Database in depth:
relational theory for Practitioners

نگاهی عمیق تر به
پایگاه داده ها

نویسنده : سی جی دیت
مترجم : پویا فودهٔ

نظریه رابطه ای برای متخصصان

نویسنده : سی جی دیت
مترجم : پویا فودهٔ

نگاهی عمیق تر به پایگاه داده ها

در جهان امروز پایگاه های داده همه جا حضور دارند. از سا یت های بزرگ تجارت الکترونیکی مانند آمازون گرفته تا دستگاه های کوچک پخش MP3 که درکف دست شما جای می گیرند. معمولا حرفه ای ها نمی توانند در این زمینه بدون دانش زیربنایی در مورد مفاهیم اساسی نظریه رابطه ای ، موفقیت چندانی کسب نمایند. با این وجود امروزه بسیاری از کسانی که در شاخه پایگاه های داده مشغول به کار هستند، هیچگونه آموزش رسمی در زمینه نظریه رابطه ای را نگذرانده اند.
در نگاه عمیق تر به پایگاه داده ها نویسنده و محقق مشهور پایگاه داده ها ، کریس دیت اصول نظریه رابطه ای را شرح داده است. این مدل برای اولین بار توسط ای اف کاد بوسیله مقاله ای در سال ۱۹۶۹ معرفی شد.

عنوان این مقاله (استنتاج ، افزونگی و پایداری رابطه های ذخیره شده در بانک های اطلاعاتی بزرگ بود) و زیر بنای پایگاه های داده با کاربردهای گوناگون امروزی به شمار می رود.
در این کتاب کریس تصورات غلط بسیاری را در مورد نظریه رابطه ای، باطل می کند و توضیح می دهد که:
- کلمه « رابطه ای » هیچ ارتباطی با رابطه دو جدول با ستون های مشترک ندارد.
- رابطه ها چند بعدی هستند. آنها مسطح و دو بعدی نیستند. اجازه ندهید کلمه « جدول » شما را گمراه کند.
- تهی اصلا مقدار نیست. حتی اگر استانداردSQL آنرا مقدار بداند.
- ویژگی های یک رابطه می توانند تا حد دلخواه پیچیده و شامل چیزهایی مانند آرایه ها، مطالب XML و حتی رابطه های دیگر باشند.
- رابطه های پایه نبایستی حتما به صورت فیزیکی ذخیره سازی شوند.
- SQL یک زبان مجموعه ای Set-Oriented نیست. بلکه زبانی خورجینی bag-Oriented (مجموعه با احتمال تکرار اعضا) است.
اگر شما در زمینه بانک اطلاعاتی کار می کنید نمی توانید بدون دانستن مطالب این کتاب موفق باشید. دیت اصول اساسی را برای متخصصان –از جمله شما – به وضوح شرح داده است تا در کارتان موفق باشید. نگذارید کمبود تحصیلات در زمینه پایگاه داده ها شما را عقب بیاندازد. بلکه اجازه دهید توضیحات کریس دیت در مورد مفاهیم رابطه ای، نظریه مجموعه ها، تفاوت مدل و پیاده سازی، جبر رابطه ای، نرمال سازی و بسیاری مفاهیم دیگر، شما را به هنگام کار با پایگاه داده ها از دیگران متمایز کند.

سی جی دیت یکی از نخستین کسانی بود که نبوغ را در کلام ای اف کاد تشخیص داد. وی از ابتدا با کاد همراه شد و با وی از نزدیک در سالهای ایجاد مدل رابطه ای همکاری کرد و در ایجاد تکنولوژی پایگاه داده ها که مورد استفاده هر روزه ماست، نقش به سزایی داشت. این شانس شماست که می توانید از محضر چنین استادی استفاده کنید، آنرا از دست ندهید.

انتشارات مهرگان قلم :
تهران - خیابان انقلاب - خیابان دانشگاه - بین روانمهر و وحید نظری - کوچه آشتیانی - پلاک ۲۴ - واحد ۳
تلفن : ۶۶۹۶۸۹۷۰ - ۶۶۹۶۰۷۶۳
موسسه نقش سیمرغ :
تهران - خیابان انقلاب - خیابان ۱۲ فروردین - کوچه حقیقت - پلاک ۵۲
تلفن : ۶۶۴۹۴۵۹۸ - ۶۶۴۹۴۵۹۹

دانشگاه آزاد اسلامی
« واحد رودهن »

تاریخ:.....
شماره:.....
پیوست:.....

باسمه تعالی
" نماز آرام بخش قلب وروح است "

جناب آقای دکتر محمود باغبان طرقدری
دبیر محترم منطقه۸ دانشگاه آزاد اسلامی
سلام علیکم

احتراماً به استحضار می رساند که کتاب Database In Depth توسط آقای پویا فوده عضو هیأت علمی گروه کامپیوتر این واحد در دست ترجمه می باشد. خواهشمند است دستور فرمایید مراتب به منظور جلوگیری از هرگونه کار تکراری، به واحدهای تابعه اعلام گردد.

