه موجود در MDX می‌اندازد، در حالی که آن‌ها متفاوت می‌باشند.

در تجربه خود دریافتیم که یادگیری دکس پس از یادگیری MDX چالش برانگیزترین گزینه اسـت. برای یادگیری دکس، باید ذهن خود را از MDX آزاد کنید. سـعی کنید هر آنچه در مورد فضـاهای چند بعدی می‌دانید را فراموش کنید و با ذهنی تمیز آماده یادگیری این زبان جدید باشید.

[۱] - *Analysis Services Multidimensional*

چندبعدی در مقابل Tabular

MDX در فضای چندبعدی تعریف‌شده توسط یک مدل کار می‌کند. شکل فضای چندبعدی بر اساس معماری ابعاد و سلسله‌مراتب تعریف شده در مدل می‌باشد و این به نوبه خود مجموعه مختصات فضای چندبعدی را مشخص می‌کند. تقاطع مجموعه اعضا در بعدهای مختلف نقاطی را در فضای چندبعدی مشخص می‌کند. ممکن است مدتی طول کشیده تا متوجه شوید که [همه] اعضای هر سلسله‌مراتب صفتی در واقع نقطه‌ای در فضای چندبعدی است.

دکس به روشی بسیار ساده‌تر کار می‌کند. در اینجا هیچ بعد، عضو و نقطه‌ای در فضای چند بعدی وجود ندارد. به عبارت دیگر اصلاً فضای چند بعدی وجود ندارد. سلسله‌مراتبی وجود دارد که می‌توانیم آن‌ها را در مدل تعریف کنیم، اما آن‌ها با سلسله‌مراتب در MDX متفاوت می‌باشند. فضای دکس بر روی جداول، ستون‌ها و روابط ساخته شده است. هر جدول در یک مدل Tabular نه یک گروه معیاری است و نه یک بعد: فقط یک جدول است و برای محاسبه مقادیر، آن را اسکن می‌کنید، فیلتر می‌کنید، یا جمع مقادیر داخل آن را انجام می‌دهید. همه چیز بر اساس دو مفهوم ساده جداول و روابط استوار است.

به زودی متوجه خواهید شد که از نقطه‌نظر مدل‌سازی، Tabular گزینه‌های کمتری نسبت به Multidimensional ارائه می‌دهد. در این مورد، داشتن گزینه‌های کمتر به معنای قدرت کم نیست زیرا می‌توانید از دکس به عنوان یک زبان برنامه‌نویسی برای غنی سازی مدل استفاده کنید. قدرت واقعی مدل سازی Tabular سرعت فوق‌العاده دکس است. در واقع، احتمالاً سعی می‌کنید از استفاده بیش از حد از MDX در مدل خود اجتناب کنید زیرا بهینه سازی سرعت MDX اغلب یک چالش است. از سوی دیگر، دکس به طرز شگفت‌انگیزی سریع است. بنابراین، بیشتر پیچیدگی محاسبات در مدل نیست، بلکه در فرمول‌های دکس است.

دکس به عنوان یک زبان برنامه‌نویسی و کوئری

دکس و MDX هر دو زبان برنامه‌نویسی و زبان کوئری هستند. در MDX، تفاوت با وجود اسکریپت[۱] MDX مشخص می‌شود. شما از MDX در اسکریپت MDX به همراه چندین دستور خاص که فقط در اسکریپت قابل استفاده هستند، مانند دستورات *SCOPE* استفاده می‌کنید. هنگام نوشتن دستورات *SELECT* که داده‌ها را بازیابی می‌کنند، از MDX در کوئری‌ها استفاده می‌کنید. در دکس، این تا حدودی متفاوت است. شما از دکس به عنوان یک زبان برنامه‌نویسی برای تعریف ستون‌های محاسبه‌شده، جداول محاسبه‌شده و معیارها استفاده می‌کنید. مفهوم ستون‌های محاسبه‌شده و جداول محاسبه‌شده برای دکس جدید است و در MDX وجود ندارد؛ معیارها مشابه اعضای محاسبه‌شده در MDX می‌باشند. همچنین می‌توانید از دکس به‌عنوان یک زبان کوئری استفاده کنید—برای مثال، برای بازیابی داده‌ها از یک مدل Tabular با استفاده از Reporting Services. با این وجود، توابع دکس نقش خاصی ندارند و می‌توانند هم در کوئری‌ها و هم در عبارات محاسباتی استفاده شوند. علاوه بر این، شما می‌توانید یک مدل Tabular را با استفاده از MDX نیز کوئری کنید. بنابراین، بخش کوئری نویسی MDX با مدل‌های Tabular کار می‌کند، در حالی که دکس تنها گزینه در برنامه‌نویسی یک مدل Tabular است.

سلسله‌مراتب‌ها

شما با استفاده از MDX، برای انجام بیشتر محاسبات به سلسله‌مراتب تکیه می‌کنید. اگر می‌خواهید فروش سال قبل را محاسبه کنید، باید *PrevMember* از *CurrentMember* در سلسله‌مراتب Year را بازیابی کنید و از آن برای لغو فیلتر MDX استفاده کنید. به عنوان مثال، می‌توانید این فرمول را به این شکل بنویسید تا یک محاسبه سال قبل را در MDX تعریف کنید:

---

[۱] - *Script*

```
CREATE MEMBER CURRENTCUBE.[Measures].[SamePeriodPreviousYearSales] AS
(
    [Measures].[Sales Amount],
    ParallelPeriod (
        [Date].[Calendar].[Calendar Year],
        1,
        [Date].[Calendar].CurrentMember
    )
);
```

این معیار از تابع *ParallelPeriod* استفاده می‌کند که پسر عموی *CurrentMember* را در سلسله‌مراتب *Calendar* برمی‌گرداند. بنابراین، بر اساس سلسله‌مراتب تعریف شده در مدل است. ما همان محاسبه را در دکس با استفاده از زمینه‌های فیلتری و توابع استاندارد هوش زمانی می‌نویسیم:

```
SamePeriodPreviousYearSales :=
CALCULATE (
    SUM ( Sales[Sales Amount] ),
    SAMEPERIODLASTYEAR ( 'Date'[Date] )
)
```

ما می‌توانیم با استفاده از *FILTER* و دیگر توابع دکس، همان محاسبه را به روش‌های بسیار دیگری بنویسیم، اما ایده یکسان است: به جای استفاده از سلسله‌مراتب، جداول را فیلتر می‌کنیم. این تفاوت، بسیار بزرگ است و احتمالاً زمانی که به دکس عادت کنید، محاسبات سلسله‌مراتبی را از دست خواهید داد.

تفاوت مهم دیگر این است که در MDX شما به *[Measures] [Sales Amount]* مراجعه می‌کنید و تابع تجمیع که باید استفاده کنید قبلاً در مدل تعریف شده است. در دکس، هیچ تجمیعی از پیش تعریف شده‌ای وجود ندارد. در واقع، همان‌طور که ممکن است متوجه شده باشید، عبارت مربوط به محاسبه *SUM (Sales[Sales Amount])* است. تجمیع از پیش تعریف‌شده دیگر در مدل وجود ندارد. هر زمان که بخواهیم از آن استفاده کنیم باید آن را تعریف کنیم. ما همیشه می‌توانیم معیاری ایجاد کنیم که مجموع فروش را محاسبه کند، اما این امر فراتر از محدوده این بخش است و بعداً در کتاب توضیح داده می‌شود.

یک تفاوت مهم دیگر بین دکس و MDX این است که MDX به شدت از دستور *SCOPE* برای پیاده‌سازی منطق تجاری (دوباره، استفاده از سلسله‌مراتب) استفاده می‌کند، در حالی که دکس به رویکرد کاملاً متفاوتی نیاز دارد. در واقع، مدیریت سلسله‌مراتب در این زبان به طور کلی وجود ندارد.

به عنوان مثال، اگر بخواهیم معیاری را در سطح Year مشخص کنیم، در MDX این دستور را می‌نویسیم:

```
SCOPE ( [Measures].[SamePeriodPreviousYearSales], [Date].[Month].[All] )
   THIS = NULL;
END SCOPE;
```

دکس چیزی شبیه به عبارت *SCOPE* ندارد. برای به دست آوردن همان نتیجه، ما باید وجود فیلترها را در زمینه فیلتری بررسی کنیم، و این سناریو بسیار پیچیده‌تر است:

```
SamePeriodPreviousYearSales :=
IF (
    ISINSCOPE ( 'Date'[Month] ),
    CALCULATE (
        SUM ( Sales[Sales Amount] ),
        SAMEPERIODLASTYEAR ( 'Date'[Date] )
    ),
    BLANK ()
)
```

به طور شهودی، این فرمول تنها زمانی مقداری را برمی‌گرداند که کاربر در حال مرور سلسله‌مراتب تقویم در سطح ماه یا پایین‌تر باشد. در غیر این صورت یک *BLANK* برمی‌گرداند. بعداً خواهید آموخت که این فرمول به طور دقیق چه چیزی را محاسبه می‌کند. آن نسبت به کد MDX معادل بسیار مستعد خطا است. صادقانه بگوییم، مدیریت سلسله‌مراتب یکی از ویژگی‌هایی است که واقعاً در دکس وجود ندارد.

محاسبات سطح برگ

در نهایت، هنگام استفاده از MDX، احتمالاً به اجتناب از محاسبات سطح برگ[1] عادت کرده‌اید. انجام محاسبات سطح برگ در MDX به قدری کند است که همیشه باید ترجیح دهید مقادیر را از قبل محاسبه کنید و از تجمیعات برای برگرداندن نتایج استفاده کنید. در دکس، محاسبات سطح برگ بسیار سریع کار می‌کنند و تجمیع‌ها هدف متفاوتی را دنبال می‌کنند و فقط برای مجموعه‌داده‌های بزرگ مفید هستند. این نیاز به یک تغییر در ذهن شما دارد هنگامی که زمان ساخت مدل‌های داده است. در بیشتر موارد، مدل داده‌ای که کاملاً در SSAS Multidimensional مطابقت دارد، برای Tabular مناسب نیست و بالعکس.

دکس برای کاربران Power BI

اگر شما بخش‌های قبلی را رد کرده‌اید و مستقیم به این بخش آمده‌اید، خوش آمده‌اید! دکس زبان مادری Power BI می‌باشد، و اگر شما تجربه‌ای در خصوص Excel، SQL، یا MDX ندارید، Power BI اولین جایی خواهد بود که شما دکس را یاد می‌گیرید. اگر شما تجربه قبلی در ساخت مدل‌ها در دیگر ابزارها ندارید، یاد خواهید گرفت که Power BI یک ابزار قدرتمند تحلیل و مدل‌سازی می‌باشد، با همراهی دکس به عنوان رفیق تمام عیار آن.

شاید شما استفاده از Power BI را از کمی قبل آغاز کرده باشید و هم‌اکنون خواهان رسیدن به مرحله بعدی باشید. اگر این چنین است، آماده یک سفر شگفت‌انگیز همراه با دکس باشید.

توصیه ما به شما این است: انتظار نداشته باشید که بتوانید کدهای پیچیده دکس را در عرض چند روز بنویسید. دکس نیازمند زمان و تلاش زیادی می‌باشد، و خبرگی در آن نیازمند تمرین. بر اساس تجربه ما، در ابتدا شما هیجان‌زده خواهید شد زمانی که با تعداد کمی از محاسبات ساده پاداش گرفتید.

این هیجان به محض اینکه شما شروع به یادگیری درباره زمینه‌های ارزیابی و *CALCULATE*، پیچیده‌ترین موضوعات این زبان، کردید کم رنگ می‌شود. در آن نقطه، همه چیز پیچیده به نظر می‌رسد. منصرف نشوید؛ اکثر توسعه‌دهندگان دکس مجبور بودند از آن سطح عبور کنند. هنگامی که در آن نقطه قرار دارید، بسیار نزدیک به حصول درک کاملی هستید که متوقف کردن آن واقعاً حیف خواهد بود. بارها و بارها بخوانید و تمرین کنید چرا که یک لامپ خیلی زودتر از آنچه که انتظار دارید خاموش می‌شود. شما به سرعت قادر به اتمام این کتاب و رسیدن به وضعیت مربیگری دکس خواهید بود.

زمینه‌های ارزیابی هسته اصلی این زبان می‌باشند. تسلط بر آن‌ها زمان می‌برد. ما کسی را نمی‌شناسیم که قادر به یادگیری دکس طی چند روز باشد. علاوه‌براین، همانند هر موضوع پیچیده دیگری، شما یاد خواهید گرفت که جزئیات بسیار زیادی را در طول زمان درک می‌کنید. زمانی که فکر می‌کنید همه چیز را یاد گرفته‌اید، دومرتبه این کتاب را بخوانید. شما جزئیات بسیاری را کشف خواهید کرد که در نگاه اول کم اهمیت به نظر می‌آیند اما، با یک نگرش پذیرای یادگیری بیشتر، واقعاً تفاوت ایجاد کنید.

از ادامه این کتاب لذت ببرید!

---

[1] – *Leaf-Level Calculations*

فصل ۲ معرفی دکس

در این فصل، ما شروع به صحبت در مورد زبان دکس خواهیم کرد. در اینجا شما ترکیب این زبان، تفاوت بین یک ستون محا سبه شده[۱] و یک معیار[۲] (در برخی نسخه‌های قدیمی *Excel* فیلد محا سبه شده نامیده می شود)، و متداول‌ترین توابع در دکس را یاد می‌گیرید.

از آنجایی که این فصل یک فصل مقدماتی می‌باشد، بسیاری از توابع را به طور عمیق پوشش نمی‌دهد. در فصول بعدی ما آن‌ها را با جزئیات بی شتری تو ضیح خواهیم داد. در حال حا ضر، معرفی این توابع و آغاز برر سی زبان دکس به صورت کلی کافی می‌باشد. زمانی که ما به ویژگی‌های مدل داده در *Power BI*، *Power Pivot*، یا *Analysis Services* اشاره می‌کنیم، از اصطلاح Tabular استفاده می‌کنیم حتی زمانی که این ویژگی در تمام این محصولات ارائه نشده باشد. برای مثال، «*DirectQuery in Tabular*» اشاره به وضعیت ویژگی *DirectQuery* دارد که در *Power BI* و *Analysis Services* در دسترس می‌باشد اما در *Excel* نه.

شناخت محاسبات دکس

قبل از آغار کار بر روری فرمول‌های پیچیده‌تر، شما نیازمند یادگیری مبانی دکس هستید. این مبانی شامل ترکیب دکس، انواع متفاوت داده‌هایی که دکس می‌تواند پوشش دهد، عملگرهای اصلی، و نحوه ارجاع به ستون‌ها و جداول می‌باشد. این مفاهیم در چند بخش بعدی مورد بحث قرار خواهند گرفت.

ما از دکس برای محاسبه مقادیر بر روی ستون‌ها در جداول استفاده می‌کنیم. ما در مورد اعداد می‌توانیم تجمیع، محاسبه، و جستوجو کنیم اما، در نهایت، تمام این محاسبات شامل جداول و ستون‌ها می‌باشند. بنابراین، اولین ترکیب برای یادگیری چگونگی ارجاع به یک ستون در یک جدول می‌باشد.

قالب کلی به شکل نو شتن نام جدول درون علامت نقل‌قول ' ' و به دنبال آن نام ستون در داخل براکت [ ] می‌با شد، همانند مثال زیر:

```
'Sales'[Quantity]
```

اگر نام جدول با عدد شروع نشود، شامل فاصله نباشد، و نیز یک کلمه رزروشده نباشد (همانند *Date, sum*) می‌توانیم علامت نقل‌قول را حذف کنیم.

همچنین ذکر نام جدول در حالتی که ما در حال ارجاع دادن به یک ستون یا معیاری در داخل همان جدولی هستیم که فرمول را در داخل آن تعریف می‌کنیم اختیاری می‌باشد. بنابراین *[Quantity]* یک ارجاع ستونی معتبر می‌باشد، اگر در یک ستون محاسبه شده یا در یک معیار تعریف شده در جدول *Sales* نو شته شود. اگرچه این گزینه در د سترس می‌با شد ما به جد شما را ترغیب می‌کنیم که از حذف نام جدول پرهیز کنید. در این لحظه، توضیح نمی‌دهیم که چرا این امر خیلی مهم می‌باشد،

---

[۱] - *Calculated Column*

[۲] - *Measure*

اما دلیل آن زمانی که شما فصل ۵ «شناخت *CALCULATE* و *CALCULATETABLE*» را مطالعه کردید روشن می شود. با وجود این، زمانی که شما کد دکس را می‌خوانید، توانایی تشخیص بین معیارها (در ادامه بحث خواهد شد) و ستون‌ها بسیار مهم می‌باشد. استاندارد عملی این است که همیشه از نام جدول در ارجاع به ستون‌ها استفاده شود و همواره از آن در رجوع به معیارها اجتناب شود. هر چقدر زودتر شما شروع به سازگار شدن با این استاندارد کنید، زندگی شما با دکس ساده‌تر می‌شود. بنابراین، شما باید عادت کنید به این شیوه به ستون‌ها و معیارها ارجاع دهید:

```
Sales[Quantity] * 2          -- این یک ارجاع ستونی است
[Sales Amount] * 2           -- این یک ارجاع معیاری است
```

منطق پشت سر این استاندارد را بعد از یادگیری مفهوم انتقال زمینه‌ای فراخواهید گرفت، که در فصل ۵ ارائه می‌شود. برای این لحظه، فقط به ما اعتماد کنید و به این استاندارد پایبند باشید.

توضیحات در دکس

کد مثال قبلی توضیحاتی در خصوص دکس را برای اولین بار نشان می‌دهد. دکس از توضیحات تک خطی و چند خطی پشتیبانی می‌کند. توضیحات تک خطی با -- و یا // شروع می‌شوند، و مابقی خط یک توضیح را شکل می‌دهد.

```
= Sales[Quantity] * Sales[Net Price]    -- توضیح تک خطی
= Sales[Quantity] * Sales[Unit Cost]    //مثال دیگری از توضیح تک خطی
```

یک توضیح چند خطی با * / شروع می‌شوند و با / * به پایان می‌رسد. تجزیه‌گر دکس هر چیزی را که بین این علامت‌ها باشد نادیده می‌گیرد و آن‌ها را به عنوان یک توضیح در نظر می‌گیرد.

```
= IF (
    Sales[Quantity] > 1,
    /* اولین مثال از یک توضیح چند خطی
       هر چیزی می تواند در اینجا نوشته شود اما توسط دکس‌نادیده گرفته می شود
    */
    "Multi",
    /* یک مورد رایج استفاده از توضیحات چند خطی برای تبدیل بخشی از کد به توضیح است
       دستور IF بعدی نادیده گرفته می شود زیرا درون یک توضیح چند خطی قرار دارد

        IF (
            Sales[Quantity] = 1,
            "Single",
            "Special note"
        )
    */
    "Single"
)
```

بهتر است از ارائه توضیحات در انتهای یک عبارت دکس در تعریف یک معیار، ستون محاسبه‌شده، یا جدول محاسبه‌شده اجتناب شود. شاید این توضیحات در ابتدا قابل رویت نباشند، و شاید آن‌ها به وسیله برخی ابزارها مثل *DAX FORMATTER* پشتیبانی نشوند، که در ادامه این فصل توضیح داده می‌شوند.

انواع داده در دکس

دکس می‌تواند محاسبات را با انواع متفاوت داده‌های عددی انجام دهد که شامل هفت نوع می‌باشد. در گذر زمان، *Microsoft*

نام‌های متفاوتی برای همان نوع داده ارائه کرده است، که تا حدی منجر به سردرگمی شده است. جدول ۲-۱ اسامی متفاوتی را ارائه می‌دهد که تحت هر یک شما می‌توانید نوع داده مربوط به آن را در دکس پیدا کنید.

جدول ۲-۱ انواع داده

نوع داده Tabular Object Model (TOM)	نوع داده Correspondent Conventional (SQL Server)	نوع داده Power Pivot و Analysis Services	نوع داده Power BI	نوع داده دکس
Int64	Integer/INT	Whole Number	Whole Number	Integer
Double	Floating point/DOUBLE	Dacimal Number	Dacimal Number	Dacimal Number
Decimal	Currency/MONEY	Currency	Fixed Dacimal Number	Currency
dateTime	Date/DATETIME	Date	DateTime,Date,Time	DateTime
Boolean	Boolean/BIT	True/False	True/False	Boolean
String	String/NVARCHAR(MAX)	Text	Text	String
Variant	-	-	-	Variant
binary	Blob/VARBINARY(MAX)	Binery	Binery	Binery

در این کتاب، ما از اسامی اولین ستون جدول ۲-۱ [از سمت چپ] به دلیل پایبندی به استاندارد عملی در جامعه پایگاه‌داده و هوش تجاری استفاده می‌کنیم. برای مثال، در *Power BI*، ستونی که در بر دارنده هم TRUE و هم FALSE می‌باشد TRUE/FALSE خوانده می‌شود، در حالی که در SQL Server، آن یک BIT نامیده می‌شود. با این وجود، نام تاریخی و متداول‌تر برای این نوع از مقادیر *Boolean* می‌باشد.

دکس دارای یک سیستم قدرتمند پوشش‌دهنده انواع داده می‌باشد به همین خاطر ما نگرانی درباره نوع داده نخواهیم داشت. در یک عبارت دکس، نوع داده نتیجه بر اساس نوع شرط استفاده شده در عبارت تعیین می شود. شما تحت شرایطی نیاز به هوشیاری نسبت به این موضوع دارید که نوع داده بازگردانده شده حاصل از یک عبارت دکس نوع داده مورد انتظار شما نمی‌باشد: در نتیجه می‌بایست نوع داده شرط‌های استفاده شده در خود عبارت موردنظر را بررسی کنید.

برای مثال، اگر یکی از شروط یک جمع تاریخ باشد، نتیجه نیز تاریخ می‌باشد، به همین طریق اگر همان عملگر با اعداد صحیح استفاده شود، نتیجه یک عدد صحیح خواهد بود. این رفتار با عنوان *سرریز عملگر*[۱] شناخته می شود، در این خصوص مثالی در شکل ۲-۱، نشان داده شده است، جایی که ستون OrderDatePlusOneWeek با اضافه کردن ۷ به مقدار ستون Order Date محاسبه می‌شود.

```
Sales[OrderDatePlusOneWeek] = Sales[Order Date] + 7
```

نتیجه یک تاریخ می‌باشد.

Order Date	OrderDatePlusOneWeek
10/08/2008	10/15/2008
10/10/2008	10/17/2008
10/12/2008	10/19/2008
09/05/2008	09/12/2008
09/07/2008	09/14/2008
09/23/2008	09/30/2008
11/05/2008	11/12/2008
11/07/2008	11/14/2008
11/09/2008	11/16/2008
11/17/2008	11/24/2008

شکل ۲-۱ اضافه کردن یک عددصحیح به یک تاریخ منجر به ایجاد تاریخی می‌شود که به تعداد همین عدد روزهای آن افزایش می‌یابد.

علاوه بر سرریز عملگر، دکس به‌طور خودکار هر زمان که از جانب عملگر[۲] نیاز باشد رشته‌ها را به اعداد و اعداد را به رشته تبدیل می‌کند. برای مثال، اگر ما از عملگر & استفاده کنیم، عملگری که رشته را ترکیب و یکی می‌کند، دکس

---

۱ - *Operator Overloading*

۲ – *Operator*

آرگومان‌های[۱] خود را به رشته یا متن تبدیل می‌کند. فرمول زیر "۵۴" را به‌صورت رشته باز می‌گرداند:

```
= 5 & 4
```

از طرف دیگر، همین فرمول در حالت زیر یک عددصحیح باز می‌گرداند که مقدار آن ۹ می‌باشد:

```
= "5" + "4"
```

مقدار به دست آمده وابسته به عملگر می‌باشد و نه به ستون‌های منبع، ستون‌هایی که به دنبال نیازمندی‌های عملگر تبدیل می‌شوند. اگرچه این رفتار رضایت‌بخش به نظر می‌رسد، بعداً در این فصل خواهید دید که چه نوع خطاهایی ممکن است در طول این تغییرات خودکار رخ دهد. به علاوه، همه عملگرها از این رفتار پیروی نمی‌کنند، برای مثال، عملگرهای مقایسه‌ای نمی‌توانند متن‌ها را با اعداد مقایسه کنند. در نتیجه، شما می‌توانید یک عدد را به یک رشته اضافه کنید، اما نمی‌توانید یک عدد را با یک رشته مقایسه کنید. شما می‌توانید در این آدرس مرجع کاملی راجع به این موضوع پیدا کنید: https://docs.microsoft.com/en-us/power-bi/desktop-data-types. از آنجایی که قوانین در این خصوص بسیار پیچیده می‌باشند، پیشنهاد می‌کنیم به طور کلی از تبدیلات خودکار اجتناب کنید. اگر لازم است تبدیلی رخ دهد، ما توصیه می‌کنیم که شما بر روی آن کنترل داشته باشید و آن تبدیل را صریح کنید. به منظور صراحت بیشتر، مثال قبلی می‌بایست به شکل زیر نوشته شود:

```
= VALUE ( "5" ) + VALUE ( "4" )
```

افرادی که عادت به کار با *Excel* یا دیگر زبان‌ها کرده‌اند ممکن است با انواع داده‌های دکس آشنا باشند. برخی جزئیات درباره انواع داده‌ها وابسته به موتور مدنظر می‌باشد، و شاید آن‌ها برای *Power BI*، *power Pivot* یا *Analysis Services* متفاوت باشند. شما می‌توانید در آدرس *http://msdn.microsoft.com/en-us/library/gg492146.aspx* اطلاعات دقیق‌تری در خصوص انواع داده *Analysis Services DAX* پیدا کنید و اطلاعات مربوط به *Power BI* در آدرس *https://docs.microsoft.com/en-us/power-bi/desktop-data-types* در دسترس می‌باشد. به‌هرحال، در اینجا به اشتراک گذاشتن تعدادی از ملاحظات مربوط به هر یک از این انواع داده سودمند می‌باشد.

عددصحیح[۲]

دکس فقط یک نوع داده ***عددصحیح*** دارد که می‌تواند یک مقدار ۶۴ بیتی را ذخیره کند. تمام محاسبات داخلی بین مقادیر عددصحیح در دکس نیز از یک مقدار ۶۴ بیتی استفاده می‌کنند.

اعداد اعشاری[۳]

یک ***عدد اعشاری*** همیشه به صورت یک مقدار ممیز شناور با دقت دو ذخیره می شود. این نوع داده دکس را با نوع داده‌های ***اعشاری*** و ***عددی*** Transact-SQL اشتباه نگیرید: نوع داده متناظر با عدد اعشاری دکس در *SQL* نوع داده شناور[۴] می‌باشد.

---

۱ - *Arguments*

۲ - *Integer*

۳ - *Decimal*

۴ - *Float*

واحدپولی[۱]

نوع داده ***واحدپولی***، با عنوان ***عدد اعشاری ثابت‌شده***[۲] در *Power BI* نیز شناخته می‌شود، یک عدد اعشاری ثابت‌شده را ذخیره می‌کند. می‌تواند چهار رقم اعشار را نشان دهد و از نظر داخلی به صورت یک مقدار عدد صحیح ۶۴ بیتی تقسیم بر ۱۰،۰۰۰ ذخیره می‌شود. جمع یا تفریق داده‌های از نوع *واحدپولی* همیشه اعشارهای بعد از چهارمین رقم اعشار را نادیده می‌گیرد، درحالی‌که ضرب و تقسیم یک مقدار ممیز شناور را ایجاد می‌کند، بنابراین دقت نتیجه را افزایش می‌دهد. به‌طورکلی، اگر ما نیازمند دقت بیشتری از چهار رقم اعشار ارائه شده باشیم، مجبوریم از نوع داده اعشاری استفاده کنیم.

قالب پیش‌فرض نوع داده ***واحدپولی*** شامل نماد واحدپولی می‌باشد. همچنین ما می‌توانیم قالب‌بندی واحدپولی را بر اعداد صحیح و اعشاری اعمال کنیم، و می‌توانیم از یک قالب بدون نماد واحدپولی برای نوع داده واحدپولی استفاده کنیم.

نوع داده *DateTime*[۳]

دکس تاریخ‌ها را در یک نوع داده **DateTime** ذخیره می‌کند. این قالب به‌صورت داخلی از یک عدد ممیز شناور استفاده می‌کند، که در آن عددصحیح مربوط به تعداد روزها از روز ۳۰ دسامبر سال ۱۸۹۹ می‌باشد، و جزء اعشاری قسمت‌های روز را مشخص می‌کند. ساعت‌ها، دقیقه‌ها، و ثانیه‌ها به قسمت‌های اعشاری یک روز تبدیل می‌شوند. بنابراین، عبارت زیر تاریخ جاری به اضافه یک روز (دقیقاً ۲۴ ساعت) را باز می‌گرداند:

```
= TODAY () + 1
```

نتیجه، تاریخ فردا در زمان ارزیابی می‌باشد. اگر شما نیاز دارید که فقط قسمت تاریخ یک *DateTime* را بردارید، همیشه به یاد داشته باشید که از تابع TRUNC برای رها شدن از قسمت اعشاری استفاده کنید.

Power BI دو نوع داده دیگر را نیز ارائه می‌دهد: تاریخ و زمان. به طور داخلی آن‌ها نوع ساده‌ای از نوع داده *DateTime* می‌باشند. در عمل، تاریخ و زمان به ترتیب فقط بخش عددصحیح یا اعشاری نوع داده *DateTime* را ذخیره می‌کنند.

مسئله سال کبیسه

نرم افزار لوتوس ۱-۲-۳، یک صفحه گسترده محبوب که در سال ۱۹۸۳ منتشر شد، یک مشکل در پوشش نوع داده‌های *DateTime* داشت. این نرم‌افزار سال ۱۹۰۰ را به عنوان یک سال کبیسه در نظرگرفت، درصورتی که آن یک سال کبیسه نبود. آخرین سال قرن فقط در صورتی که بتوان دو رقم اول را بدون باقیمانده بر چهار تقسیم کرد یک سال کبیسه می‌باشد. در آن زمان، تیم توسعه‌دهنده اولین نسخه اکسل برای حفظ قابلیت سازگاری با لوتوس۱-۲-۳ به طور عمد این مسئله را تکرار کرد. از آن زمان به بعد، هر نسخه جدید اکسل به خاطر قابلیت سازگاری این مشکل را حفظ کرد.

در زمان چاپ این کتاب در سال ۲۰۱۹، این مسئله هنوز در دکس هم به خاطر حفظ قابلیت سازگاری عقب‌گردانه با اکسل وجود دارد. وجود این مسئله (آیا ما باید آن را یک ویژگی بنامیم؟) ممکن است منجر به خطاهایی در دوره‌های زمانی قبل از ۱ مارس سال ۱۹۰۰ شود. بنابراین، طبق طراحی، اولین تاریخی که به طور رسمی به وسیله دکس پشتیبانی می‌شود ۱ مارس سال ۱۹۰۰ می‌باشد. محاسبات تاریخ که بر روی دوره‌های زمانی قبل از آن تاریخ اجرا می‌شوند ممکن است منجر به خطایی شوند و می‌بایست به عنوان یک

---

۱ - *Currency*

۲ - *Fixed Decimal Number*

۳ - *Datetime*

عدم دقت مدنظر باشد.

بولین[۱]

نوع داده *بولین* برای بیان شرط‌های منطقی استفاده می‌شود. برای مثال، یک ستون محاسبه‌شده که با عبارت زیر تعریف شده است از نوع داده بولین می‌باشد:

```
= Sales[Unit Price] > Sales[Unit Cost]
```

همچنین شما نوع داده بولین را به‌صورت اعداد در جایی خواهید دید که *TRUE* برابر با ۱ و *FALSE* برابر با ۰ می‌باشد. این نوع علامت‌گذاری گاهی می‌تواند برای هدف مرتب‌سازی سودمند باشد چرا که *TRUE>FALSE* می‌باشد.

رشته[۲]

هر رشته‌ای در دکس به‌صورت یک رشته یونیکد[۳] ذخیره می‌شود، که در آن هر کاراکتر در ۱۶ بیت ذخیره می‌گردد. به‌طور پیش‌فرض، مقایسه بین رشته‌ها به حروف بزرگ یا کوچک حساس نمی‌باشد، بنابراین دو رشته «*Power BI*» و «*BI POWER*» برابر در نظر گرفته می‌شوند.

تغییرپذیر[۴]

داده‌های از نوع *تغییرپذیر* برای عباراتی استفاده می‌شود که با توجه به شرایط ممکن است انواع مختلف داده را برگردانند. برای مثال، دستور زیر می‌تواند هم یک عددصحیح و یا یک رشته را بازگرداند، بنابراین یک داده از نوع تغییرپذیر را باز می‌گرداند:

```
IF ( [measure] > 0, 1, "N/A" )
```

داده‌هایی از نوع تغییرپذیر نمی‌توانند به عنوان یک نوع داده برای ستونی در یک جدول معمولی استفاده شوند. یک معیار دکس، و به طور کلی، یک عبارت دکس می‌تواند از نوع داده‌های تغییرپذیر باشد.

باینری[۵]

داده‌هایی از نوع *باینری* در مدل داده برای ذخیره تصاویر یا دیگر انواع اطلاعات ساختارنیافته استفاده می‌شوند. این نوع از داده‌ها در دکس قابل دسترس نمی‌باشند. آن‌ها عمدتاً به وسیله *Power View* استفاده می‌شوند، اما شاید در دیگر ابزارها همانند *Power BI* در دسترس نباشند.

عملگرهای دکس

اکنون که شما اهمیت عملگرها را در تعیین نوع یک عبارت مشاهده کرده‌اید، جدول ۲-۲ را مشاهده بفرمایید که فهرستی از

---

۱ - *Boolean*

۲ - *String*

۳ - *Unicode*

۴ - *Variant*

۵ - *Binary*

عملگرهای قابل دسترس در دکس را ارائه می‌دهد.

جدول ۲-۲ عملگرها

نوع عملگر	نشانه	کاربرد	مثال
پرانتز	()	تعیین اولویت و گروه‌بندی آرگومان‌ها	۳*(۲+۵)
حسابی	+	جمع	4+2
	-	تفریق/ علامت منفی	5-3
	*	ضرب	4*2
	/	تقسیم	4/2
مقایسه	=	برابر است با	[CountryRegion] = "USA"
	<>	برابر نیست با	[CountryRegion] <> "USA"
	>	بزرگ‌تر از	[Quantity] > 0
	=>	بزرگ‌تر مساوی	[Quantity] >= 100
	<	کوچک‌تر از	[Quantity] < 0
	=<	کوچک‌تر مساوی	[Quantity] <= 100
ترکیب متن	&	ترکیب داده‌های رشته‌ای	"Value is" & [Amount]
منطقی	&&	شرط and بین دو عبارت بولین	[CountryRegion] = "USA" && [Quantity]>0
	||	شرط or بین دو عبارت بولین	[CountryRegion] = "USA" || [Quantity] > 0
	IN	گنجاندن عنصری در یک فهرست بولین	[CountryRegion] IN {"USA", "Canada"}
	NOT	نفی بولین	NOT [Quantity] > 0

علاوه بر این، عملگرهای منطقی همچنین به عنوان توابع دکس با ترکیبی مشابه با ترکیب *Excel* در دسترس می‌باشند. برای مثال، ما می‌توانیم عبارت‌هایی مثل این را بنویسم:

```
AND ( [CountryRegion] = "USA", [Quantity] > 0 )
OR ( [CountryRegion] = "USA", [Quantity] > 0 )
```

این مثال‌ها به ترتیب برابر با مثال‌های زیر می باشند:

```
[CountryRegion] = "USA" && [Quantity] > 0
[CountryRegion] = "USA" || [Quantity] > 0
```

استفاده از توابع به جای عملگرها برای منطق بولین زمانی که شما در حال نوشتن شرط‌های پیچیده هستید سودمند می‌باشد. در حقیقت وقتی صحبت از قالب‌بندی بخش‌های بزرگ کد می‌شود، قالب‌بندی و خواندن توابع بسیار ساده‌تر از عملگرها می‌با شد. با این‌حال نقطه ضعف ا صلی توابع این می‌با شد که فقط دو پارامتر را می‌توانیم در یک زمان ردوبدل کنیم. بنابراین، اگر بیش از دو شرط برای ارزیابی داشته باشیم می‌بایست از توابع تودرتو[۱] استفاده کنیم.