با تشکر فراوان
دکتر فتاح ناظم
معاون پژوهشی واحد رودهن

۸۵/۸۷۲۱
۸۵/۷/۲۴

رونوشت: آقای فوده

آدرس : رودهن ـ مجتمع دانشگاهی ـ صندوق پستی ۱۸۹
تلفن : ۹ ـ ۵۷۲۵۸۹۱ ـ ۰۲۲۱

نگاهی عمیق‌تر به پایگاه داده‌ها

نظریه رابطه‌ای برای متخصصین

نویسنده: سی جی دیت

انتشار ۲۰۰۵ (۱۳۸۴)

ناشر: اوریلی

مترجم : پویا فودهٔ

انتشار ترجمه ۱۳۸۷

ناشر: مهرگان قلم، نقش سیمرغ

انتشار نسخه الکترونیکی ۱۳۹۲

شابک: 978-600-90567-4-3

کتابشناسی ملی: 1285603

رده‌بندی کنگره: QAV6/9د9ن8 1378

تقدیم به یاد و خاطره
ای.اف. کـــاد

کسی که شیفته تجربه بدون پشتوانه تئوری است
مانند ناخدایی است که بدون سکان و یا قطب‌نما
به کشتی می‌رود و هیچ‌گاه نمی‌تواند اطمینان
داشته باشد که به کجا خواهد رسید.
تجربه همیشه بایستی بر پایه معلومات تئوری بنا شود.
لئوناردو داوینچی

مشکل مردم این نیست که چیزی نمی‌دانند
بلکه مشکل این است که آنچه می‌دانند، را نمی‌دانند.
جوش بیلینگز

یکی چاره آورد از دل بجای
که بد ژرف بین او به تدبیر و رای
فردوسی

فهرست

دیباچه ترجمه فارسی

فن‌آوری پایگاه داده‌ها یکی از مهمترین و با سابقه‌ترین تکنیک‌هایی است که به صورت واقعی و جدی در ایران مورد استفاده قرار گرفته است و به جرات می‌توان اعلام داشت که بسیاری از دانش‌آموختگان مهندسی کامپیوتر، علوم کامپیوتر و حتی مهندسی صنایع از آن برای ایجاد سیستم‌های کاربردی استفاده می‌نمایند.

اگر چه بسیاری از پایگاه‌های داده تولید شده از کیفیت و کارایی نسبتا خوبی برخوردارند و توانسته‌اند بر بخشی از مشکلات موجود در صنایع و سازمان‌های خدماتی فائق آیند، ولی غالب پایگاه‌های تولید شده به علت نیاز ناگهانی و نبود متخصص با تجربه، از مبانی علمی و اصولی برخوردار نیستند.

این کتاب نوشته سی‌جی دیت و ترجمه آقای مهندس فودهٔ برای افرادی که با مبانی پایگاه داده‌ها آشنایی داشته و حتی چندین سیستم پایگاه داده طراحی و پیاده‌سازی نموده‌اند بسیار مفید خواهد بود. این کتاب به آنها نگاه جدیدی از مبانی پایگاه داده القا می‌نماید و شرایطی را به وجود می‌آورد که علم و تجارب به دست آمده را به چالش کشیده و تصحیح یا تکمیل نمایند.

گرچه این کتاب ترجمه است و مترجم برای حفظ امانت قسمتی از مطالب را آنچنان که در زبان فارسی متداول است نگارش نموده است، ولیکن اینجانب با تجربه نزدیک به بیست سال تدریس، مشاوره و اجرای سیستم‌های پایگاه داده‌ها از مطالعه این کتاب اطلاعات خوبی به دست آوردم و لذت بردم.

دکتر محمد علی نعمت بخش
دانشگاه اصفهان – گروه کامپیوتر
خرداد ۱۳۸۷

پیش‌گفتار مترجم

امروزه و در کشور ما تمام کسانی که به نحوی با دانش نرم‌افزار و یا مدیریت نهادهای اداری و صنعتی سروکار دارند، اهمیت پایگاه داده‌ها را کاملا احساس می‌کنند. به ویژه فعالیت تقریبا تمامی افرادی که به برنامه نویسی مشغول‌اند، در زمینه پایگاه داده‌هاست. دلیل این وضعیت آن است که در شرایط فعلی و عدم وجود قانون کپی رایت برای نرم‌افزارهای خارجی و عدم اجرای دقیق آن برای نرم‌افزارهای ایرانی، تولید برنامه‌های کاربردی عمومی در عمل صرفه اقتصادی ندارد و تمام توان متخصصین نرم‌افزار به سمت تولید سیستم‌های نرم‌افزاری اختصاصی و نیمه اختصاصی برای سازمان‌های فعال در صنعت و تجارت متمرکز گردیده است. بدیهی است هسته مرکزی چنین سیستم‌هایی پایگاه داده‌های آنهاست.