سازندگان جدول

در دکس ما می‌توانیم جداول بی‌نامی را به طور مستقیم در کد مورد نظر تعریف کنیم. اگر جدول مورد نظر دارای تنها یک ستون با شد، این ترکیب فقط نیاز به فهر ستی از مقادیر-یک مقدار برای هر ردیف–دارد که از طریق آکولاد مشخص می شوند. ما می‌توانیم چند ردیف را از طریق پرانتز مشخص کنیم، اما اگر این جدول تنها یک ستون داشته باشد، پرانتز اختیاری می‌شود. برای مثال، دو تعریف زیر، برابر می‌باشند:

[۱] - *Nest Functions*

```
{ "Red", "Blue", "White" }
{ ( "Red" ), ( "Blue" ), ( "White" ) }
```

اگر جدول مورد نظر چند ستون داشته باشد، پرانتزها الزامی می شوند. هر ستون می‌بایست نوع داده یکسانی در کل ردیف‌هایش داشته باشد؛ در غیر این صورت، دکس به طور خودکار این ستون را تبدیل به نوع داده‌ای می‌کند که می‌تواند تمام انواع داده‌های ارائه شده در ردیف‌های مختلف همان ستون را در خود جای دهد.

```
{
    ( "A", 10, 1.5, DATE ( 2017, 1, 1 ), CURRENCY ( 199.99 ), TRUE ),
    ( "B", 20, 2.5, DATE ( 2017, 1, 2 ), CURRENCY ( 249.99 ), FALSE ),
    ( "C", 30, 3.5, DATE ( 2017, 1, 3 ), CURRENCY ( 299.99 ), FALSE )
}
```

سازنده جدول به طور معمول با عملگر IN مورد استفاده قرار می‌گیرد. برای مثال موارد زیر ترکیبات معتبر محتملی در یک گزاره[۱] دکس می‌باشند:

```
'Product'[Color] IN { "Red", "Blue", "White" }

( 'Date'[Year], 'Date'[MonthNumber] ) IN { ( 2017, 12 ), ( 2018, 1 ) }
```

این مثال دوم ترکیب مورد نیاز برای مقایسه یک مجموعه از ستون‌ها (چندتایی[۲]) را با استفاده از عملگر IN را نشان می‌دهد. چنین ترکیبی نمی‌تواند با یک عملگر مقایسه‌ای استفاده شود. به عبارت دیگر، ترکیب زیر معتبر نمی‌باشد:

```
( 'Date'[Year], 'Date'[MonthNumber] ) = ( 2007, 12 )
```

به هر حال، ما می‌توانیم آن را با استفاده از عملگر IN همراه با یک سازنده جدول که تنها یک ردیف دارد باز نویسی کنیم، همانند مثال زیر:

```
( 'Date'[Year], 'Date'[MonthNumber] ) IN { ( 2007, 12 ) }
```

دستورات شرطی

در دکس ما می‌توانیم یک عبارت شرطی را با استفاده از تابع IF بنویسیم. برای مثال، ما می‌توانیم بر اساس مقدار *Quantity* که به ترتیب از یک بزرگ‌تر باشد یا نه، عبارتی بنویسیم که *MULTI* یا *SINGLE* را باز می‌گرداند.

```
IF (
    Sales[Quantity] > 1,
    "MULTI",
    "SINGLE"
)
```

تابع IF سه پارامتر دارد، اما فقط دو پارامتر اول آن الزامی می‌باشند. سومین پارامتر اختیاری است، و به طور پیش‌فرض BLANK می‌باشد. کد زیر را در نظر بگیرید:

```
IF (
    Sales[Quantity] > 1,
    Sales[Quantity]
)
```

---

۱ - *Predicate*

۲ - *Tuple*

این کد برابر با نسخه صریح زیر می‌باشد:

```
IF (
    Sales[Quantity] > 1,
    Sales[Quantity],
    BLANK ()
)
```

شناخت ستون‌ها و معیارهای محاسبه‌شده

اکنون که شما مبانی ترکیب دکس را می‌دانید، نیاز به فراگیری یکی از مهم‌ترین مفاهیم در دکس دارید: تفاوت بین ستون‌های محاسبه‌شده[۱] و معیارها[۲]. اگرچه ستون‌های محاسبه‌شده و معیارها ممکن است در نگاه اول شبیه به هم به نظر برسند چرا که شما می‌توانید محا سبات خا صی را با ا ستفاده از هر دوی آن‌ها انجام دهید، اما به واقع، آن‌ها متفاوت می‌با شند. شناخت این تفاوت، کلیدی برای باز کردن قفل قدرت دکس می‌باشد.

ستون‌های محاسبه‌شده

بسته به ابزاری که شما از آن استفاده می‌کنید، می‌توانید یک ستون محاسبه‌شده را به شیوه‌های مختلف ایجاد کنید. در عمل، مفهوم آن یکسان باقی می‌ماند: یک ستون محاسبه‌شده ستون جدیدی است که به مدل شما اضافه می‌شود، اما به جای اینکه از یک منبع داده بارگزاری شود، با توسل به فرمول دکس ایجاد می‌شود.

یک ستون محاسبه‌شده درست شبیه به دیگر ستون‌های درون جدول می‌باشد و ما می‌توانیم از آن در ردیف‌ها، ستون‌ها، فیلترها، یا مقادیر یک ماتریس یا هر نوع گزارش دیگری استفاده کنیم. همچنین اگر نیاز باشد می‌توانیم از یک ستون محا سبه شده برای تعریف یک رابطه ا ستفاده کنیم. عبارت دکس تعریف شده برای یک ستون محا سبه شده در زمینه ردیفی فعلی جدولی که ستون محاسبه‌شده به آن تعلق دارد عمل می‌کند. هر نوع ارجاعی به یک ستون مقدار آن ستون را برای ردیف فعلی باز می‌گرداند. ما نمی‌توانیم به طور مستقیم به مقادیر سایر ردیف‌ها دسترسی داشته باشیم.

اگر شما از حالت پیش‌فرض Import Mode مربوط به *Tabular* استفاده می‌کنید و از *DirectQuery* استفاده نمی‌کنید، یک مفهوم مهم در مورد ستون‌های محاسبه‌شده که می‌بایست به خاطر بسپارید این می‌باشد که این ستون‌ها در طی پردازش پایگاه‌داده محاسبه می‌شوند و سپس در مدل مدنظر ذخیره می‌گردند. این مفهوم شاید عجیب به نظر برسد اگر شما به ستون‌های محاسبه‌شده-*SQL*[۳] عادت داشته باشید (ناپایدار) که در زمان کوئری[۴] ارزیابی می‌شوند و از حافظه استفاده نمی‌کنند. به هر حال، در *Tabular* تمام ستون‌های محا سبه شده فضایی در حافظه ا شغال می‌کنند و در طول پردازش جدول محاسبه می‌شوند.

این رفتار هر زمان که ما ستون‌های محاسبه‌شده پیچیده‌ای ایجاد می‌کنیم سودمند می‌باشد. زمان مورد نیاز برای محا سبه ستون‌های محا سبه شده پیچیده همی شه زمان پردازش[۵] می‌با شد و نه زمان کوئری[۶]، که منجر به تجربه بهتر کاربر می شود. با این وجود، توجه دا شته با شید که یک ستون‌محا سبه شده از *RAM* ارز شمند ا ستفاده می‌کند. برای مثال، اگر ما

---

۱ - *Calculated Columns*

۲ - *Measures*

۳ - *SQL-Computed Columns*

۴ - *Query*

۵ - *Process Time*

۶ - *Query Time*

فرمول پیچیده‌ای برای ستون محاسبه شده داشته باشیم، ممکن است اغوا شویم تا مراحل محاسبه را به ستون‌های میانی[۱] متفاوت تجزیه کنیم. اگرچه این تکنیک در طول توسعه پروژه سودمند می‌باشد، اما عادت بدی در تولید می‌باشد چرا که هر محاسبه میانی در *RAM* ذخیره می‌شود و فضای در دسترس را هدر می‌دهد.

اگر یک مدل بر پایه *DirectQuery* باشد، این رفتار بسیار متفاوت می‌باشد. در حالت *DirectQuery* ستون‌های محاسبه‌شده در لحظه‌ای محاسبه می‌شوند که موتور *Tabular* منبع داده را کوئری می‌کند. این شاید منجر به کوئری‌های سنگینی شود که توسط منبع داده اجرا می‌شوند، بنابراین باعث کندی مدل می‌شود.

محاسبه مدت‌زمان یک سفارش

تصور کنید ما یک جدول Sales داریم که حاوی هم *order dates* و هم *delivery dates* می‌باشد. با استفاده از این دو ستون، ما می‌توانیم تعداد روزهای مربوط به تحویل سفارش را محاسبه کنیم. چرا که تاریخ‌ها به‌صورت تعداد روزهای بعد از ۱۲/۳۰/۱۸۹۹ ذخیره می‌شوند، یک تفریق ساده تفاوت روزهای بین دو تاریخ را محاسبه می‌کند:

```
Sales[DaysToDeliver] = Sales[Delivery Date] - Sales[Order Date]
```

با این وجود، از آنجایی که این دو ستونی که برای تفریق استفاده می‌شوند تاریخ می‌باشند، نتیجه نیز تاریخ می‌باشد. برای ایجاد یک نتیجه عددی، نتیجه را از طریق زیر به یک عددصحیح تبدیل کنید:

```
Sales[DaysToDeliver] = INT ( Sales[Delivery Date] - Sales[Order Date] )
```

نتیجه در شکل ۲-۲ نشان داده شده است.

Order Date	Delivery Date	DaysToDeliver
01/02/2007	01/08/2007	6
01/02/2007	01/09/2007	7
01/02/2007	01/10/2007	8
01/02/2007	01/11/2007	9
01/02/2007	01/12/2007	10
01/02/2007	01/13/2007	11
01/02/2007	01/14/2007	12

شکل ۲-۲ از طریق تفریق دو تاریخ و تبدیل نتیجه به یک عددصحیح، دکس تعداد روزهای بین دو تاریخ را محاسبه می‌کند.

معیارها

ستون‌های محاسبه‌شده سودمند می‌باشند، اما شما می‌توانید در یک مدل دکس محاسبات را به روش دیگری نیز تعریف کنید. هر زمان که نمی‌خواهید مقادیر را برای تک‌تک ردیف‌ها محاسبه کنید و به جای آن خواهان تجمیع مقادیر از ردیف‌های زیاد یک جدول هستید، این محاسبات را سودمند خواهید یافت. آن‌ها *معیار* نامیده می‌شوند.

برای مثال، می‌توانید چند ستون محاسبه‌شده را در جدول Sales برای محاسبه مقدار حاشیه ناخالص[۲] تعریف کنید:

```
Sales[SalesAmount] = Sales[Quantity] * Sales[Net Price]
Sales[TotalCost] = Sales[Quantity] * Sales[Unit Cost]
Sales[GrossMargin] = Sales[SalesAmount] - Sales[TotalCost]
```

چه اتفاقی خواهد افتاد اگر شما قصد دارید حاشیه ناخالص را به صورت درصدی از میزان فروش نشان دهید؟ شما

---

[۱] - *Intermediate*

[۲] - *Gross Margin*

می‌توانید یک ستون محاسبه‌شده با فرمول زیر ایجاد کنید:

```
Sales[GrossMarginPct] = Sales[GrossMargin] / Sales[SalesAmount]
```

این فرمول مقدار صحیح را در سطح ردیف محاسبه می‌کند- همان‌طور که در شکل ۲-۳ می‌توانید مشاهده کنید—اما در سطح مجموع کل[۱] نتیجه به وضوح اشتباه می‌باشد.

SalesKey	SalesAmount	TotalCost	GrossMargin	GrossMarginPct
20070104611301-0002	$72.19	$38.74	$33.45	46.34%
20070104611301-0003	$23.75	$11.50	$12.25	51.58%
20070104611320-0006	$216.57	$116.22	$100.35	46.34%
20070104611320-0007	$23.75	$11.50	$12.25	51.58%
20070104611506-0002	$72.19	$38.74	$33.45	46.34%
20070104611506-0003	$23.75	$11.50	$12.25	51.58%
20070104611914-0002	$64.59	$38.74	$25.85	40.02%
20070104611914-0003	$21.25	$11.50	$9.75	45.88%
20070104611952-0004	$64.59	$38.74	$25.85	40.02%
20070104611952-0005	$21.25	$11.50	$9.75	45.88%
20070104611998-0002	$64.59	$38.74	$25.85	40.02%
20070104611998-0003	$63.75	$34.50	$29.25	45.88%
Total	**$732.23**	**$401.92**	**$330.31**	**551.46%**

شکل ۲-۳ ستون *GrossMarginPct* مقدار صحیحی را در هر ردیف نشان می‌دهد، اما مجموع کل صحیح نمی‌باشد

مقدار نشان داده شده در سطح مجموع کل جمع تک‌تک در صدهای محاسبه شده به صورت ردیف‌به‌ردیف در درون ستون محاسبه‌شده می‌باشد. هنگامی که ما مقدار کل درصدی را محاسبه می‌کنیم، نمی‌توانیم به ستون‌های محاسبه‌شده تکیه کنیم. در عوض، ما نیاز به محاسبه درصد بر اساس جمع ستون‌های منفرد داریم. می‌بایست ما مقدار تجمیع شده را به صورت جمع حاشیه ناخالص تقسیم بر جمع میزان فروش محاسبه کنیم. در این مورد، ما نیاز به محاسبه این نرخ بر حسب تجمیعات داریم، شما نمی‌توانید از تجمیع ستون‌های محاسبه شده استفاده کنید. به عبارت دیگر، ما نسبت جمع‌ها را محاسبه می‌کنیم، نه جمع نسبت‌ها.

تغییر ساده جمع ستون GrossMarginPct به یک میانگین و تکیه بر نتایج آن به همان میزان مورد بالا اشتباه می‌باشد چرا که انجام کار بدین شکل ارزیابی اشتباهی از این درصد ارائه می‌دهد، بدون در نظر گرفتن تفاوت بین مقادیر. نتیجه این مقادیر میانگین‌شده در شکل ۲-۴ قابل مشاهده می‌باشد، و شما می‌توانید به سادگی بررسی کنید که (۳۳۰,۳۱/۷۳۲,۲۳) برابر با مقدار نمایش داده شده نیست، ۴۵,۹۶٪؛ به جای آن می‌بایست ۴۵,۱۱ ٪ باشد.

SalesKey	SalesAmount	TotalCost	GrossMargin	Average of GrossMarginPct
20070104611301-0002	$72.19	$38.74	$33.45	46.34%
20070104611301-0003	$23.75	$11.50	$12.25	51.58%
20070104611320-0006	$216.57	$116.22	$100.35	46.34%
20070104611320-0007	$23.75	$11.50	$12.25	51.58%
20070104611506-0002	$72.19	$38.74	$33.45	46.34%
20070104611506-0003	$23.75	$11.50	$12.25	51.58%
20070104611914-0002	$64.59	$38.74	$25.85	40.02%
20070104611914-0003	$21.25	$11.50	$9.75	45.88%
20070104611952-0004	$64.59	$38.74	$25.85	40.02%
20070104611952-0005	$21.25	$11.50	$9.75	45.88%
20070104611998-0002	$64.59	$38.74	$25.85	40.02%
20070104611998-0003	$63.75	$34.50	$29.25	45.88%
Total	**$732.23**	**$401.92**	**$330.31**	**45.96%**

شکل ۲-۴ تغییر روش تجمیع به روش میانگین نتیجه صحیح را ارائه نمی‌دهد.

پیاده‌سازی صحیح GrossMarginPct از طریق یک یک معیار می‌باشد:

```
GrossMarginPct := SUM ( Sales[GrossMargin] ) / SUM (Sales[SalesAmount] )
```

[۱] - *Grand Total*

همان‌طور که ما قبلاً هم عنوان کردیم، نتیجه صحیح نمی‌تواند با یک ستون محاسبه‌شده حاصل شود. اگر شما نیاز به کار بر روی مقادیر تجمیع‌شـــده به جای کار بر مبنای ردیف‌به‌ردیف دارید، می‌بایســـت معیارها را ایجاد کنید. شـــاید تا کنون متوجه شده باشید که ما از علامت := به جای علامت مساوی (=) برای تعریف معیارها استفاده کرده‌ایم. این استانداردی است که ما در سراسر این کتاب از آن استفاده کرده‌ایم تا تشخیص تمایز بین معیارها و ستون‌های محاسبه‌شده در کد ساده‌تر باشد.

بعد از تعریف GrossMarginPct به صورت یک معیار، نتیجه صحیح حاصل می شود، همان‌طور که می‌توانید در شکل ۲-۵ مشاهده کنید.

SalesKey	SalesAmount	TotalCost	GrossMargin	GrossMarginPct
20070104611301-0002	$72.19	$38.74	$33.45	46.34%
20070104611301-0003	$23.75	$11.50	$12.25	51.58%
20070104611320-0006	$216.57	$116.22	$100.35	46.34%
20070104611320-0007	$23.75	$11.50	$12.25	51.58%
20070104611506-0002	$72.19	$38.74	$33.45	46.34%
20070104611506-0003	$23.75	$11.50	$12.25	51.58%
20070104611914-0002	$64.59	$38.74	$25.85	40.02%
20070104611914-0003	$21.25	$11.50	$9.75	45.88%
20070104611952-0004	$64.59	$38.74	$25.85	40.02%
20070104611952-0005	$21.25	$11.50	$9.75	45.88%
20070104611998-0002	$64.59	$38.74	$25.85	40.02%
20070104611998-0003	$63.75	$34.50	$29.25	45.88%
Total	**$732.23**	**$401.92**	**$330.31**	**45.11%**

شکل ۲-۵ *GrossMarginPct* که به‌صورت یک معیار تعریف شده است مجموع کل صحیح را نشان می‌دهد

معیارها و ستون‌های محاسبه‌شده هر دو از عبارات دکس استفاده می‌کنند؛ تفاوت در زمینه ارزیابی می‌باشد. یک معیار در زمینه مربوط به یک ابزارمصـــورســـازی[۱] یا در زمینه مربوط به کوئری دکس ارزیابی می‌شـــود. در صـــورتی که، یک ســـتون محاســـبه‌شـــده در ســـطح ردیف جدولی که به آن تعلق دارد محاســـبه می‌شـــود. زمینه ابزارمصـــورســـازی (بعداً در این کتاب، یاد خواهید گرفت که این یک زمینه فیلتری می‌باشـــد) وابســـته با انتخاب‌های کاربر در گزارش یا در قالب کوئری دکس می‌باشـــد. بنابراین، زمانی که از SUM(Sales[SalesAmount]) در یک معیار ا ستفاده می‌کنید، منظور ما جمع تمام ردیف‌هایی ا ست که تحت یک ابزارمصـــورســـازی تجمیع شـــده‌اند. با این حال، زمانی که ما از Sales[SalesAmount] در یک ســـتون محاســـبه‌شـــده استفاده می‌کنیم، منظور ما مقدار ستون SalesAmount در ردیف فعلی می‌باشد.

یک معیار نیاز دارد تا در یک جدول تعریف شـــود. این یکی از الزامات زبان دکس می‌باشـــد. با این وجود، این معیار به واقع به این جدول تعلق ندارد. در حقیقت، ما می‌توانیم یک معیار را از جدولی به جدول دیگر بدون از دســـت دادن عملکرد آن انتقال دهیم.

تفاوت‌های بین ستون‌های محاسبه‌شده و معیارها

اگرچه آن‌ها شبیه به هم به نظر می‌رسند، تفاوت بزرگی بین ستون‌های محاسبه‌شده و معیارها وجود دارد. مقدار یک ستون محاسبه‌شده در طول به‌روزرسانی داده‌ها محاسبه می‌شود، و از ردیف فعلی به عنوان زمینه استفاده می‌کند. نتیجه وابسته به فعالیت کاربر در گزارش نمی‌با شد. یک معیار بر روی تجمیع مربوط به داده‌های تعریف شده به و سیله زمینه فعلی عمل می‌کند. برای مثال، در یک ماتریس یا در یک جدول محوری[۲]، جداول مرجع بر اساس مختصات سلول‌ها فیلتر می‌شوند، و داده‌ها با استفاده از این فیلترها تجمیع شده و محاسبه می‌شوند. به عبارت دیگر، یک معیار همیشه بر روی مجموع داده‌ها تحت زمینه ارزیابی عمل می‌کند. زمینه ارزیابی در فصل ۴ «شناخت زمینه‌های ارزیابی» بیشتر توضیح داده خواهد شد.

---

۱ - *Visual*

۲ - *Pivot table*

انتخاب بین ستون‌های محاسبه‌شده و معیارها

حال که شما تفاوت بین ستون‌های محاسبه شده و معیارها را مشاهده کردید، این بحث که چه موقع از یکی نسبت به دیگری استفاده شود بسیار سودمند می‌باشد. برخی مواقع هر دو گزینه می‌توانند انتخاب شوند، اما در اغلب موارد، الزامات محاسبات نوع انتخاب را تعیین می‌کند.

به عنوان یک توسعه‌دهنده شما می‌بایست یک ستون محاسبه شده را هر زمان که خواهان انجام موارد زیر هستید تعریف کنید:

- نتایج محاسبه شده را در یک برشگر[۱] قرار دهید، یا نتایج را در ردیف‌ها یا ستون‌های یک ماتریس یا جدول محوری مشاهده کنید (برخلاف ناحیه مقادیر[۲])، یا از ستون محاسبه شده به عنوان یک شرط فیلتر در یک کوئری دکس استفاده کنید.
- می‌خواهید عبارتی تعریف کنید که به طور کامل به ردیف فعلی محدود شود. برای مثال Price * Quantity نمی‌تواند به شکل میانگین یا جمع این دو ستون کار کند.
- می‌خواهید دسته‌بندی متن یا اعداد انجام دهید. برای مثال، دامنه‌ای از مقادیر برای یک معیار، یک دامنه سنی برای مشتریان، مانند ۲۵-۱۸، ۱۸-۰، و غیره. این دسته‌ها اغلب به عنوان فیلتر یا برای برش دادن و تکه کردن مقادیر استفاده می‌شوند.

به هر حال، فرد هر زمانی که خواهان نمایش مقادیر حاصل از محاسبه می‌باشد باید یک معیار را تعریف کند که انتخاب‌های کاربر را منعکس می‌کند، و این مقادیر نیاز به ارائه به‌صورت تجمیعی در یک گزارش دارند، برای مثال:

- محاسبه درصد سود یک انتخاب مربوط به گزارش.
- محاسبه نسبت یک محصول در مقایسه با تمام محصولات اما با حفظ هر دو فیلتر سال و منطقه.

ما می‌توانیم محاسبات زیادی را با هر دوی ستون‌های محاسبه‌شده و معیارها بیان کنیم، اگرچه در مورد هر یک از آن‌ها نیاز به عبارت‌های متفاوت دکس داریم. برای مثال، فرد می‌تواند GrossMargin را به‌صورت ستون محاسبه شده به شکل زیر تعریف کند:

```
Sales[GrossMargin] = Sales[SalesAmount] - Sales[TotalProductCost]
```

با وجود این، GrossMargin می‌تواند به‌صورت معیار نیز به شکل زیر تعریف شود:

```
GrossMargin := SUM ( Sales[SalesAmount] ) - SUM ( Sales[TotalProductCost] )
```

ما پیشنهاد می‌کنیم که در این مورد شما از معیار استفاده کنید زیرا در زمانی که کوئری محاسبه می شود حافظه و فضای هارد دیسک را مصرف نمی‌کند. به عنوان یک قانون، هر زمانی که شما می‌توانید یک محاسبه را به هر دو روش بیان کنید، معیارها مسیر ارجح‌تری برای ادامه کار می‌باشند. شما می‌بایست استفاده از ستون‌های محاسبه‌شده را به تعداد معدودی که به شدت مورد نیاز می‌باشند محدود کنید. کاربرانی با تجربه کار با *Excel* به طور عادی ستون‌های محاسبه‌شده را به معیارها ترجیح می‌دهند چرا که ستون‌های محاسبه‌شده خیلی شبیه به شیوه انجام محاسبات در *Excel* می‌باشند. با وجود این، بهترین شیوه برای محاسبه یک مقدار در دکس از طریق یک معیار می‌باشد.

استفاده از معیارها در ستون‌های محاسبه‌شده

واضح است که یک معیار می‌تواند به یک یا چند ستون محاسبه‌شده اشاره داشته باشد. با شهود کمتری، شرایط معکوس هم

---

[۱] - *Slicer*

[۲] - *Values*

درست می‌باشد. یک ستون محاسبه‌شده می‌تواند به یک معیار اشاره کند. به این ترتیب، ستون محاسبه‌شده محاسبه یک معیار برای زمینه تعریف شده به وسیله ردیف جاری را به پیش می‌برد. این عملیات نتیجه یک معیار را به یک ستون تبدیل و تثبیت می‌کند، ستونی که تحت تاثیر اقدامات کاربر قرار نمی‌گیرد. به وضوح، تنها عملیات خاصی می‌تواند نتایج با معنایی ایجاد کند چرا که یک معیار به طور عادی محاسباتی را انجام می‌دهد که به شدت وابسته به انتخاب کاربر در ابزارمصورسازی می‌باشد. به علاوه، هر زمانی که شما، به عنوان توسعه‌دهنده، از معیارها در یک ستون محاسبه شده استفاده می‌کنید، متکی به یک ویژگی هستید که *انتقال زمینه‌ای*[1] نامیده می شود، که یک تکنیک پیشرفته محاسباتی در دکس می‌باشد. قبل از اینکه شما از یک معیار در یک ستون محاسبه شده استفاده کنید، ما به جد به شما توصیه می‌کنیم که فصل ۴ را مطالعه و درک کنید، که با جزئیات زمینه‌های ارزیابی و انتقالات زمینه‌ای را توضیح می‌دهد.

معرفی متغیرها

هنگام نوشتن یک عبارت دکس، فرد می‌تواند از تکرار همان عبارت اجتناب کند و به شکل چشم‌گیری خوانایی کد را با استفاده از متغیرها[2] بهبود دهد. برای مثال، به عبارت زیر نگاه کنید:

```
VAR TotalSales = SUM ( Sales[SalesAmount] )
VAR TotalCosts = SUM ( Sales[TotalProductCost] )
VAR GrossMargin = TotalSales - TotalCosts
RETURN
    GrossMargin / TotalSales
```

متغیرها با کلمه کلیدی VAR تعریف می‌شوند. بعد از آنکه شما متغیری را تعریف کردید، نیاز به یک بخش RETURN دارید که مقدار نتیجه این عبارت را تعریف می‌کند. فرد می‌تواند متغیرهای زیادی را تعریف کند، و این متغیرها محلی برای عبارتی هستند که درون آن تعریف شده‌اند.

یک متغیر تعریف‌شده در یک عبارت نمی‌تواند بیرون از عبارت خودش استفاده شود. چیزی به عنوان تعریف متغیر جهان شمول وجود ندارد. این بدین معنا می‌باشد که شما نمی‌توانید متغیرهایی را تعریف کنید که در کل کدهای دکس مدل استفاده شوند.

متغیرها با استفاده از یک ارزیابی بطئی[3] محاسبه می شوند. این بدین معنا می‌باشد که اگر فردی یک متغیر را تعریف کند که، به هر دلیلی، در کد موردنظر استفاده نشده باشد، خود این متغیر هرگز ارزیابی نمی‌شود. اگر آن نیاز به محاسبه شدن دارد، این محاسبه فقط یک بار رخ می‌دهد. استفاده‌های بعدی از این متغیر مقدار محاسبه شده قبلی را خواهد خواند. بنابراین، متغیرها زمانی که در یک عبارت پیچیده چند مرتبه استفاده می‌شوند به عنوان یک تکنیک بهینه‌سازی نیز سودمند می‌باشند.

متغیرها ابزار مهمی در دکس می‌باشند. همان‌طور که در فصل ۴ شما یاد خواهید گرفت، متغیرها بسیار سودمند می‌باشند زیرا به جای زمینه‌ای که متغیر در آن مورد استفاده قرار می‌گیرد، آن‌ها از زمینه ارزیابی تعریف استفاده می‌کنند. در فصل ۶، «متغیرها» ما به طور کامل متغیرها و چگونگی استفاده از آن‌ها را پوشش خواهیم داد. همچنین ما از متغیرها به طور گسترده در سراسر این کتاب استفاده خواهیم کرد.

---

[1] - *Context Transition*

[2] - *Variables*

[3] - *Lazy Evaluation*

مدیریت خطاها در عبارت‌های دکس

اکنون که شما برخی از مبانی مربوط به ترکیب را مشاهده کردید، زمان آن فرا رسیده که یاد بگیرید چگونه به شکل کامل محاسبات نامعتبر را مدیریت کنید. یک عبارت دکس ممکن است حاوی محاسبات نامعتبری باشد زیرا داده‌هایی که این عبارت به آن‌ها رجوع می‌کند برای این فرمول معتبر نمی‌باشند. برای مثال، فرمول مدنظر ممکن است حاوی یک تقسیم بر صفر باشد یا یک رجوع به مقدار ستونی باشد که عددی نیست در حالی که در یک عملیات حسابی مثل ضرب مورد استفاده قرار گرفته است. خوب است یاد بگیرید که چگونه این خطاها به طور پیش‌فرض مدیریت می شوند و چگونه می‌توان جلوی این شرایط را به منظور رسیدگی خاص گرفت.

با این حال، پیش از بحث درباره چگونگی پوشش خطاها، ما انواع متفاوت خطاهایی را توضیح می‌دهیم که ممکن است در طول یک ارزیابی فرمول دکس ظاهر شوند. آن‌ها عبارت اند از:

- خطاهای تبدیل[۱]
- خطاهای عملیات حسابی[۲]
- مقادیر خالی یا گم شده[۳]

خطاهای تبدیل

اولین نوع خطا از نوع خطای تبدیل می‌باشد. همان‌طورکه ما پیش‌تر در این فصل نشان داده‌ایم، دکس به طور خودکار مقادیر رشته‌ای و عددی را هر زمانی که عملگر به آن نیاز داشته باشد به یکدیگر تبدیل می‌کند. تمام این مثال‌ها عبارت‌های معتبر دکس می‌باشند:

```
"10" + 32 = 42
"10" & 32 = "1032"
10 & 32 = "1032"
DATE (2010,3,25) = 3/25/2010
DATE (2010,3,25) + 14 = 4/8/2010
DATE (2010,3,25) & 14 = "3/25/201014"
```

این فرمول‌ها همیشه صحیح می‌باشند چرا که آن‌ها با مقادیر ثابت کار می‌کنند. با این حال، اگر ستون VatCode رشته‌ای باشد، در مورد فرمول زیر چطور؟

```
Sales[VatCode] + 100
```

از آنجا که اولین عملوند[۴] این جمع یک ستون است که از نوع داده‌های متنی می‌باشد، شما به عنوان توسعه‌دهنده می‌بایست مطمئن شوید که دکس می‌تواند تمام مقادیر درون آن ستون را به اعداد تبدل کند. اگر دکس در تبدیل برخی از محتواها به مورد مناسبی که عملگر[۵] نیاز دارد شکست بخورد، یک خطای تبدیل رخ خواهد داد. موارد زیر برخی از موقعیت‌های معمول این نوع خطا می‌باشند:

---

۱ - *Conversion Errors*

۲ - *Arithmetic Operations Errors*

۳ - *Empty Or Missing Values*

۴ - *Operand*

۵ - *Operator*

```
"1 + 1" + 0 = Cannot convert value '1 + 1' of type Text to type Number
DATEVALUE ("25/14/2010") = Type mismatch
```

اگر شـــما می‌خواهید از این نوع خطاها اجتناب کنید، افزودن منطق تشـــخیص خطا در عبارت‌های دکس به منظور پیشگیری از موقعیت‌های خطا و بازگرداندن نتیجه‌ای که با معنا باشد مهم می‌باشد. فرد می‌تواند از طریق تشخیص خطا بعد از آنکه رخ داد یا با بررسی این عملوندها برای وضعیت خطا به‌صورت پیشگیرانه، نتیجه یکسانی به‌دست آورد. با این وجود، بررسی پیشگیرانه موقعیت خطا بهتر از این است که اجازه داده شود خطا رخ دهد و بعد شناسایی گردد.

خطاهای عملیات حسابی

دومین دســته خطاها از نوع خطاهای مربوط به عملیات حسـابی می‌باشـند، مانند تقسـیم بر صـفر یا جذر یک عدد منفی. این خطاها وابســـته به تبدیل نمی‌باشـــند: دکس آن‌ها را هر زمانی که ما تلاش کنیم یک تابع یا اســـتفاده از یک عملگر با مقادیر نامعتبر را فراخوانی کنیم، نشان می‌دهد.

تقسیم بر صفر نیاز به بررسی خاصی دارد چرا که رفتار آن خیلی واضح نمی‌باشد (شاید به استثناء ریاضی‌دانان). زمانی که فردی یک عدد را بر صـــفر تقســـیم می‌کند، دکس به طور معمول مقدار خاص *بی‌نهایت*[۱] را باز می‌گرداند. در موارد خاص تقسیم صفر بر صفر یا *بی‌نهایت* تقسیم بر *بی‌نهایت*، دکس مقدار خاص NaN[۲] (یک عدد نیست) را باز می‌گرداند.

به خاطر اینکه این رفتار غیر عادی می‌باشد، در جدول ۲-۳ خلاصه شده است.

جدول ۲-۳ مقادیر خاص به دست آمده برای تقسیم بر صفر

عبارت	نتیجه
10 / 0	Infinity
7 / 0	Infinity
0 / 0	NaN
(10 / 0) / (7 / 0)	NaN

مهم است توجه شود که Infinity و NaN خطا نمی‌باشند بلکه مقادیر خاصی در دکس می‌باشند. در حقیقت، اگر فرد یک عدد را بر *بی‌نهایت* تقسیم کند عبارت نمی‌تواند یک خطا ایجاد کند. در عوض، صفر را بر می‌گرداند:

```
9954 / ( 7 / 0 ) = 0
```

به غیر از این وضعیت خاص، دکس می‌تواند هنگام فراخوانی یک تابع همراه با یک پارامتر اشتباه، همانند جذر یک عدد منفی خطاهای حسابی را برگرداند:

```
SQRT ( -1 ) = An argument of function 'SQRT' has the wrong data type or the result is too
large or too small
```

اگر دکس خطاهایی شبیه به این را تشخیص دهد، ادامه محاسبه عبارت را متوقف و یک خطا ارائه می‌کند. فرد می‌تواند از تابع ISERROR به منظور برر سی اینکه آیا عبارت منجر به خطا می شود ا ستفاده کند. ما بعداً این سناریو را در این ف صل

---

۱ - *Infinity*

۲ - *Not A Number*

نشان می‌دهیم.

به یاد داشته باشید که مقادیر خاص شبیه به NaN در رابط کاربری چند ابزار همانند *Power BI* به شکل مقادیر عادی نمایش داده می شود. با این حال، زمانی که به و سیله دیگر ابزارهای کلاینت[۱] نمایش داده می شوند می‌توان با آن‌ها به صورت خطاهایی برخورد کرد همانند جدول محوری *Excel*. در نهایت، این مقادیر خاص به عنوان خطا توسط توابع تشخیص خطا شناسایی می‌شوند.

مقادیر خالی یا گمشده

سومین د سته‌ای که ما برر سی می‌کنیم و ضعیت خطای خا صی نمی‌با شد، به غیر از ح ضور مقادیر خالی. مقادیر خالی ممکن است زمانی که با دیگر عناصر در یک محاسبه ترکیب می‌شوند منجر به نتایج غیرمنتظره یا خطاهای محاسباتی شوند.