سازنده پایگاه‌داده‌ها بایستی به سه چیز مجهز باشد. هنر طراحی، تسلط عملی بر یک یا چند DBMS و دانش تئوری طراحی پایگاه داده‌ها. اولی امری ذاتی و خدادادی است که افراد مختلف کم و بیش از آن بهره‌مندند. دومی با تجربه و تمرین و احتمالا شرکت در دوره‌های آموزشی حاصل می‌شود و اما سومین مورد که به معنای تسلط به نظریه رابطه‌ای و شامل مباحثی علمی و بنیادی است. مطالبی که عمدتا بایستی در دانشگاه آموخته شوند. متاسفانه -یا خوشبختانه!- فقط کشور ما نیست که در این زمینه دارای ضعف است. سی‌جی‌دیت، نویسنده کتاب و از پیش‌کسوتان نظریه رابطه‌ای در مقدمه به این نکته اشاره می‌کند که مباحث نظریه‌ای رابطه توسط اکثریت جامعه بانک اطلاعاتی به خوبی فراگرفته نشده است ولی تنها با تکیه بر مهارت‌های عملی نمی‌توان پایگاه‌های داده با کیفیت و کارایی بالا طراحی کرد.

هدف از ترجمه این کتاب افزایش دانش تئوری مورد نیاز کسانی است که قصد دارند به طور عملی در زمینه پایگاه داده‌ها کار کنند. بنابراین کتاب حاضر برای کسانی که در زمینه نرم‌افزار فارغ‌التحصیل شده و هم اکنون در این زمینه مشغول کار هستند و جهت یا درس پایگاه‌داده‌های دانشجویان کارشناسی نرم‌افزار و IT (پس از یادگیری مقدمات و زبان SQL) و یا به عنوان بخشی از مباحث کارشناسی ارشد نرم‌افزار، مفید است.

در پایان از تمام کسانی که مرا در انجام این کار یاری کردند تشکر می‌کنم خصوصا: همسرم که در تمام مراحل پشتیبان من بود و همچنین ویرایش نسخه نهایی را بر عهده داشت، آقای دکتر محمدعلی نعمت‌بخش که توصیه‌های مفیدی بر نسخه نهایی ارائه نمودند، دوستان عزیزم هادی صبوحی که ویرایش نسخه اولیه را بر عهده داشت و مهدی محمدی که درک بسیاری از مطالب کتاب مدیون تجربه‌های عملی است که حین کار دوشادوش با او به دست آورده‌ام.

پویا فودهٔ

رودهن، گروه کامپیوتر دانشگاه آزاد

فروردین ۱۳۸۷

پیش‌گفتار نشر آزاد

پنج سال از انتشار نسخه چاپی ترجمه کتاب گذشت. در این مدت کتاب در کتابفروشی‌های آکادمیک حاضر بود و من هم از نامه‌ها و تماس‌های تشویق‌آمیز و انتقادآمیز استادان و دوستان بهره‌مند. با گذشت این مدت تصور نمی‌کنم که دیگر تجدید چاپی در آینده این کتاب باشد؛ با این حال می‌دانم که مطالب آن هنوز کهنه نشده و برای بسیاری قابل استفاده است. از همین روی این کتاب را این با به صورت آزاد و بر روی اینترنت منتشر می‌نمایم و امیدوارم که سودمند باشد.

دانشگاه صنعتی مالزی

شهریور ۱۳۹۲

دیباچه

من زمانی که برای نخستین بار کتابی از کریس دیت نخریدم را به خوبی به خاطر می‌آورم، درست خواندید گفتم «نخریدم». اواخر سال 1991 یا اوایل 1992 بود که در کتابفروشی کوچکی در میشیگان مال به دنبال کتابی در مورد SQL می‌گشتم تا اینکه کتابی پیدا کردم که نوشته سی‌جی‌دیت بود. تا آن زمان نام وی را نشنیده بودم ولی کتاب خوبی به نظر می‌آمد. داشتم در مورد خرید آن تصمیم می‌گرفتم که ..... وای قیمت کتاب! کتابی که در دستم بود حدود نیم سانتی متر ضخامت داشت و حجم آن کمتر از 200 صفحه بود با این حال 30 دلار قیمت داشت. قدری با خودم کلنجار رفتم سپس کتاب را سرجایش گذاشتم و به راهم ادامه دادم.

چه اشتباهی! هیچ گرانی بی علت نیست و من در آن زمان متوجه این موضوع نبودم.

در نهایت رئیسم را متقاعد کردم که شرکت باید کتاب را برایم بخرد و این کار حدود یک ماه طول کشید ولی بالاخره کتاب بدستم رسید و آن را خواندم. نه یک بار که چندین بار. یکی از بهترین و آموزنده ترین بخش‌های کتاب (برای من) ضمائم آن بود که در آن انتقادات کریس از زبان SQL رایج در آن زمان مطرح شده بود.

بوسیله این کتاب چیزهای زیادی از کریس یاد گرفتم. من مجذوب کار با بانک‌های اطلاعاتی و زبانهای اعلانی مانند SQL شدم: می‌توانستم نتایجی که بدنبال آن بودم را توصیف کنم تا موتور بانک اطلاعاتی آنها را برای من بدست آورد. علاوه بر این بیشتر با کریس و نقش وی در همکاری با کاد در آغار عصر رابطه‌ای آشنا شدم. سالهای متوالی مشترک مجله طراحی و برنامه نویسی پایگاه داده‌ها بودم فقط به خاطر اینکه کریس یک ستون ثابت در آن داشت.

اطلاعاتی که از طریق کریس درباره SQL بدست آوردم – شروع آن بوسیله کتاب نازک و گران که نسبت به خرید آن بی میل بودم- از سالها قبل تاکنون در زندگی من اثرگذار بوده است. آشنایی من با بانک‌های اطلاعاتی در آن زمان نقطه عطف بزرگی در زندگی من بود. دقت و شیوایی قلم کریس اثر عمیقی در دید من نسبت به SQL و بطور کلی بانک‌های اطلاعاتی داشت. از آن زمان SQL همواره موضوع مورد علاقه من برای آموختن و تالیف بوده است.

من از این فکر که اگر کتاب کریس به دست من نیافتاده بود امروز در چه حالی بودم به لرزه می‌افتم. شما اشتباه مرا تکرار نکنید و این کتاب را به قفسه کتابفروشی برنگردانید. آنرا بخوانید! از کسی که در اختراع و تدوین مدل رابطه‌ای همکاری کرده است چیزهایی بیاموزید که آینده

کاری شما به آن بستگی دارد. ممکن است با همه آنچه کریس گفته است موافق نباشید –و نیازی هم نیست که باشید- ولی گفته‌های او را بفهمید. برای درک دید او از مدل رابطه‌ای وقت بگذارید. برای فهم نظرات منتشر شده از وی در مورد نحوه پیاده سازی‌های (بعضا غلط) این مدل در محصولات مختلف را درک کنید. این کارها را انجام دهید و خود را یک سر و گردن بالاتر از کسانی که برای یادگیری اصول بنیادی وقت نگذاشته‌اند ببینید.

اگر بخواهم کلامم را با ذکر نکته ای به پایان برم این نکته تاکید بر اهمیت شور و شوق برای آموختن است. برای من باعث افتخار بود که در کنار کریس دیت به ویرایش این کتاب پرداختم. من از صحبت با کریس و خواندن دست نوشته‌های وی چیزهای زیادی آموختم. ممکن است بخواهید بدانید که چگونه همکاری من با این کتاب شروع شد؟ تصور می‌کنم عامل این کار بطور غیر مستقیم مباحثی بود که با افراد هوشمند گروه اوراکل–ال در رابطه با بهینه سازی کوئری‌های SQL صورت می‌گرفت. ولیکن من تصور می‌کنم این همکاری ثمره مطالعه نوشته‌های کریس برای نخستین بار و در سالها پیش بود. حس کنجکاوی و عشق به آموختن مرا در کارم جدی کرد و از آن پس در مسیر پیشرفت قرار گرفتم و این عوامل می‌تواند در شما هم همین تاثیر را داشته باشد.

جاناتان جنیک
مانیسینگ، میشیگان
مارس ۲۰۰۵

پیش‌گفتار

پس از سال‌ها کار در بخش‌های مختلف جامعه بانک اطلاعاتی، احساس کردم که کتابی برای حرفه‌ای‌ها (و نه نوآموزها) مورد نیاز است که در آن اصول پایه نظریه رابطه‌ای – نه به گونه‌ای که توسط خصوصیات عجیب محصولات موجود، سمبل کاری‌های تجاری و استاندارد SQL لوث شده است- بیان شود. من این کتاب را برای پاسخگویی به این نیاز نوشتم. مخاطب اصلی من بانک اطلاعاتی کارها و یا افراد شاغل در این رشته هستند که به عدم درک تئوری زمینه تخصصی خود -آنطور که باید و شاید-، صادقانه اعتراف می‌کنند. درست است که این تئوری فقط مدل رابطه‌ای است و اصول بنیادی این نظریه ساده‌اند. ولی این حقیقت هم وجود دارد که این اصول بسیار گمراه‌کننده و سطحی‌نگرانه –و یا هردو حالت- آموزش داده شده‌اند به گونه‌ای که در بیشتر موارد چنین به نظر می‌رسد که اساسا درک نشده‌اند. برای مثال در اینجا چند سوال در این مورد مطرح شده است. شما جواب چند تا از آنها را می‌دانید؟