دکس مقادیر گم شده، مقادیر خالی، یا سلول‌های خالی را به شیوه‌ای م شابه، با ا ستفاده از مقدار BLANK پو شش می‌دهد. BLANK یک مقدار واقعی نمی‌با شد اما در عوض شیوه‌ای خاص برای م شخص کردن این شرایط به شمار می‌رود. ما می‌توانیم مقدار BLANK را در یک عبارت دکس با فراخوانی تابع BLANK که متفاوت از یک ر شته خالی می‌با شد به د ست آوریم. برای مثال، عبارت زیر همیشه یک مقدار خالی را باز می‌گرداند، موردی که می‌تواند هم به‌صورت یک رشته خالی و یا به‌صورت یک «*(blank)*» در ابزارهای کلاینت نمایش داده شود:

```
= BLANK ()
```

این عبارت به خودی‌خود سودمند نمی‌باشد، اما خود تابع BLANK هر زمانی که نیاز به بازگرداندن یک مقدار خالی با شد مفید واقع می شود. برای مثال، شاید شما خواهان نمایش یک سلول خالی به جای صفر با شید. عبارت زیر تخفیف کلی مربوط به یک تراکنش فروش را محاسبه می‌کند، اگر میزان تخفیف برابر صفر باشد سلول خالی (*blank*) می‌ماند:

```
=IF (
    Sales[DiscountPerc] = 0,                -- بررسی شود که آیا تخفیف وجود دارد
    BLANK (),                               -- در صورت عدم وجود تخفیف، blank را برگردانید
    Sales[DiscountPerc] * Sales[Amount]     -- در غیر این صورت تخفیف را محاسبه کنید
)
```

BLANK، به خودی‌خود، یک خطا نمی‌باشد؛ آن صرفاً یک مقدار خالی است. بنابراین، عبارتی که شامل یک BLANK می‌باشد ممکن است بر اساس آنچه که مورد نیاز محاسبه است یک مقدار یا یک خالی (*blank*) را بازگرداند. برای مثال، عبارت زیر هر زمان که *Sales[Amount]* برابر BLANK باشد BLANK را باز می‌گرداند.

```
= 10 * Sales[Amount]
```

به عبارت دیگر، هر زمانی که یک یا هر دو شرط BLANK باشد نتیجه یک عملیات حسابی BLANK است. زمانی که برر سی یک مقدار خالی ضروری می‌با شد این مو ضوع چالشی را ایجاد می‌کند. به خاطر تبدیلات ضمنی، تشخیص اینکه یک عبارت، صفر (یا ر شته خالی) ا ست یا BLANK می‌با شد با به کارگیری یک عملگر یکسان ممکن نمی‌با شد. در واقع، شرایط منطقی زیر همیشه صادق است:

```
BLANK () = 0        -- همیشه TRUE را باز می گرداند
BLANK () = ""       -- همیشه TRUE را باز می گرداند
```

بنابراین، اگر ستون‌های *Sales[DiscountPerc]* یا *Sales[Clerk]* خالی باشند، شرط‌های زیر مقدار *TRUE* را باز می‌گردانند حتی اگر این بررسی به ترتیب در مورد صفر و یک رشته خالی باشد:

---

[۱] - *Client*

```
Sales[DiscountPerc] = 0   -- اگر DiscountPerc خالی یا ۰ باشد ، TRUE را برمی‌گرداند
Sales[Clerk] = ""         اگر Clerk برابر BLANK یا "" باشد، TRUE را برمی‌گرداند
```

در چنین مواردی، فرد می‌تواند از تابع ISBLANK برای بررسی اینکه آیا یک مقدار BLANK می‌باشد یا خیر استفاده کند:

```
ISBLANK ( Sales[DiscountPerc] )   -- فقط در صورتی که DiscountPerc برابر BLANK باشد، TRUE را برمی‌گرداند
ISBLANK ( Sales[Clerk] )          -- فقط در صورتی که Clerk برابر BLANK باشد، TRUE را برمی‌گرداند
```

این تسری BLANK در یک عبارت دکس در چند عملیات حسابی و منطقی دیگر نیز اتفاق می‌افتد، همان‌طور که در مثال زیر نشان داده شده است:

```
BLANK () + BLANK () = BLANK ()
10 * BLANK () = BLANK ()
BLANK () / 3 = BLANK ()
BLANK () / BLANK () = BLANK ()
```

به هر حال، تسری BLANK در خروجی یک عبارت برای همه فرمول‌ها اتفاق نمی‌افتد. برخی محاسبات BLANK را تسری نمی‌دهند. در عوض، مقداری را بر اساس دیگر شرط‌های فرمول باز می‌گردانند. نمونه‌هایی از این موارد عبارت‌اند از عمل جمع، تفریق، تقسیم بر BLANK، و یک عملیات منطقی که شامل BLANK می‌باشد. عبارت‌های زیر برخی از این وضعیت‌ها را همراه با نتایج آن‌ها نشان می‌دهند:

```
BLANK () - 10 = -10
18 + BLANK () = 18
4 / BLANK () = Infinity
0 / BLANK () = NaN
BLANK () || BLANK () = FALSE
BLANK () && BLANK () = FALSE
( BLANK () = BLANK () ) = TRUE
( BLANK () = TRUE ) = FALSE
( BLANK () = FALSE ) = TRUE
( BLANK () = 0 ) = TRUE
( BLANK () = "" ) = TRUE
ISBLANK ( BLANK() ) = TRUE
FALSE || BLANK () = FALSE
FALSE && BLANK () = FALSE
TRUE || BLANK () = TRUE
TRUE && BLANK () = FALSE
```

مقادیر خالی در اکسل و SQL

اکسل شیوه متفاوتی برای پوشش مقادیر خالی دارد. در اکسل، تمام مقادیر خالی زمانی که در یک جمع یا ضرب مورد استفاده قرار می‌گیرند به صورت ۰ در نظر گرفته می شوند، اما اگر آن‌ها بخشی از یک تقسیم یا یک عبارت منطقی باشند، ممکن است خطایی را باز گرداند.

در *SQL*، مقادیر *null* در یک عبارت به صورت متفاوت از آنچه که در مورد BLANK در دکس رخ می‌دهد تسری می‌یابند. همان‌طور که می‌توانید در مثال‌های قبل مشاهده کنید، حضور یک BLANK در یک عبارت دکس همیشه منجر به خروجی BLANK نمی شود، در حالی که حضور NULL در *SQL* اغلب به صورت NULL برای کل عبارت ارزیابی می شود. این تفاوت مربوط به هر زمانی است که شما از *DirectQuery* بر روی یک پایگاه‌داده رابطه‌ای استفاده می‌کنید چرا که

برخی محاسبات در *SQL* و برخی دیگر در دکس اجرا می‌شوند. تفاوت معنایی BLANK در این دو موتور ممکن است منجر به رفتارهای غیرمنتظره شود.

شناخت رفتار مقادیر خالی یا گمشده در یک عبارت دکس و استفاده از BLANK به منظور بازگرداندن یک سلول خالی در یک محاسبه مهارت مهمی برای کنترل نتایج یک عبارت دکس می‌باشد. همان‌گونه که ما در بخش بعدی نشان می‌دهیم، فرد اغلب هنگامی که مقادیر اشتباه یا دیگر خطاها را تشخیص می‌دهد می‌تواند از BLANK به عنوان نتیجه استفاده کنید.

ردیابی خطاها

اکنون که انواع مختلف خطاهایی که می‌توانند رخ دهند را با جزئیات بیان کردیم، هنوز نیاز به نشان دادن تکنیک‌هایی به شما برای ردیابی و اصلاح این خطاها یا حداقل نمایش یک پیغام خطا داریم که حاوی اطلاعات با معنا باشد. وجود خطا در یک عبارت دکس بیشتر اوقات وابسته به مقدار موجود در ستون‌هایی است که در خود این عبارت استفاده شده‌اند. بنابراین، شاید فرد خواهان کنترل وجود این شرایط خطا و باز گرداندن یک پیغام خطا باشد. تکنیک استاندارد این است که بررسی شود آیا یک عبارت خطایی را باز می‌گرداند و، اگر این چنین شد، این خطا را با یک پیغام یا یک مقدار پیش‌فرض جایگزین کند. برای این کار در دکس چند تابع وجود دارد.

اولین آن‌ها تابع IFERROER است، که شبیه به تابع IF می‌باشد، اما به جای ارزیابی یک وضعیت منطقی بولین، بررسی می‌کند که آیا یک عبارت خطایی را باز می‌گرداند. دو کاربرد رایج تابع IFERROER به‌صورت زیر می‌باشد:

```
= IFERROR ( Sales[Quantity] * Sales[Price], BLANK () )
= IFERROR ( SQRT ( Test[Omega] ), BLANK () )
```

در اولین عبارت، اگر Sales[Quantity] یا Sales[Price] رشته‌ای باشند که نتوان آن‌ها را به عدد تبدیل کرد، عبارت بازگردانده شده یک مقداری خالی می‌باشد؛ در غیر این صورت حاصل‌ضرب Quantity و Price بازگردانده می‌شوند.

در دومین عبارت، هر زمانی که ستون Test[Omega] حاوی یک عدد منفی باشد، نتیجه یک سلول خالی است.

این شیوه استفاده از تابع IFERROR مرتبط با یک الگوی کلی‌تر می‌باشد که به استفاده از ISERROR و IF نیاز دارد:

```
= IF (
    ISERROR ( Sales[Quantity] * Sales[Price] ),
    BLANK (),
    Sales[Quantity] * Sales[Price]
)

= IF (
    ISERROR ( SQRT ( Test[Omega] ) ),
    BLANK (),
    SQRT ( Test[Omega] )
)
```

در این موارد، IFERROR گزینه بهتری می‌باشد. هر زمان که نتیجه همان عبارت بررسی‌شده برای خطا باشد؛ فرد می‌تواند از IFERROR استفاده کند؛ نیازی نسیت این عبارت را در دو محل تکرار کنید، همچنین این کد خواناتر و امن‌تر می‌باشد. با این حال، توسعه‌دهنده می‌بایست از IF استفاده کند اگر آن‌ها قصد بازگرداندن نتیجه یک عبارت متفاوت را دارند.

علاوه براین، فرد می‌تواند از طریق بررسی پارامترها قبل از استفاده از آن‌ها از ایجاد خطاهای کلی اجتناب کند. برای

مثال، فرد می‌تواند معلوم کند آیا آرگومان SQRT مثبت می‌باشد، در غیر این صورت BLANK را برای اعداد منفی بازگرداند:

```
= IF (
    Test[Omega] >= 0,
    SQRT ( Test[Omega] ),
    BLANK ()
)
```

با توجه به این‌که سومین آرگومان یک دستور IF به طور پیش‌فرض BLANK می‌باشد، فرد می‌تواند همان عبارت را به شکل مختصر به‌صورت زیر بنویسد:

```
= IF (
    Test[Omega] >= 0,
    SQRT ( Test[Omega] )
)
```

یک سناریو رایج بررسی مقادیر خالی می‌باشد. ISBLANK مقادیر خالی را پیدا می‌کند، اگر آرگومان آن *BLANK* باشد *TRUE* را باز می‌گرداند. این قابلیت به ویژه زمانی مهم است که در دسترس نبودن یک مقدار به معنای ۰ بودن آن نباشد. مثال زیر هزینه ارسال کالا را با استفاده از یک هزینه ارسال پیش‌فرض برای محصول اگر تراکنش خود وزنی را مشخص نکرده باشد، محاسبه می‌کند:

```
= IF (
   ISBLANK ( Sales[Weight] ),              -- اگر وزن موجود نباشد
   Sales[DefaultShippingCost],             -- هزینه پیش فرض را برگردانید
   Sales[Weight] * Sales[ShippingPrice]    -- در غیر این صورت وزن را در هزینه حمل ضرب کنید
)
```

اگر ما به سادگی وزن محصول و قیمت ارسال کالا را ضرب کنیم، به خاطر تسری *BLANK* در ضرب‌ها یک هزینه خالی برای تمام تراکنش‌های فروش که داده مربوط به وزن را ندارند خواهیم داشت.

زمان استفاده از متغیرها، خطاها باید در زمان تعریف متغیرها بررسی شوند نه جایی که ما ازآن‌ها استفاده می‌کنیم. در حقیقت، اولین فرمول در کد زیر صفر را باز می‌گرداند، فرمول دوم همیشه خطایی را نشان می‌دهد، و فرمول آخر بر اساس نسخه محصولی که از دکس استفاده می‌کند نتایج متفاوتی را ایجاد می‌کند (آخرین نسخه نیز خطا می‌دهد):

```
IFERROR ( SQRT ( -1 ), 0 )                  -- این ۰ را برمی گرداند

VAR WrongValue = SQRT ( -1 )                -- خطا در اینجا رخ می دهد، بنابراین نتیجه این است
RETURN                                      -- همیشه یک خطا
    IFERROR ( WrongValue, 0 )               -- این خط هرگز اجرا نمی شود

IFERROR (                                   -- نتایج متفاوت بسته به نسخه نرم افزار
    VAR WrongValue = SQRT ( -1 )            -- IFERROR در نسخه های ۲۰۱۷ خطا می دهد
    RETURN                                  -- IFERROR در نسخه های تا سال ۲۰۱۶ مقدار ۰ را برمی گرداند
        WrongValue,
    0
)
```

خطا زمانی رخ می‌دهد که WrongValue ارزیابی می‌شود. بنابراین، این موتور هرگز تابع IFERROR را در دومین مثال اجرا نخواهد کرد، در حالی که خروجی سومین مثال وابسته به نسخه‌های محصول می‌باشد. اگر شما نیاز به بررسی خطاها دارید، زمانی که از متغیرها استفاده می‌کنید اقدامات احتیاطی بیشتری را اتخاذ کنید.

از کاربرد توابع پوشش‌دهنده خطا اجتناب کنید

اگرچه در ادامه کتاب ما بهینه‌سازی‌ها را توضیح خواهیم داد، اما باید توجه داشته باشید که توابع پوشش‌دهنده خطا ممکن است مشکلات عملکردی شدیدی در کد شما ایجاد کنند. این‌طور نیست که آن‌ها به خودی‌خود کند باشند. مشکل این است که موتور دکس نمی‌تواند از مسیرهای بهینه‌شده در کدهایش در زمان بروز خطا استفاده کند. در اغلب موارد، بررسی عملوندها برای خطاهای احتمالی کارآمدتر از موتور پوشش‌دهنده خطا می‌باشد. برای مثال، به جای نوشتن این:

```
= IFERROR (
    SQRT ( Test[Omega] ),
    BLANK ()
  )
```

خیلی بهتر است این را بنویسید:

```
= IF (
    Test[Omega] >= 0,
    SQRT ( Test[Omega] ),
    BLANK ()
  )
```

دومین عبارت نیاز به مشخص کردن خطا ندارد و سریع‌تر از مورد قبلی می‌باشد. البته، این یک قانون کلی است. برای توضیح دقیق این موضوع، فصل ۱۹ «بهینه‌سازی دکس» را مشاهده کنید.

دلیل دیگر برای اجتناب از IFERROR این است که نمی‌تواند خطاهایی را که در سطح عمیق‌تر عملیات اتفاق می‌افتند رهگیری کند. برای مثال، کد زیر هرگونه خطایی را در تبدیل ستون Table[Amount] با در نظر گرفتن یک مقدار خالی در صورتی که Amount حاوی عدد نباشد، رهگیری می‌کند. همان‌طور که قبلاً بحث شد، این اجرا هزینه‌بر است زیرا برای هر ردیف در Table ارزیابی می‌شود.

```
SUMX (
    Table,
    IFERROR ( VALUE ( Table[Amount] ), BLANK () )
)
```

توجه داشته باشید که به دلیل بهینه‌سازی در موتور دکس، کد زیر همان خطاهایی که در مثال قبل رهگیری شده بودند را رهگیری نمی‌کند. اگر Table[Amount] حاوی یک رشته باشد که فقط در یک ردیف عدد نباشد، کل عبارت خطایی ایجاد می‌کند که توسط IFERROR شناسایی نمی‌شود.

```
IFERROR (
    SUMX (
        Table,
        VALUE ( Table[Amount] )
    ),
    BLANK ()
)
```

ISERROR رفتاری مشابه با IFERROR دارد. مطمئن شوید که آن‌ها را با دقت و فقط برای رهگیری خطاهایی که مستقیماً توسط عبارت ارزیابی‌شده درون IFERROR/ISERROR و نه در محاسبات تودرتو ایجاد می‌شود، استفاده کنید.

ایجاد خطا

گاهی اوقات، یک خطا فقط یک خطا است و در صورت بروز خطا، فرمول نباید مقدار پیش‌فرض را برگرداند. در واقع، بازگرداندن یک مقدار پیش‌فرض منجر به ایجاد نتیجه واقعی می‌شود که نادرست است. برای مثال، یک جدول پیکربندی[۱] که حاوی داده‌های متناقض است، باید گزارشی نامعتبر به جای اعدادی که غیرقابل اعتماد هستند، تولید کند، اما هنوز ممکن است درست در نظر گرفته شود.

علاوه بر این، به‌جای یک خطای عمومی[۲]، ممکن است فرد بخواهد یک پیغام خطایی ایجاد کند که برای کاربران معنادارتر باشد. چنین پیغامی به کاربران کمک می‌کند تا مشکل را پیدا کنند.

سناریویی را در نظر بگیرید که برای تنظیم تقریبی سرعت صوت در یک محاسبه علمی پیچیده، نیاز به محاسبه جذر دمای مطلق اندازه‌گیری شده بر حسب کلوین دارد. بدیهی است که ما انتظار نداریم که دما یک عدد منفی باشد. اگر به دلیل مشکل در اندازه‌گیری این اتفاق بیفتد، باید یک خطا ایجاد و محاسبه را متوقف کنیم.

در این صورت، این کد خطرناک است زیرا مشکل را پنهان می‌کند:

```
= IFERROR (
    SQRT ( Test[Temperature] ),
    0
)
```

در عوض، برای محافظت از محاسبات، فرد باید فرمول بالا را به شکل زیر بنویسد:

```
= IF (
    Test[Temperature] >= 0,
    SQRT ( Test[Temperature] ),
    ERROR ( "The temperature cannot be a negative number. Calculation aborted." )
)
```

قالب‌بندی[۳] کد دکس

قبل از اینکه به توضیح زبان دکس ادامه دهیم، قصد داریم یک جنبه مهم از دکس را پوشش دهیم، یعنی قالب‌بندی کد دکس. دکس یک زبان تابعی است، به این معنی که هر چقدر هم که پیچیده باشد، یک عبارت دکس مانند فراخوانی یک تابع عمل می‌کند. پیچیدگی کد به پیچیدگی عباراتی تبدیل می‌شود که فرد به عنوان پارامتر برای بیرونی‌ترین تابع استفاده می‌کند.

به همین دلیل، دیدن عبارت‌هایی که ۱۰ خط یا بیشتر را شامل می شود، طبیعی است. دیدن عبارت ۲۰ خطی دکس امری رایج است، لاجرم با آن آشنا خواهید شد. با این وجود، از زمانی که فرمول‌ها شروع به افزایش طول و پیچیدگی می‌کنند، قالب‌بندی کد به گونه‌ای که برای شخص خوانا باشد بسیار مهم است.

هیچ استاندارد «رسمی» برای قالب‌بندی کد دکس وجود ندارد، با این حال معتقدیم که توصیف استانداردی که در کد خود استفاده می‌کنیم مهم است. احتمالاً استاندارد کاملی نیست، و ممکن است شما چیزی متفاوت را ترجیح دهید. ما هیچ مشکلی با آن نداریم: استاندارد بهینه خود را پیدا کنید و از آن استفاده کنید. تنها چیزی که باید به خاطر بسپارید این است: کد خود را قالب‌بندی کنید و هرگز همه چیز را در یک خط ننویسید. در غیر این صورت زودتر از آنچه انتظار دارید دچار

---

۱ - *Configuration*

۲ - *Generic Error*

۳ - *Formatting*

مشکل خواهید شد.

برای درک اینکه چرا قالب‌بندی مهم می‌باشد، به فرمولی نگاه کنید که یک محاسبه هوش زمانی را حساب می‌کند. این فرمول تا حدی پیچیده هنوز پیچیده‌ترین فرمولی نیست که شما خواهید نوشت. اگر به شیوه‌ای آن را قالب‌بندی نکنید، نحوه نمایش این عبارت به این شکل می‌باشد:

```
IF(CALCULATE(NOT ISEMPTY(Balances), ALLEXCEPT (Balances, BalanceDate)),SUMX (ALL(Balances
[Account]), CALCULATE(SUM (Balances[Balance]),LASTNONBLANK(DATESBETWEEN(BalanceDate[Date],
BLANK(),MAX(BalanceDate[Date])),CALCULATE(COUNTROWS(Balances))))),BLANK())
```

تلاش برای درک آنچه که این فرمول در شکل فعلی آن محاسبه می‌کند تقریباً غیرممکن است. هیچ سرنخی وجود ندارد که بیرونی‌ترین تابع کدام است و چگونه دکس پارامترهای مختلف را برای ایجاد جریان کامل اجرا ارزیابی می‌کند. نمونه‌های بسیار زیادی از فرمول‌هایی را دیده‌ایم که توسط دانشجویان به این شیوه نوشته شده‌اند کسانی که در مقطعی برای درک اینکه چرا فرمول نتایج نادرست برمی‌گرداند درخواست کمک می‌کنند. حدس بزنید چه اتفاقی می‌افتد؟ اولین کاری که ما انجام می‌دهیم این است که عبارت را قالب‌بندی کنیم؛ فقط بعد از آن است که ما شروع به کار روی آن می‌کنیم.

همان عبارت بالا که به شکل صحیحی قالب‌بندی شده است، به این صورت به نظر می‌رسد:

```
IF (
    CALCULATE (
        NOT ISEMPTY ( Balances ),
        ALLEXCEPT (
            Balances,
            BalanceDate
        )
    ),
    SUMX (
        ALL ( Balances[Account] ),
        CALCULATE (
            SUM ( Balances[Balance] ),
            LASTNONBLANK (
                DATESBETWEEN (
                    BalanceDate[Date],
                    BLANK (),
                    MAX ( BalanceDate[Date] )
                ),
                CALCULATE (
                    COUNTROWS ( Balances )
                )
            )
        )
    ),
    BLANK ()
)
```

کد همان است، اما این بار دیدن سه پارامتر IF بسیار ساده‌تر می‌باشد. مهم‌تر از همه، دنبال کردن بلوک‌هایی که به طور طبیعی از خطوط تورفته به وجود می‌آیند و نحوه تشکیل جریان کامل اجرا آسان‌تر است. خواندن کد هنوز سخت است، اما اکنون مشکل دکس است، نه قالب‌بندی ضعیف. یک ترکیب طولانی‌تر با استفاده از متغیرها می‌تواند به شما در خواندن کد کمک کند، اما حتی در این مورد هم، قالب‌بندی در ارائه شناخت صحیح از محدوده هر متغیر مهم است:

```
IF (
    CALCULATE (
        NOT ISEMPTY ( Balances ),
        ALLEXCEPT (
            Balances,
            BalanceDate
        )
    ),
    SUMX (
        ALL ( Balances[Account] ),
        VAR PreviousDates =
            DATESBETWEEN (
                BalanceDate[Date],
                BLANK (),
                MAX ( BalanceDate[Date] )
            )
        VAR LastDateWithBalance =
            LASTNONBLANK (
                PreviousDates,
                CALCULATE (
                    COUNTROWS ( Balances )
                )
            )
        RETURN
            CALCULATE (
                SUM ( Balances[Balance] ),
                LastDateWithBalance
            )
    ),
    BLANK ()
)
```

DAXFormatter.com

ما یک وب‌سایت اختصاصی برای قالب‌بندی کد دکس ایجاد کرده‌ایم. ما این سایت را برای خودمان ایجاد کردیم زیرا قالب‌بندی کد عملیات زمان‌بری ا ست و ما نمی‌خوا ستیم برای هر فرمولی که می‌نوی سیم وقت خود را صرف قالب‌بندی آن کنیم. پس از آن‌که این ابزار به درســـتی کار کرد، تصـــمیم گرفتیم آن را به دامنه عمومی ارائه کنیم تا کاربران بتوانند کد دکس خود را قالب‌بندی کنند (به هر حال، ما توانستیم قوانین قالب‌بندی خود را از این طریق ترویج کنیم).

می‌توانید این وب‌سایت را در آدرس *www.daxformatter.com* پیدا کنید. رابط کاربری ساده است: فقط کد دکس خود را کپی کنید، روی دکمه *FORMAT* کلیک کنید، و صـفحه بازخوانی می‌شـود و نسـخه‌ای با قالب زیبایی از کد شـما را نشـان می‌دهد، سپس می‌توانید آن را کپی و در پنجره اصلی جایگذاری کنید.

این مجموعه قوانینی است که ما برای قالب‌بندی دکس استفاده می‌کنیم

- همیشـــه نام توابع مانند IF، SUMX و CALCULATE را از هر عبارت دیگری با اســـتفاده از فاصـــله جدا کنید و همیشه آن‌ها را با حروف بزرگ بنویسید.
- تمام ارجاعات ستونی را به شکل TableName[ColumnName] بنوی سید، بدون فا صله بین نام جدول و کروشه آغازین. نام جدول همیشه ذکر می‌شود.
- تمام ارجاعات معیاری را به شکل [MeasureName] بدون نام جدول بنویسید.
- همیشه از یک فاصله بعد از کاما استفاده کنید و هرگز قبل از آن فاصله قرار ندهید.
- اگر فرمول در یک خط جا گرفت، قانون دیگری مورد نیاز نمی‌باشد.

- اگر فرمول در یک خط جا نگرفت، آنگاه:
  - نام تابع را همراه با پرانتز ابتدایی به تنهایی در یک خط قرار دهید
  - تمام پارامترها را در خطوط جداگانه، با چهار فاصله تورفتگی و همراه با کاما در انتهای عبارت به جز آخرین پارامتر، نگه دارید.
  - پرانتز پایانی را با فراخوانی تابع هم‌تراز کنید به گونه‌ای که خودش در یک خط قرار بگیرد

این‌ها قوانین کلی هستند که ما استفاده می‌کنیم. فهرست دقیق‌تری از این قوانین در آدرس *http://sql.bi/daxrules* در دسترس می‌باشد.

اگر راهی برای بیان فرمول‌ها یافتید که به بهترین شکل با شیوه خواندن شما مطابقت دارد، از آن استفاده کنید. هدف از قالب‌بندی آسان‌تر کردن فرمول برای خواندن است، بنابراین از تکنیکی استفاده کنید که به بهترین نحو برای شما کار کند. مهم‌ترین نکته‌ای که باید هنگام تعریف مجموعه شخصی قوانین قالب‌بندی به‌خاطر بسپارید این است که همیشه باید بتوانید خطاها را در اسرع وقت مشاهده کنید. اگر در کد قالب‌بندی‌نشده، که قبلاً نشان داده شد، دکس از مفقود شدن پرانتز انتهایی شکایت کند، تشخیص اینکه این خطا در کجا قرار دارد سخت می‌باشد. در فرمول قالب‌بندی شده، مشاهده اینکه چگونه هر پرانتز انتهایی با فراخوانی تابع آغازین مطابقت دارد، بسیار ساده‌تر است.

کمک در خصوص قالب بندی دکس

قالب‌بندی دکس کار ساده‌ای نیست زیرا اغلب ما آن را با استفاده از یک فونت کوچک درون یک جعبه متن می‌نویسیم. بسته به نسخه *Power BI*، *Excel* و *Visual Studio* ویرایشگرهای متنی متفاوتی را برای دکس ارائه می‌کنند. با این وجود، چند نکته ممکن است به نوشتن کد دکس کمک کند:

- اگر شما خواهان افزایش اندازه قلم می‌باشید تا دیدن آن ساده شود، می‌توانید کلید *Ctrl* را نگه داشته و در همان حال کلید میانی موس را بچرخانید.
- برای افزودن یک خط جدید به فرمول، کلیدهای ترکیبی *Shift+Enter* را فشار دهید.
- اگر ویرایش در جعبه متن دشوار است، می‌توانید همیشه کد را در یک نرم افزار ویرایش متنی دیگر، همانند *Notepad* یا *DAX Studio* کپی کرده و سپس فرمول را دوباره در جعبه متن جای‌گذاری کنید.

وقتی به یک عبارت دکس نگاه می‌کنید، در نگاه اول ممکن است درک اینکه آیا یک ستون محاسبه شده است یا یک معیار دشوار باشد. بنابراین، در کتاب‌ها و مقالات خودمان، هر زمان که ستون محاسبه‌شده را تعریف می‌کنیم، از علامت مساوی (=) و عملگر قراردادی (:=) برای تعریف معیارها استفاده می‌کنیم:

```
CalcCol = SUM ( Sales[SalesAmount] )          -- یک ستون محاسبه شده است
Store[CalcCol] = SUM ( Sales[SalesAmount] )   -- یک ستون محاسبه شده در جدول Store است
CalcMsr := SUM ( Sales[SalesAmount]  )        -- یک معیار است
```

در نهایت، هنگام استفاده از ستون‌ها و معیارها در کد، توصیه می‌کنیم همیشه نام جدول را قبل از نام یک ستون قرار دهید و هرگز آن را قبل از یک معیار قرار ندهید، همان‌طور که ما در هر مثالی چنین می‌کنیم.

معرفی تجمیع‌کننده‌ها و پیمایشگرها

تقریباً هر مدل داده‌ای باید بر روی داده‌های تجمیع شده کار کند. دکس مجموعه‌ای از توابع ارائه می‌دهد که مقادیر یک ستون

در یک جدول را تجمیع می‌کنند و یک مقدار واحد را برمی‌گردانند. ما این گروه از توابع را *توابع تجمیعی*[۱] می‌نامیم. به عنوان مثال، معیار زیر مجموع تمام اعداد موجود در ستون SalesAmount جدول Sales را محاسبه می‌کند:

```
Sales := SUM ( Sales[SalesAmount] )
```

تابع SUM اگر در یک ستون محاسبه‌شده استفاده شود، تمام ردیف‌های جدول را تجمیع می‌کند. هر زمان که در یک معیار ا ستفاده می شود، فقط ردیف‌هایی را در نظر می‌گیرد که تو سط بر شگرها، ردیف‌ها، ستون‌ها و شرایط فیلتر در گزارش فیلتر می‌شوند.

توابع تجمیعی بسیاری وجود دارد (SUM، AVERAGE، MIN، MAX، و STDEV)، و رفتار آن‌ها فقط در شیوه‌ای که مقادیر را تجمیع می‌کنند تغییر می‌کند: SUM مقادیر را جمع می‌کند، در حالی که MIN حداقل مقدار را برمی‌گرداند. تقریباً همه این توابع فقط بر روی مقادیر عددی یا تاریخ عمل می‌کنند. تنها MIN و MAX می‌توانند روی مقادیر متنی نیز کار کنند. علاوه بر این، دکس هرگز سلول‌های خالی را هنگام انجام تجمیع در نظر نمی‌گیرد، و این رفتار با همتایان آن‌ها در *Excel* متفاوت می‌باشد (در ادامه این فصل بیشتر در مورد آن توضیح خواهیم داد).

نکته توابع MIN و MAX رفتار دیگری نیز ارائه می‌دهند: اگر با دو پارامتر استفاده شوند، حداقل یا حداکثر دو پارامتر را برمی‌گردانند. بنابراین، *MIN(1,2)* مقدار ۱ را برمی‌گرداند و *MAX(1,2)* مقدار ۲ را باز می‌گرداند. این عملکرد زمانی مفید است که فرد نیاز به محاسبه حداقل یا حداکثر عبارت‌های پیچیده دارد، چرا که الزام نوشتن چندباره یک عبارت را در دستورات *IF* از بین می‌برد.

تمام توابع تجمیعی که تاکنون توضیح داده‌ایم روی ستون‌ها کار می‌کنند. بنابراین، آن‌ها مقادیر را تنها از یک ستون تجمیع می‌کنند. برخی از توابع تجمیعی می‌توانند یک عبارت را به جای یک ستون واحد تجمیع کنند. به دلیل نحوه کار آن‌ها به عنوان پیمایشگر شناخته می شوند. این مجموعه از توابع مفید می‌با شند، به ویژه زمانی که نیاز به محا سباتی با ا ستفاده از ستون‌های جداول مختلف و مرتبط دارید، یا زمانی که باید تعداد ستون‌های محاسبه‌شده را کاهش دهید.

پیمایشگرها همیشه حداقل دو پارامتر را می‌پذیرند: اولین پارامتر جدولی ا ست که آن‌ها آن را ا سکن می‌کنند؛ دومین پارامتر معمولاً عبارتی ا ست که برای هر ردیفی از جدول ارزیابی می شود. پس از اتمام ا سکن جدول و ارزیابی عبارت به صورت سطربه‌سطر، پیمایشگرها نتایج جزئی را با توجه به معانی خودشان تجمیع می‌کنند.

به عنوان مثال، اگر تعداد روزهای مورد نیاز برای تحویل سفارش را در یک ستون محاسبه‌شده به نام DaysToDeliver محاسبه کنیم و گزارشی را بر اساس آن بسازیم، گزارش نشان داده شده در شکل ۲-۶ را به دست می‌آوریم. توجه داشته باشید که مجموع کل جمع تمام روزها را نشان می‌دهد که برای این سنجه[۲] مفید نیست:

```
Sales[DaysToDeliver] = INT ( Sales[Delivery Date] - Sales[Order Date] )
```

---

۱ - *Aggregation Functions*

۲ - *Metric*

SalesKey	Order Date	Delivery Date	DaysToDeliver
200701022CS425-0013	01/02/2007	01/08/2007	6
200701022CS425-0014	01/02/2007	01/09/2007	7
200701022CS425-0015	01/02/2007	01/10/2007	8
200701022CS425-0016	01/02/2007	01/11/2007	9
200701022CS425-0017	01/02/2007	01/12/2007	10
200701022CS425-0018	01/02/2007	01/13/2007	11
200701023CS425-0202	01/02/2007	01/08/2007	6
200701023CS425-0203	01/02/2007	01/09/2007	7
200701023CS425-0204	01/02/2007	01/10/2007	8
200701023CS425-0205	01/02/2007	01/11/2007	9
Total			**848075**

شکل ۲-۶ مجموع کل به‌صورت جمع نشان داده می‌شود، زمانی که ممکن است شما به جای آن یک میانگین بخواهید.

مجموع کلی که در واقع می‌توانیم از آن ا ستفاده کنیم، به معیاری به نام AvgDelivery نیاز دارد که زمان تحویل برای هر سفارش و میانگین تمام مدت‌زمان را در سطح مجموع کل نشان می‌دهد:

```
AvgDelivery := AVERAGE ( Sales[DaysToDeliver] )
```

نتیجه این معیار جدید در گزارش نشان داده شده در شکل ۲-۷ قابل مشاهده است.

SalesKey	Order Date	Delivery Date	DaysToDeliver	AvgDelivery
200701022CS425-0013	01/02/2007	01/08/2007	6	6.00
200701022CS425-0014	01/02/2007	01/09/2007	7	7.00
200701022CS425-0015	01/02/2007	01/10/2007	8	8.00
200701022CS425-0016	01/02/2007	01/11/2007	9	9.00
200701022CS425-0017	01/02/2007	01/12/2007	10	10.00
200701022CS425-0018	01/02/2007	01/13/2007	11	11.00
200701023CS425-0202	01/02/2007	01/08/2007	6	6.00
200701023CS425-0203	01/02/2007	01/09/2007	7	7.00
200701023CS425-0204	01/02/2007	01/10/2007	8	8.00
200701023CS425-0205	01/02/2007	01/11/2007	9	9.00
Total			**848075**	**8.46**

شکل ۲-۷ معیار تجمیعی از طریق میانگین، میانگین روزهای تحویل را در سطح مجموع کل نشان می‌دهد.

این معیار مقدار میانگین را با گرفتن میانگین از یک ستون محا سبه شده حـساب می‌کند. با ا ستفاده از یک پیمایشگر فرد می‌تواند ستون محاسبه‌شده را حذف کند، که منجر به صرفه‌جویی فضا در مدل می‌شود. در واقع، اگرچه درست است که AVERAGE نمی‌تواند میانگین یک عبارت را حســاب کند، همتای آن AVERAGEX می‌تواند جدول Sales را پیمایش و روزهای تحویل را ردیف‌به‌ردیف محاسـبه کند و در پایان میانگین نتایج را بگیرد. کد زیر همان نتیجه تعریف قبلی را به دسـت می‌آورد:

```
AvgDelivery :=
AVERAGEX (
    Sales,
    INT ( Sales[Delivery Date] - Sales[Order Date] )
)
```

بزرگ‌ترین مزیت عبارت آخر این است که به وجود یک ستون محاسبه‌شده متکی نیست. بنابراین، ما می‌توانیم کل گزارش را بدون ایجاد ستون‌های محاسبه‌شده پرهزینه ایجاد کنیم.

اکثر پیمایشگرها نامی مشابه با همتای غیرپیمایشگر خود دارند. به عنوان مثال، SUM دارای یک SUMX منتاظر، و MIN دارای یک MINX متناظر اسـت. با این وجود، به خاطر داشـته باشـید که برخی از پیمایشـگرها با هیچ تجمیع‌کننده‌ای مطابقت ندارند. بعداً در این کتاب، با FILTER، ADDCOLUMNS، GENERATE، و توابع دیگر آشـنا خواهید شـد که حتی اگر نتایج خود را تجمیع نکنند، پیمایشگر هستند.

وقتی برای اولین بار دکس را یاد می‌گیرید، ممکن است فکر کنید که پیمایشگرها ذاتاً کند هستند. مفهوم انجام محاسبات ردیف‌به‌ردیف شبیه یک عملیات فشرده *CPU* است. در واقع، پیمایشگرها سریع هستند و هیچ جریمه عملکردی برای استفاده از پیمایشگرها به جای تجمیع‌کننده‌های استاندارد ایجاد نمی‌شود. تجمیع‌کننده‌ها فقط یک نسخه ساده[۱] از پیمایشگرها هستند.