1- فرم اول نرمال دقیقا یعنی چه؟

2- رابطه‌ها و گزاره‌ها چه ارتباطی با همدیگر دارند؟

3- بهینه‌سازی معنایی چیست؟

4- وابستگی پیوندی چیست؟

5- نیم تفاضل چه اهمیتی دارد؟

6- چرا معلق‌کردن کنترل قواعد جامعیت قابل قبول نیست؟

7- متغیر رابطه‌ای چیست؟

8- تجزیه بدون کاستی یعنی چه؟

9- آیا یک رابطه می‌تواند دارای ویژگی‌هایی باشد که مقادیر آنها خود یک رابطه باشد؟

10- تفاوت بین مدل رابطه‌ای و SQL چیست؟

11- چرا «قانون اطلاعات» مهم است؟

12- چگونه XML با مدل رابطه‌ای مطابقت می‌کند؟

این کتاب پاسخگویی به سوالات فوق و بسیاری سوالات دیگر را امکان پذیر می‌کند و بطور کلی به بانک اطلاعاتی‌کارها را در درک عمیق تئوری رابطه‌ای و استفاده از آن در فعالیت‌های روزمره حرفه‌ای‌شان، یاری می‌نماید.

چه چیزی این کتاب را متمایز می‌کند؟

من در یک کلام می‌گویم که این کتاب ذاتا کتابی جدید است. من در کتاب‌های قبلی خود بارها گفته‌ام که فقط جستجویی در اطراف خود کرده و مطالبی را که نیاز به تکرارشان بوده است را گزینش نموده‌ام. ولی اکنون سعی کردم که آنها را به گونه‌ای متفاوت بیان کنم. ترتیبی متفاوت، به وجود آوردنی متفاوت، سبک و گفتاری متفاوت و مخاطبینی متفاوت (آخری از همه مهمتر است). ممکن است برخی از قسمتهای این کتاب در جاهای دیگری مطرح شده باشد ولی با این حال این کتاب بطور کلی کتابی جدید است. برخی قسمتهای کتاب به ناچار شبیه کتاب‌هایی است که قبلا نوشته‌ام چرا که همه آنها از یک منبع سرچشمه می‌گیرند: از مغز من و تجربیاتم و سمینارهایی که طی سالها برگزار کرده‌ام. این‌ها تقلید به حساب نمی‌آیند. به هر حال من آگاهانه از تعدادی از مثال‌های قدیمی دوباره استفاده کرده‌ام چرا که این مثال‌ها دقیقا همان نکاتی را که مایل به توضیح آنها هستم –نه کمتر و نه بیشتر– را نشان می‌دهند.

اجازه دهید که به موضوع مخاطبین این کتاب برگردیم. همانطور که اشاره شد من قبلا کتاب‌های زیادی در مورد تکنولوژی بانک‌های اطلاعاتی منتشر کرده‌ام. پس این کتاب چه تفاوتی با آنها دارد؟ آیا این کتاب با آنها رقابت می‌کند؟

به نظر من پاسخ سوال آخر«نه» است. من دو کتاب دیگر دارم که ممکن است در نگاه اول رقیب این کتاب به نظر برسند.

- آشنایی با پایگاه داده‌ها. ویرایش هشتم 2004
- پایگاه داده‌ها، انواع آن و مدل رابطه‌ای. ویرایش سوم 2006

کتاب اول عملا تمامی مباحث مربوط به بانک اطلاعاتی، و نه فقط مدل رابطه‌ای را پوشش می‌دهد. بدین معنا که در درجه اول کتابی دانشگاهی است و فرض را بر این می‌گذارد که خواننده هیچ آگاهی و تجربه‌ای در مورد بانک اطلاعاتی ندارد. همچنین متن آن رسمی‌تر از این کتاب است همانطور که باید برازنده یک کتاب درسی باشد.

کتاب دوم اجرای مجددی است از کتاب قبلی من و هیو داروین بعنوان «بنیانی برای آینده پایگاه داده‌ها» است. این کتاب متن درسی پیشرفته‌ای است که از کتاب قبلی رسمی‌تر است. با وجود اینکه این کتاب‌ها از نظر موضوع اشتراک‌هایی با همدیگر دارند من حقیقتا رقابتی بین این سه کتاب نمی‌بینم.

تفاوت مهم دیگر این کتاب جنبه خودآموز بودن آن است (البته می‌توان در مباحثات دانشگاهی هم از آن استفاده کرد). برای کمک به فهم مطالب تمرین‌های زیادی در کتاب وجود دارد. گرچه انجام آنها الزامی نیست با این حال من فکر می‌کنم که حل کردن –حداقل برخی از– آنها کار مفیدی است.

برای برطرف کردن شبهه رقابت بین کتاب‌هایم در مورد دو کتاب دیگر هم توضیح می‌دهم:

- بانک اطلاعاتی و مدل رابطه‌ای: بازبینی و تجزیه و تحلیل 2001
- داده‌های گذرا و مدل رابطه‌ای 2003

به نظر من کتاب اول مکمل این کتاب است و در آن مقاله‌های تـدکـاد –که طی آن مدل رابطه‌ای را برای نخستین بار به جهانیان معرفی کرد– را به گونه‌ای بی‌تکلف بازنگری کرده‌ام. در کتاب دوم –همانطور که از نام آن مشخص است– به نظریه رابطه‌ای نمی‌پردازد و فقط به یکی از شاخه‌های این نظریه مربوط می‌شود. با این حال ممکن است چنین به نظر برسد که فصل اول این کتاب که مروری کلی بر نظریه رابطه‌ای دارد، با این کتاب تا حدودی شباهت دارد ولیکن من واقعا چنین تصوری ندارم.