در واقع، توابع تجمیعی پایه نسخه کوتاه شده توابع متناظر همراه با پسوند *X*[۲] هستند. برای مثال عبارت زیر را در نظر بگیرید:

```
SUM ( Sales[Quantity] )
```

از نظر داخلی آن به این نسخه متناظر از همان کد ترجمه می‌شود:

```
SUMX ( Sales, Sales[Quantity] )
```

تنها مزیت استفاده از *SUM* ترکیب کوتاه‌تر آن است. با این حال، هیچ تفاوتی در عملکرد بین *SUM* و *SUMX* در تجمیع یک ستون وجود ندارد. آن‌ها از همه نظر عملکرد یکسانی دارند.

ما در فصل ۴ جزئیات بیشتری را در مورد این رفتار ارائه خواهیم داد. در آنجا مفهوم زمینه‌های ارزیابی را معرفی می‌کنیم تا به درستی نحوه کار پیمایشگرها را توضیح دهیم.

استفاده از توابع رایج دکس

اکنون که مبانی دکس و نحوه رسیدگی به وضعیت‌های خطا را دیدید، آنچه در ادامه می‌آید توضیح مختصری از متداول‌ترین توابع و عبارت‌های دکس است.

توابع تجمیعی

در بخش‌های قبلی، تجمیع‌کننده‌های اصلی مانند SUM، AVERAGE، MIN و MAX را توضیح دادیم. شما یاد گرفتید که برای مثال SUM و AVERAGE فقط روی ستون‌های عددی کار می‌کنند.

همچنین دکس یک ترکیب جایگزین برای توابع تجمیعی که از *Excel* به ارث برده است ارائه می‌دهد که پسوند *A* را به نام تابع اضافه می‌کند تا همان نام و رفتار *Excel* را بدست آورد. با این حال، این توابع فقط برای ستون‌های حاوی مقادیر بولین مفید می‌باشند زیرا TRUE به‌صورت ۱ و FALSE به شکل ۰ ارزیابی می‌شوند. ستون‌های متنی همیشه ۰ در نظر گرفته می شوند. بنابراین، مهم نیست که چه چیزی در محتوای ستون وجود دارد، اگر فرد از MAXA در یک ستون متنی استفاده کند، نتیجه همیشه ۰ خواهد بود. علاوه بر این، دکس هرگز سلول‌های خالی را هنگام انجام تجمیع در نظر نمی‌گیرد. اگرچه این توابع را می‌توان بر روی ستون‌های غیر عددی بدون بازگرداندن خطا استفاده کرد، اما نتایج آن‌ها مفید نیستند زیرا هیچ تبدیل به اعداد خودکاری برای ستون‌های متنی صورت نمی‌گیرد. این توابع AVERAGEA، COUNTA، MINA و MAXA نام دارند. پیشنهاد می‌کنیم از این توابع استفاده نکنید، زیرا رفتار آن‌ها در آینده به دلیل سازگاری با کد موجود که ممکن است بر رفتار فعلی متکی باشد، بدون تغییر باقی می‌ماند.

---

۱ - *Syntax-Sugared*

۲ - *X-Suffixed*

نکته علی‌رغم اینکه این نام‌ها با توابع آماری یکسان هستند، در دکس و اکسل به طور متفاوتی استفاده می‌شوند زیرا در دکس یک ستون دارای یک نوع داده است، و نوع داده آن رفتار توابع تجمیعی را تعیین می‌کند. اکسل نوع داده متفاوتی را برای هر سلول مشخص می‌کند، در حالی که دکس تنها یک نوع داده را برای کل ستون در نظر می‌گیرد. دکس با داده‌ها به شکل جدولی با انواع کاملاً تعریف‌شده برای هر ستون سروکار دارد، در حالی که فرمول‌های اکسل روی مقادیر سلول‌های ناهمگن بدون انواع خوب تعریف‌شده کار می‌کنند. اگر ستونی در *Power BI* دارای نوع داده عددی باشد، تمام مقادیر می‌توانند فقط اعداد یا سلول‌های خالی باشند. اگر یک ستون از نوع متنی باشد، همیشه برای این توابع ۰ می‌باشد (به جز COUNTA)، حتی اگر بتوان متن را به عدد تبدیل کرد، در حالی که در اکسل این مقدار به شکل سلول به سلول یک عدد در نظر گرفته می‌شود. به این دلایل، این توابع برای ستون‌های *Text* چندان مفید نیستند. فقط MIN و MAX از مقادیر متنی در دکس نیز پشتیبانی می‌کنند.

توابعی که شما پیش از این یاد گرفتید برای تجمیع مقادیر سودمند می‌با شند. گاهی اوقات، شما علاقه‌مند به تجمیع مقادیر نی ستید و صرفاً خواهان شمارش آن‌ها می‌با شید. بنابراین، دکس مجموعه‌ای از توابع را ارائه می‌دهد که برای شمارش ردیف‌ها یا مقادیر سودمند می‌باشند.

- تابع COUNT بر روی هر نوع داده‌ای به غیر از بولین عمل می‌کند.
- تابع COUNTA بر روی هر نوع ستونی عمل می‌کند.
- تابع COUNTBLANK تعداد سلول‌های خالی (تهی[1] یا رشته‌های خالی) در یک ستون را باز می‌گرداند.
- تابع COUNTROWS تعداد ردیف‌های جدول را باز می‌گرداند.
- تابع DISTINCTCOUNT تعداد مقادیر متمایز یک ستون را باز می‌گرداند، مقدار خالی نیز اگر وجود داشته باشد لحاظ می‌شود.
- تابع DISTINCTCOUNTNOBLANK تعداد مقادیر متمایز ستون را برمی‌گرداند، بدون در نظر گرفتن مقدار خالی.

COUNT و COUNTA توابع تقریباً یکسانی در دکس هستند. آن‌ها تعداد مقادیر ستونی را که خالی نیستند، صرف‌نظر از نوع داده آن‌ها، برمی‌گردانند. آن‌ها از *Excel* به ارث رسیده‌اند، جایی که COUNTA هر نوع داده‌ای از جمله رشته‌ها را می‌پذیرد، در حالی که COUNT فقط ستون‌های عددی را می‌پذیرد. اگر بخواهیم تمام مقادیر یک ستون را که دارای یک مقدار خالی هستند بشماریم، می‌توانیم از تابع COUNTBLANK استفاده کنیم. هم مقادیر خالی و هم مقادیر تهی توسط COUNTBLANK مقادیر خالی در نظر گرفته می‌شوند. در نهایت اگر بخواهیم تعداد ردیف‌های یک جدول را بشماریم می‌توانیم از تابع COUNTROWS ا ستفاده کنیم. مراقب با شید که COUNTROWS به یک جدول به عنوان پارامتر نیاز دارد، نه ستون.

دو تابع آخر، DISTINCTCOUNT و DISTINCTCOUNTNOBLANK، مفید هستند زیرا دقیقاً همان کاری را انجام می‌دهند که نام آن‌ها نشان می‌دهد: شمارش مقادیر متمایز ستونی که به عنوان تنها پارامتر خود اتخاذ می‌کنند. DISTINCTCOUNT مقدار *BLANK* را به عنوان یکی از مقادیر ممکن می‌شمارد، در حالی که *DISTINCTCOUNTNOBLANK* مقدار *BLANK* را نادیده می‌گیرد.

نکته DISTINCTCOUNT تابعی است که در نسخه ۲۰۱۲ دکس معرفی شده است. نسخه‌های قبلی دکس شامل DISTINCTCOUNT نبودند؛ برای محاسبه تعداد مقادیر متمایز یک ستون، باید از

[1] - *Empty*

COUNTROWS (DISTINCT (table[column])) استفاده کنیم. این دو الگو نتیجه یکسانی را برمی‌گردانند، اگرچه خواندن DISTINCTCOUNT آسان‌تر است و تنها به فراخوانی یک تابع نیاز دارد. DISTINCTCOUNTNOBLANK تابعی است که در سال ۲۰۱۹ معرفی شد و همان معنای یک عملیات COUNT DISTINCT را در *SQL* بدون نیاز به نوشتن عبارت طولانی‌تر در دکس ارائه می‌کند.

توابع منطقی[1]

گاهی اوقات می‌خواهیم یک شرط منطقی در یک عبارت بسازیم—برای مثال، محاسبات مختلف را بسته به مقدار یک ستون اجرا کنیم یا جلوی یک و ضعیت خطا را بگیریم. در این موارد می‌توانیم از یکی از توابع منطقی در دکس ا ستفاده کنیم. بخش قبلی با عنوان «مدیریت خطاها در عبارت‌های دکس» مهم‌ترین دو تابع این گروه را تشریح کرد: IF و IFERROR. ما تابع IF را در بخش «دستورات شرطی» در اوایل این فصل توضیح دادیم.

توابع منطقی بسیار ساده هستند و آنچه را انجام می‌دهند که نام آن‌ها نشان می‌دهد. آن‌ها AND، FALSE، IF، IFERROR، NOT، TRUE و OR هستند. به عنوان مثال، اگر بخواهیم میزان ضرب مقدار در قیمت را فقط زمانی محاسبه کنیم که ستون Price حاوی یک مقدار عددی باشد، می‌توانیم از الگوی زیر استفاده کنیم:

```
Sales[Amount] = IFERROR ( Sales[Quantity] * Sales[Price], BLANK ( ) )
```

اگر ما از تابع IFERROR استفاده نکنیم و اگر ستون Price حاوی یک عدد نامعتبر باشد، نتیجه برای ستون محاسبه‌شده یک خطا خواهد بود چرا که اگر یک ردیف خطای محاسباتی ایجاد کند، این خطا به کل ستون تسری می‌یابد. به هر حال، کاربرد IFERROR برای ردیابی خطا و جایگزین کردن آن با یک مقدار خالی می‌باشد.

دیگر تابع جالب درون این دسته تابع SWITCH، زمانی سودمند است که ما ستونی داریم که این ستون حاوی تعداد کمی مقادیر متمایز می‌باشد، و می‌خواهیم بر اساس مقدار آن رفتارهای متفاوتی داشته باشیم. برای مثال، ستون Size در جدول Product حاوی *L, M, S, XL* می‌باشد و ممکن است بخواهیم این مقدار را در یک ستون واضح‌تر رمزگشایی کنیم. ما می‌توانیم با استفاده از فراخوانی‌های تودرتوی IF این نتیجه را به دست آوریم:

```
'Product'[SizeDesc] =
IF (
    'Product'[Size] = "S",
    "Small",
    IF (
        'Product'[Size] = "M",
        "Medium",
        IF (
            'Product'[Size] = "L",
            "Large",
            IF (
                'Product'[Size] = "XL",
                "Extra Large",
                "Other"
            )
        )
    )
)
```

یک مسیر راحت‌تر برای بیان همان فرمول، با استفاده از SWITCH، به شرح زیر است:

[1] - *Logical Functions*

```
'Product'[SizeDesc] =
SWITCH (
    'Product'[Size],
    "S", "Small",
    "M", "Medium",
    "L", "Large",
    "XL", "Extra Large",
    "Other"
)
```

کد عبارت آخر خواناتر می‌باشد، اگرچه سریع‌تر نیست، چرا که دکس در داخل دستورات SWITCH را به یک مجموعه از توابع IF تودرتو ترجمه می‌کند.

نکته SWITCH اغلب برای بررسی مقدار یک پارامتر و تعریف نتیجه یک معیار استفاده می‌شود. به عنوان مثال، ممکن است فرد یک جدول پارامتر شامل YTD، MTD، QTD را به عنوان سه ردیف ایجاد کند و به کاربر اجازه دهد از بین سه ردیف موجود انتخاب کند که کدام تجمیع را در یک معیار استفاده کند. این یک سناریوی معمول قبل از سال ۲۰۱۹ بود. اکنون به لطف معرفی گروه‌های محاسباتی که در فصل ۹ «گروه‌های محاسباتی» پوشش داده شده است، دیگر به آن نیازی نیست. گروه‌های محاسباتی روش ارجح‌تری برای محاسبه مقادیری هستند که کاربر می‌تواند آن‌ها را به شکل پارامتر بیان کند.

رهنمود یک روش جالب برای استفاده از تابع SWITCH به منظور بررسی شرایط متعدد در یک عبارت وجود دارد. از آنجایی که SWITCH به مجموعه‌ای از توابع IF تودرتو تبدیل می‌شود، جایی که اولین موردی که مطابقت دارد برنده می‌شود، می‌توانید چند شرط را با استفاده از این الگو آزمایش کنید.

```
SWITCH (
   TRUE (),
   Product[Size] = "XL" && Product[Color] = "Red", "Red and XL",
   Product[Size] = "XL" && Product[Color] = "Blue", "Blue and XL",
   Product[Size] = "L" && Product[Color] = "Green", "Green and L"
)
```

استفاده از TRUE به عنوان اولین پارامتر به این معنی است که «اولین نتیجه را برگردانید، در جایی که شرط به شکل TRUE ارزیابی می‌شود.»

توابع اطلاعاتی[1]

هر زمان که نیاز به تجزیه‌وتحلیل نوع یک عبارت باشد، می‌توانید یکی از توابع اطلاعاتی را به کار ببرید. تمام این توابع اطلاعاتی یک مقدار بولین را باز می‌گردانند و می‌توانند در هر نوع عبارت منطقی استفاده شوند. این توابع عبارت‌اند از: ISBLANK، ISERROR، ISLOGICAL، ISNONTEXT، ISNUMBER و ISTEXT.

توجه به این نکته ضروری ا ست که وقتی یک ستون به‌جای یک عبارت به عنوان پارامتر در نظر گرفته می شود، توابع

[1] - *Information Functions*

ISNUMBER، ISTEXT و ISNONTEXT همیشه بسته به نوع داده ستون و وضعیت خالی هر سلول، TRUE یا FALSE را برمی‌گردانند. این باعث می‌شود که این توابع در دکس تقریباً بی‌فایده باشند. آن‌ها در اولین نسخه دکس از *Excel* به ارث رسیده‌اند.

ممکن است از خود بپرسید که آیا می‌توانید از ISNUMBER با یک ستون از نوع متن فقط برای بررسی امکان تبدیل به یک عدد استفاده کنید. متأسفانه این رویکرد امکان‌پذیر نیست. اگر می‌خواهید بررسی کنید که آیا یک مقدار متنی قابل تبدیل به عدد است یا خیر، باید تبدیل را امتحان کنید و در صورت عدم موفقیت، خطا را پوشش دهید. به عنوان مثال، برای آزمایش اینکه آیا ستون Price (که از نوع رشته است) دارای یک عدد معتبر است، فرد باید بنویسد

```
Sales[IsPriceCorrect] = NOT ISERROR ( VALUE ( Sales[Price] ) )
```

دکس سعی می‌کند از یک مقدار رشته‌ای به یک عدد تبدیل کند. اگر موفق شد، TRUE را برمی‌گرداند (زیرا ISERROR مقدار FALSE را برمی‌گرداند). در غیر این صورت، FALSE را برمی‌گرداند (*زیرا* ISERROR مقدار TRUE را برمی‌گرداند). برای مثال، اگر برخی از ردیف‌ها دارای مقدار رشته‌ای «*N/A*» برای قیمت باشند، تبدیل شکست می‌خورد

با این حال، اگر سعی کنیم از ISNUMBER مانند عبارت زیر استفاده کنیم، همیشه مقدار FALSE را به عنوان نتیجه دریافت می‌کنیم:

```
Sales[IsPriceCorrect] = ISNUMBER ( Sales[Price] )
```

در این مورد، ISNUMBER همیشه FALSE را برمی‌گرداند، زیرا بر اساس تعریف موجود در مدل، ستون Price بدون توجه به محتوای هر ردیف یک عدد نیست بلکه یک رشته است.

توابع ریاضیاتی[۱]

مجموعه توابع ریاضی موجود در دکس مشابه مجموعه توابع ریاضیاتی موجود در *Excel* می‌باشد، با ترکیب و رفتار یکسان. توابع ریاضیاتی رایج عبارتند از ABS، EXP، FACT، LN، LOG، LOG10، MOD، PI، POWER، QUOTIENT، SIGN و SQRT. توابع تصادفی RAND و RANDBETWEEN هستند. با استفاده از EVEN و ODD می‌توانید اعداد را بررسی کنید. GCD و LCM برای محاسبه بزرگ‌ترین مخرج مشترک و کوچک‌ترین مضرب مشترک دو عدد مفید می‌باشند. QUOTIENT عددصحیح تقسیم دو عدد را برمی‌گرداند.

در نهایت، چند تابع گردکننده وجود دارد که شایسته ارائه مثالی مربوط به آن‌ها هستند، در حقیقت، شاید ما از چند روش برای رسیدن به یک هدف استفاده کنیم. به این ستون‌های محاسباتی همراه با نتایج آن‌ها در شکل ۲-۸ توجه کنید:

```
FLOOR = FLOOR ( Tests[Value], 0.01 )
TRUNC = TRUNC ( Tests[Value], 2 )
ROUNDDOWN = ROUNDDOWN ( Tests[Value], 2 )
MROUND = MROUND ( Tests[Value], 0.01 )
ROUND = ROUND ( Tests[Value], 2 )
CEILING = CEILING ( Tests[Value], 0.01 )
ISO.CEILING = ISO.CEILING ( Tests[Value], 0.01 )
ROUNDUP = ROUNDUP ( Tests[Value], 2 )
INT = INT ( Tests[Value] )
FIXED = FIXED ( Tests[Value], 2, TRUE )
```

[۱] - *Mathematical Functions*

Test	Value	FLOOR	TRUNC	ROUNDDOWN	MROUND	ROUND	CEILING	ISO.CEILING	ROUNDUP	INT	FIXED
A	1.123450	1.12	1.12	1.12	1.12	1.12	1.13	1.13	1.13	1	1.12
B	1.265000	1.26	1.26	1.26	1.26	1.27	1.27	1.27	1.27	1	1.27
C	1.265001	1.26	1.26	1.26	1.27	1.27	1.27	1.27	1.27	1	1.27
D	1.499999	1.49	1.49	1.49	1.50	1.50	1.50	1.50	1.50	1	1.50
E	1.511110	1.51	1.51	1.51	1.51	1.51	1.52	1.52	1.52	1	1.51
F	1.000001	1.00	1.00	1.00	1.00	1.00	1.01	1.01	1.01	1	1.00
G	1.999999	1.99	1.99	1.99	2.00	2.00	2.00	2.00	2.00	1	2.00

شکل ۲-۸ این خلاصه نتایج استفاده از توابع مختلف گردکننده را نشان می‌دهد.