چکیده تمامی مطالب فوق این است که : با وجود اینکه من قبلا در مورد این موضوعات مطالب زیادی نوشته‌ام که برخی مطالب آنها به صورت اجتناب ناپذیری با هم مشابه بوده اند ولیکن تصور نمی‌کنم که هیچکدام از کتاب‌های دیگر من – و تا آنجا که می‌دانم هیچ کس دیگر– مطالب این کتاب را در یک جا و با این روش پوشش داده باشد.

مقدمات دیگر

من بایستی نکات دیگری را هم متذکر شوم. اولا همانطور که گفتم من از مثال‌های کتابها و مقالات قبلی خودم - خصوصا مثال اجرایی معروف بانک اطلاعاتی توزیع کنندگان و قطعات -در این کتاب هم استفاده کرده‌ام. من از اینکه این دایناسور را یک بار دیگر به دنبال خودم می‌کشم عذر می‌خواهم ولیکن توجه داشته باشید که من در ابتدا مثال‌ها را به دقت طراحی کرده‌ام تا دقیقا نکاتی را که قصد بیان آنها را دارم را روشن کنند.

ثانیا، درک من از مدل رابطه‌ای طی سال‌ها افزایش یافته است و این روند هم اکنون نیز ادامه دارد. این کتاب آخرین اندیشه‌های من در ارتباط با این موضوع را نشان می‌دهد بنابراین اگر بین این کتاب و مطالب قبلی من تناقضی مشاهده کردید (که البته میزان آن بسیار اندک است)، مطالب این کتاب جانشین کتاب‌های خواهد بود و به هر حال من تاکید میکنم چنین اختلاف‌هایی اکثرا بسیار جزیی هستند. من همیشه سعی کرده ام که –در صورت احساس نیاز- جزء اولین کسانی باشم که مطالب جدید را بیان می‌کنند.

ثالثا، من قصد دارم مطالب را بصورت تئوری بیان کنم. ولی به این جمله اعتقاد دارم که «تئوری همیشه به کار می‌آید». این نکته را به این علت بیان کردم که بسیاری از مردم با آن مخالف‌اند و تصور می‌کنند اگر چیزی تئوری شد، دیگر نمی‌تواند کاربردی باشد. ولی حقیقت اینست که تئوری –حداقل تئوری که من در موردش صحبت می‌کنم یعنی تئوری رابطه‌ای- بسیار کاربردی است. هدف این تئوری تنها این نیست که یادگرفته شود. بلکه هدف این است که ما را قادر به ساخت سیستم‌هایی صددرصد کاربردی نماید. ایجاد هر یک از قسمت‌های این تئوری دارای دلایل محکم کاربردی است. به راستی بخش عمده این تئوری نه تنها کاربردی است که بنیادی، سهل، ساده، سودمند و احتمالا لذت بخش هم هست. (امیدوارم این موضوع در هنگام مطالعه کتاب ثابت شود.)

(در حقیقت ما نباید به دنبال چیزی فراتر از تئوری رابطه‌ای باشیم. این نظر که تئوری کاربردی است آن‌قدر بدیهی است که نیازی به دفاع ندارد. برای نمونه یک صنعت میلیارد دلاری فقط می‌تواند بر پایه یک اندیشه تئوریک بنا شود. یک فرد بدبین ممکن است بگوید «بله، ولی تئوری تا به حال چه فایده‌ای برای من داشته است؟» می‌توان گفت که اگر اهمیت تئوری را دریابیم،

آنگاه می‌توانیم همواره خود را در مقابل به منتقدین برحق بدانیم. این هم دلیل دیگری است که بر اساس آن من فکر می‌کنم کتاب‌هایی از این دست مورد نیاز هستند.)

و نکته‌ای دیگر: زبان استاندارد «رابطه‌ای» SQL است و من فرض کرده‌ام که شما با این زبان و همچنین مفاهیم کلی در مورد بانک اطلاعاتی آشنا هستید. با این حال –همانطور که بزودی خواهید دید– من با صراحت از SQL انتقاد کرده‌ام. حقیقت تاسف بار این است که SQL در بسیاری موارد نمی‌تواند به خوبی از مدل رابطه‌ای پشتیبانی کند. این زبان از نظر ناقص بودن دستورات و نحوه اجرای آنها دارای مشکلات بسیاری است.(برای تاکید بر این موضوع، «رابطه‌ای» را داخل کوتیشن قرار دادم.) این فقط یکی از دلایلی است که برای یادگیری مدل رابطه‌ای وجود دارد. چرا که پشتیبانی SQL از این مدل بسیار ناقص و نارسا است. این زبان مانند طنابی در دست شماست که می‌توانید با آن خود را دار بزنید. پس برای اینکه چنین اتفاقی برایتان نیافتد نیاز به تئوری دارید. شما نیازمند تئوری هستید تا خود را مقید به مقرراتی کنید که بایستی SQL شما را وادار به رعایت آنها می‌کرد، ولی این کار را نکرده است. تکرار سطرها در این مورد مثال خوبی است. این کار در مدل رابطه‌ای غیر مجاز است ولی SQL آن را مجاز می‌داند. شما باید بدانید چرا این کار در تئوری ممکن نیست تا بدانید چرا نباید از این «ویژگی» موجود در SQL «بهره‌برداری» کنید. استفان فارولت بعنوان خواننده و منتقد این کتاب چنین نوشت: «اگر به اندازه ذره‌ای تجربه داشته باشید، به خوبی درک می‌کنید که هیچ راهی برای فرار از تئوری وجود ندارد.»

و یک نکته دیگر در مورد SQL : در هر کجای این کتاب که من این کلمه را به کار برده‌ام منظورم فقط SQL استاندارد است و نه لهجه‌های مختلف این زبان که هر یک متعلق محصولی‌خاص می‌باشد (جز آن که بطور صریح خلاف آن عنوان شده باشد) و خصوصا زمانی که کلمه «اس–کیو–ال» را به کار می‌برم منظورم «sequel» نیست.

در پایان توجه شما را به کنفرانس‌هایی که در مورد مطالب بیان شده در این کتاب ارائه نموده‌ام جلب می‌کنم. برای اطلاعات بیشتر به آدرس http://www.dbdebunk.com و یا http://www.thethirdmanifesto.com مراجعه نمایید.

پاسخ به سوالات

شما می‌توانید برای عضویت در لیست پستی ما و یا درخواست کاتالوگ با ما به نشانی زیر مکاتبه کنید:

mailto:info@oreilly.com

همچنین برای طرح سوالات فنی و یا اظهار نظر در مورد مطالب کتاب به آدرس زیر نامه بنویسید:

mailto:bookquestions@oreilly.com

ما وب سایتی برای این کتاب داریم که می‌توانید مثال‌های بیشتر و غلط نامه کتاب را در آن مشاهده کنید. آدرس این سایت عبارت است از:

http://www.oreilly.com/catalog/databaseid

قدردانی

در اینجا از تمام کسانی که بطور مستقیم و غیر مستقیم در بوجود آمدن این کتاب دخالت داشتند تشکر می‌کنم. در اینجا نام بازبینان این کتاب را ذکر می‌کنم. استفان فارولت، جاناتن جنیک، لکس‌هان، آناتونی‌مولینارو، پیترابسون و میشل‌ونر. که شرح‌های سودمندی برای دست‌نوشته‌های من نوشتند. همچنین از ناگراج آلور و هیو داروین بخاطر مباحثات فنی متعددی که به من داشتند تشکر می‌کنم. همچنین از همسرم لیندی بخاطر حمایت و پشتیبانی‌اش در هنگام انجام این کار –و همچنین تمامی پروژه‌های بانک اطلاعاتی طی سالیان متوالی– قدردانی می‌کنم. از تمامی افراد انتشارات اوریلی خصوصا جاناتان جنیک و ژنویو انترمونت بخاطر دلگرمی و همیاری و پشتیبانی‌شان در تمامی زمان نوشتن این کتاب سپاسگزارم.

سی جی دیت

هلدبورگ، کالیفرنیا، ۲۰۰۵

فصل اول

مقدمه

متخصصین – با هر انضباط و تفکری – بایستی اصول رشته کاری خود را بدانند. بنابراین اگر شما یک متخصص بانک اطلاعاتی هستید باید مدل رابطه‌ای را بشناسید چرا که این مدل اساس (و یا بخش عمده) پایگاه داده است. امروزه مدل رابطه‌ای در هر دوره دانشگاهی و یا تجاری پایگاه‌داده‌ها –حداقل بصورت سطحی- تدریس می‌شود ولی این کار در اکثر موارد این امر به خوبی انجام نشده است و نتیجه مطلوبی از این تدریس حاصل نمی‌شود. متاسفانه مدل رابطه‌ای توسط عموم افرادی که با بانک اطلاعاتی سر و کار دارند، به خوبی درک نشده است. برخی دلایل این امر عبارتند از:

- فهمیدن این مدل مانند یادگیری در خلاء است. درک رابطه بین اشیا – حداقل برای تازه کارها – و همچنین شناسایی مسائل و راه حل آنها، مشکل است.
- خود مدرسین نیز بطور کامل این مفاهیم را درک نکرده‌اند.
- اکثرا در افراد تجربی- این مدل اصلا تدریس نشده است و بجای آن زبان SQL و یا برخی لهجه‌های آن مانند لهجه اوراکل تدریس شده است.