FLOOR، TRUNC، و ROUNDDOWN مشابه یکدیگر هستند، به جز شیوه‌ای که می‌توانیم تعداد ارقام را برای گرد کردن مشخص بکنیم. در جهت مخالف، CEILING و ROUNDUP از نظر نتایج مشابه یکدیگر می‌باشند. شما می‌توانید تفاوت کمی در نحوه گرد کردن بین تابع MROUND و ROUND مشاهده کنید.

توابع مثلثاتی[۱]

دکس مجموعه‌ای غنی از توابع مثلثاتی را ارائه می‌دهد که برای محاسبات خاص مفید هستند: COS، COSH، COT، COTH، SIN، SINH، TAN، و TANH. پیشوند آن‌ها با *A*، نسخه آرک[۲] (آرکسین، آرکوزین و غیره) را محاسبه می‌کند. ما وارد جزئیات این توابع نمی‌شویم چرا که استفاده از آن‌ها ساده است.

DEGREES و RADIANS به ترتیب تبدیل به درجه و رادیان را انجام می‌دهند و SQRTPI جذر پارامتر آن را پس از ضرب در عدد پی محاسبه می‌کند.

توابع متنی[۳]

اکثر توابع متنی موجود در دکس مشابه توابع موجود در *Excel* می‌باشند، تنها با چند استثناء. توابع متنی عبارتند از CONCATENATE، CONCATENATEX، EXACT، FIND، FIXED، FORMAT، LEFT، LEN، LOWER، MID، REPLACE، REPT، RIGHT، SEARCH، SUBSTITUTE، TRIM، UPPER و VALUE. این توابع برای دست‌کاری متن و استخراج داده‌ها از رشته‌هایی که حاوی مقادیر متعدد هستند مفید می‌باشند. برای مثال، شکل ۲-۹ نمونه‌ای از استخراج نام و نام خانوادگی از رشته‌ای را نشان می‌دهد که حاوی مقادیری است که با کاما از هم جدا شده‌اند و عنوانی در وسط آن قرار دارد که می‌خواهیم حذف کنیم.

Name	Comma1	Comma2	FirstLastName	SimpleConversion
Ferrari, Alberto	8		Alberto Ferrari	Ferrari, Alberto Ferrari
Ferrari, Mr., Alberto	8	13	Alberto Ferrari	Alberto Ferrari
Russo, Mr., Marco	6	11	Marco Russo	Marco Russo

شکل ۲-۹ این مثال نام و نام خانوادگی استخراج شده با استفاده از توابع متنی را نشان می‌دهد.

ما برای رسیدن به این نتیجه، شروع به محاسبه موقعیت دو کاما می‌کنیم. سپس از این اعداد برای استخراج بخش درست متن استفاده می‌کنیم. ستون SimpleConversion فرمولی را پیاده‌سازی می‌کند که اگر کمتر از دو کاما در رشته وجود داشته باشد، ممکن است مقادیر نادرست را برگرداند، و اگر اصلاً کاما وجود نداشته باشد، خطا ایجاد می‌کند. ستون FirstLastName عبارت پیچیده‌تری را پیاده‌سازی می‌کند که در صورت از دست دادن کاما شکست نمی‌خورد:

---

۱ - *Trigonometric Functions*

۲ - *Arc*

۳ - *Text Functions*

```
People[Comma1] = IFERROR ( FIND ( ",", People[Name] ), BLANK ( ) )
People[Comma2] = IFERROR ( FIND ( " ,", People[Name], People[Comma1] + 1 ), BLANK ( ) )
People[SimpleConversion] =
MID ( People[Name], People[Comma2] + 1, LEN ( People[Name] ) )
    & " "
    & LEFT ( People[Name], People[Comma1] - 1 )
People[FirstLastName] =
TRIM (
    MID (
        People[Name],
        IF ( ISNUMBER ( People[Comma2] ), People[Comma2], People[Comma1] ) + 1,
        LEN ( People[Name] )
    )
)
    & IF (
        ISNUMBER ( People[Comma1] ),
        " " & LEFT ( People[Name], People[Comma1] - 1 ),
        ""
    )
```

همان‌طور که می‌بینید، ستون FirstLastName با یک عبارت دکس طولانی تعریف شده است، اما شما باید از آن برای جلوگیری از خطاهای احتمالی استفاده کنید چراکه اگر تنها یک مقدار خطا ایجاد کند، به کل ستون تسری می‌یابد.

توابع تبدیلی[۱]

شما قبلاً یاد گرفتید که دکس به صورت خودکار تبدیل انواع داده‌ها را برای سازگار کردن آن‌ها با نیاز عملگرها انجام می‌دهد. اگرچه این تبدیل به‌صورت خودکار انجام می‌شود، یک مجموعه از توابع هنوز می‌توانند تبدیلات ساده انواع داده را انجام دهد.

تابع CURRENCY می‌تواند یک عبارت را به نوع ارزی تبدیل کند، در حالی که تابع INT یک عبارت را به نوع عددصحیح تبدیل می‌کند. DATE و TIME دوره‌های تاریخ و زمان را به عنوان پارامتر قبول می‌کنند و DateTime صحیحی را باز می‌گرداند. تابع VALUE یک رشته را به یک قالب عددی تبدیل می‌کند، درحالی‌که تابع FORMAT یک مقداری عددی را به عنوان اولین پارامتر و یک قالب رشته‌ای را به عنوان دومین پارامتر می‌گیرد و می‌تواند مقادیر عددی را به رشته‌ای تبدیل کند. تابع FORMAT بیشتر با DateTime مورد استفاده قرار می‌گیرد. برای مثال، عبارت زیر این تاریخ را باز می‌گرداند «*2019 Jan 12*».

```
= FORMAT ( DATE ( 2019, 01, 12 ), "yyyy mmm dd" )
```

عمل مخالف که مقادیر رشته‌ای را به مقادیر DateTime تبدیل می‌کند، با استفاده از تابع DATEVALUE انجام می‌شود.

[۱] - *Conversion Functions*

DATEVALUE به همراه تاریخ با قالب‌های مختلف

DATEVALUE رفتار خاصی را در رابطه با تاریخ‌ها در قالب‌های مختلف نشان می‌دهد. در استاندارد اروپا، تاریخ‌ها با قالب «*dd/mm/yy*» نوشته می‌شوند، در حالی که آمریکایی‌ها ترجیح می‌دهند از قالب «*mm/dd/yy*» استفاده کنند. به عنوان مثال، روز ۲۸ فوریه در این دو فرهنگ، نمایش‌های رشته‌ای متفاوتی دارد. اگر به DATEVALUE تاریخی ارائه کنید که با استفاده از تنظیمات پیش‌فرض منطقه‌ای قابل تبدیل نباشد، به جای اینکه فوراً خطایی ایجاد کند، تبدیل ثانویه ماه و روز را انجام می‌دهد. DATEVALUE همچنین از قالب بدون ابهام «*yyyy-mm-dd*» پشتیبانی می‌کند. به عنوان مثال، سه عبارت زیر به‌صورت ۲۸ فوریه ارزیابی می‌شوند، بدون توجه به تنظیمات منطقه‌ای شما:

```
DATEVALUE ( "28/02/2018" )      -- این روز ۲۸ فوریه در قالب اروپایی است
DATEVALUE ( "02/28/2018" )      -- این روز ۲۸ فوریه در قالب آمریکایی است
DATEVALUE ( "2018-02-28" )      -- این روز ۲۸ فوریه است -قالب مبهم نیست
```

گاهی اوقات، DATEVALUE خطاها را در زمانی که شما انتظار دارید ارائه نمی‌دهد. با این حال، این رفتار تابع بر اساس طراحی می‌باشد.

توابع تاریخ و زمان[1]

تقریباً در هر نوع تجزیه‌وتحلیل داده، پوشش زمان و تاریخ بخش مهمی از کار است. بسیاری از توابع دکس در خصوص تاریخ و زمان کار می‌کنند. برخی از آن‌ها با توابع مشابه در *Excel* مطابقت دارند و تغییرات ساده‌ای را به و از یک نوع داده DateTime انجام می‌دهند. توابع تاریخ و زمان عبارتند از DATE، DATEVALUE، DAY، EDATE، EOMONTH، HOUR، MINUTE، MONTH، NOW، SECOND، TIME، TIMEVALUE، TODAY، WEEKDAY، WEEKNUM، YEAR، و YEARFRAC.

این توابع برای محاسبه مقادیر مربوط به تاریخ‌ها مفید می‌باشند، اما از آن‌ها برای انجام محاسبات هوش زمانی معمولی مانند مقایسه مقادیر تجمیعی نسبت به مدت مشابه سال قبل یا محاسبه مقدار ابتدای سال تا به امروز[2] یک معیار استفاده نمی‌شود. برای انجام محاسبات هوش زمانی، از مجموعه دیگری از توابع به نام توابع هوش زمانی استفاده می‌شود که در فصل ۸، «محاسبات هوش زمانی» توضیح می‌دهیم.

همان‌طور که قبلاً در این فصل ذکر کردیم، نوع داده DateTime به‌صورت داخلی از یک عدد ممیز شناور استفاده می‌کند که در آن قسمت صحیح با تعداد روزهای پس از ۳۰ دسامبر ۱۸۹۹ مطابقت دارد و قسمت اعشاری نشان دهنده کسری از روز از نظر زمانی است. ساعت‌ها، دقیقه‌ها و ثانیه‌ها به بخش‌های اعشاری روز تبدیل می شوند. بنابراین، افزودن یک عددصحیح به مقدار DateTime مقدار را با توجه به تعداد روز مربوطه افزایش می‌دهد. با این حال، احتمالاً استفاده از توابع تبدیلی برای استخراج روز، ماه و سال از یک تاریخ راحت‌تر است. عبارت‌های زیر که در شکل ۲-۱۰ استفاده شده‌اند نحوه استخراج این اطلاعات را از جدول حاوی فهرستی از تاریخ‌ها نشان می‌دهد:

```
'Date'[Day] = DAY ( Calendar[Date] )
'Date'[Month] = FORMAT ( Calendar[Date], "mmmm" )
'Date'[MonthNumber] = MONTH ( Calendar[Date] )
'Date'[Year] = YEAR ( Calendar[Date] )
```

[1] - *Date And Time Functions*

[2] - *Year-To-Date*

Date	Day	Month	Year
1/1/2010	1	January	2010
1/2/2010	2	January	2010
1/3/2010	3	January	2010
1/4/2010	4	January	2010
1/5/2010	5	January	2010
1/6/2010	6	January	2010
1/7/2010	7	January	2010
1/8/2010	8	January	2010
1/9/2010	9	January	2010

شکل ۲-۱۰ این مثال نحوه استخراج اطلاعات تاریخ را با استفاده از توابع تاریخ و زمان نشان می‌دهد.

توابع رابطه‌ای[1]

دو تابع مفید که می‌توانید برای پیمایش در روابط درون فرمول دکس استفاده کنید، RELATED و RELATEDTABLE هستند.

شما از قبل می‌دانید که یک ستون محاسبه‌شده می‌تواند به مقادیر ستون‌های جدولی که در آن تعریف شده است رجوع کند. بنابراین، یک ستون محاسبه‌شده در Sales می‌تواند به هر ستون Sales رجوع کند. با این حال، اگر فرد باید به ستونی در جدول دیگر مراجعه کند، چه باید کرد؟ به طور کلی فرد نمی‌تواند از ستون‌ها در جداول دیگر استفاده کند مگر اینکه رابطه‌ای در مدل بین دو جدول تعریف شده باشد. اگر دو جدول، یک رابطه مشترک دارند، می‌توانید از تابع RELATED برای دسترسی به ستون‌های جدول مرتبط استفاده کنید.

برای مثال، ممکن است فرد بخواهد یک ستون محاسبه‌شده در جدول Sales محاسبه کند که بررسی می‌کند آیا محصول فروخته شده در دسته مربوط به «*Cell phones*» قرار دارد یا خیر و در صورت صحیح بودن، یک ضریب کاهش برای هزینه استاندارد اعمال کند. برای محاسبه چنین ستونی، فرد باید از شرطی استفاده کند که مقدار دسته محصول را که در جدول Sales قرار ندارد، بررسی کند. با این وجود، زنجیره‌ای از روابط از جدول Sales شروع می‌شود و از طریق Product and Product Subcategory به Product Category می‌رسد، همان‌طور که در شکل ۲-۱۱ نشان داده شده است.

شکل ۲-۱۱ *Sales* یک رابطه زنجیره‌ای با *Product Category* دارد.

صرف‌نظر از اینکه چند مرحله برای حرکت از جدول اصلی به جدول مربوطه لازم است، دکس زنجیره کامل روابط را دنبال می‌کند و مقدار ستون مربوطه را برمی‌گرداند. بنابراین، فرمول ستون AdjustedCost می‌تواند به شکل زیر باشد:

[1] - *Relational Functions*

```
Sales[AdjustedCost] =
IF (
    RELATED ( 'Product Category'[Category] ) = "Cell Phone",
    Sales[Unit Cost] * 0.95,
    Sales[Unit Cost]
)
```

در یک رابطه یک‌به‌چند، تابع RELATED می‌تواند به سمت تکی رابطه از سمت چندی دسترسی داشته باشد چرا که در آن مورد، فقط یک ردیف در صورت وجود در جدول مربوطه وجود دارد. اگر یک چنین ردیفی وجود نداشته باشد، RELATED مقدار BLANK را باز می‌گرداند.

اگر عبارتی در سمت تکی رابطه با شد و نیاز به د ستر سی به طرف‌چندی با شد، RELATED مفید نیست زیرا ممکن است ردیف‌های زیادی از طرف دیگر برای یک ردیف در دسترس باشد. در این صورت می‌توانیم از RELATEDTABLE استفاده کنیم. RELATEDTABLE یک جدول حاوی تمام ردیف‌های مربوط به ردیف فعلی را برمی‌گرداند. به عنوان مثال، اگر بخواهیم بدانیم که در هر دسته چند محصول وجود دارد، می‌توانیم با این فرمول یک ستون در جدول Product Category ایجاد کنیم:

```
'Product Category'[NumOfProducts] = COUNTROWS ( RELATEDTABLE ( Product ) )
```

برای هر دسته محصول، این ستون محاسبه‌شده تعداد محصولات مرتبط را نشان می‌دهد، همان‌طور که در شکل ۲-۱۲ نشان داده شده است.

Category	NumOfProducts
Audio	115
Cameras and camcorders	372
Cell phones	285
Computers	606
Games and Toys	166
Home Appliances	661
Music, Movies and Audio Books	90
TV and Video	222

شکل ۲-۱۲ با استفاده از *RELATEDTABLE* می‌توانید تعداد محصولات را بشمارید.

همان‌طور که در مورد RELATED صدق می‌کند، RELATEDTABLE نیز می‌تواند زنجیره‌ای از روابط را دنبال کند که همیشه از سمت تکی شروع می شود و به سمت چندی رابطه می‌رود. RELATEDTABLE اغلب همراه با پیمایشگرها استفاده می شود. به عنوان مثال، اگر بخواهیم مجموع ضرب مقدار در قیمت خالص را برای هر دسته محاسبه کنیم، می‌توانیم یک ستون محاسبه‌شده جدید به‌صورت زیر بنویسیم:

```
'Product Category'[CategorySales] =
SUMX (
    RELATEDTABLE ( Sales ),
    Sales[Quantity] * Sales[Net Price]
)
```

نتیجه این ستون محاسبه‌شده در شکل ۲-۱۳ نشان داده شده است.

Category	CategorySales
Audio	$384,518.16
Cameras and camcorders	$7,192,581.95
Cell phones	$1,604,610.26
Computers	$6,741,548.73
Games and Toys	$360,652.81
Home Appliances	$9,600,457.04
Music, Movies and Audio Books	$314,206.74
TV and Video	$4,392,768.29

شکل ۲-۱۳ با استفاده از *RELATEDTABLE* و پیمایشگرها، می‌توانیم میزان فروش در هر دسته را محاسبه کنیم.

از آنجایی که این ستون محاسبه می‌شود، این نتیجه در جدول ادغام می‌شود و با توجه به انتخاب کاربر در گزارش تغییر نمی‌کند، به شکلی که اگر در یک معیار نوشته می‌شد تغییر می‌کرد.

نتیجه گیری

در این فصل، توابع جدید زیادی را یاد گرفتید و شروع به بررسی کدهای دکس کردید. ممکن است همه توابع را فوراً به خاطر نیاورید، اما هرچه بیشتر از آن‌ها استفاده کنید، با آن‌ها آشناتر می‌شوند.

عمده موضوعات مهمی که در این فصل یاد گرفتید عبارت‌انداز:

- ستون‌های محاسبه شده ستون‌هایی در جدول هستند که با یک عبارت دکس محاسبه می شوند. ستون‌های محاسبه‌شده در زمان به‌روزرسانی داده‌ها محاسبه می‌شوند و بسته به انتخاب کاربر، مقدار خود را تغییر نمی دهند.
- معیارها محاسباتی هستند که در دکس بیان می‌شوند. به جای اینکه مانند ستون‌های محاسبه‌شده در زمان به‌روزرسانی محاسبه شوند، در زمان کوئری محاسبه می‌شوند. در نتیجه، مقدار یک معیار به انتخاب کاربر در گزارش بستگی دارد.
- خطاها ممکن است در هر زمانی در یک عبارت دکس رخ دهند؛ بهتر است از قبل شرایط خطا را شناسایی کنید تا اینکه اجازه دهید خطا اتفاق بیفتد و بعد از وقوع آن را رهگیری کنید.
- تجمیع‌کننده‌هایی مانند SUM برای تجمیع ستون‌ها مفید هستند، در حالی که برای تجمیع عبارت‌ها، باید از پیمایشگرها استفاده کنید. پیمایشگرها با اسکن یک جدول و ارزیابی سطربه سطر یک عبارت کار می‌کنند. در پایان پیمایش، پیمایشگرها نتیجه را بر اساس معنای خود تجمیع می‌کنند.

در فصل بعد با مطالعه مهم‌ترین توابع جدولی موجود در دکس به مسیر یادگیری خود ادامه خواهید داد.