بنابراین کتاب حاضر برای بانک اطلاعاتی کارهای حرفه‌ای و خصوصا افراد تجربی است، که با این مدل آشنایی دارند ولی دانش آنها کمتر از چیزی است که باید باشد. در این شرایط این کتاب برای شروع کار کسانی که هیچ چیز از بانک اطلاعاتی نمی‌دانند مناسب نیست. من می‌دانم که شما چیزهایی در مورد SQL می‌دانید. ولی اگر – اگر در اینجا لحن صحبتم کمی تند است ببخشید – اگر آگاهی شما از مدل رابطه‌ای تماما از SQL ناشی شده است، متاسفانه شما مدل رابطه‌ای را آنطور که باید و شاید نمی‌شناسید و در زمره افرادی که «به اندازه کافی نمی‌دانند» قرار دارید.

توجه

SQL ≠ مدل رابطه‌ای

در اینجا برخی از مواردی که SQL به وضوح آنها را روشن نکرده است را بیان کرده ام (و تعداد بیشماری از این موارد وجود دارد):

- پایگاه داده‌ها، رابطه‌ها و تاپل‌ها واقعا چه هستند
- تفاوت بین نوع‌ها و رابطه‌ها
- تفاوت بین مقادیر رابطه‌ای و متغیرهای رابطه‌ای
- رابطه بین گزاره‌ها و استنتاج‌ها
- صحت ویژگی‌های رابطه‌ای
- نقش حیاتی قیود جامعیت

تمامی موارد فوق و بسیاری موارد دیگر در این کتاب عنوان شده اند.

دوباره تاکید می‌کنم: اگر آگاهی شما از مدل رابطه‌ای فقط از SQL ناشی شده است، در زمره افرادی که «به اندازه کافی نمی‌دانند» قرار دارید. یکی از پیامدهای ناپسند این امر این است که شما مجبورید برخی مطالب را از نو بیاموزید و به فراموشی سپردن و بازآموزی مطالبی که قبلا به اشتباه یاد گرفته‌اید، کاری دشوار است. یک نکته دیگر... خواهش می‌کنم به این دلیل که تصور می‌کنید به طور کامل با مبحثی آشنایی دارید، از مطالعه آن صرفنظر نکنید. برای مثال مطمئن هستید که می‌دانید که کلید چیست؛ و عبارات رابطه‌ای؛ و یا پیوند.

درباره اصطلاحات فنی

با دیدن لیست مباحثی رابطه‌ای که در قسمت قبل مطرح کردن احتمالا بلافاصله متوجه شده‌اید که از کلمات رسمی مانند رابطه، تاپل* و ویژگی استفاده کرده‌ام. SQL از اینگونه عبارات استفاده نمی‌کند و بجای آن عبارات دوستانه‌تری مانند جدول، سطر و ستون را به کار می‌برد. من به طور کل با این نظر که عبارات دوستانه، مطالب را پذیرفتنی‌تر می‌کند موافقم. ولی در این مورد به نظر می‌رسد که این عبارات ساده مطلب را مطبوع‌تر نمی‌کند بلکه آنرا تحریف می‌کنند و در واقع باعث نفهمیدن اصل مطلب می‌شوند. واقعیت این است که رابطه همان جدول نیست، تاپل سطر نیست و ویژگی هم ستون نیست. و فقط می‌توانند در متون غیر رسمی خود را بجای آنها جا بزنند

* بر وزن قابل

– خود من هم در بسیاری کتاب‌ها این کار را کرده ام– می‌خواهم بگویم که عبارات دوستانه فقط به واقعیت شباهت دارند و از بیان ذات آنچه به دنبال آن هستید، ناتوان‌اند. به زبان دیگر: اگر شما کیفیت چیزی را به درستی فهمیده باشید، آنگاه مجاز خواهید بود که برای نامیدن آن از عبارات ساده استفاده کنید ولی برای موضوعی که برای نخستین بار قصد درگیر شدن با آنرا دارید، واقعا لازم است که از اصطلاحات رسمی شروع کنید. بنابراین در این کتاب در اکثر موارد من از اصطلاحات رسمی استفاده کرده‌ام و البته در موقع لزوم آنها را بطور کامل توضیح داده‌ام (البته نه در فصل اول کتاب).

و نکته‌ای دیگر در مورد اصطلاحات فنی: گفته شد که SQL سعی دارد که مجموعه‌ای از عبارات را ساده‌تر کند. و در اینجا باید بگویم که در نهایت کار را پیچیده‌تر می‌کند. برای نمونه کلمات عملگر (اپراتور)، تابع، پروسیجر، روتین و متد همگی در اصل به یک موضوع اشاره می‌کنند (با تفاوت‌های بسیار جزئی) و من در این کتاب برای همگی از کلمه عملگر استفاده کرده‌ام.

و نکته‌ای دیگر در مورد SQL ؛ منظورم از این کلمه فقط SQL استاندارد است و نه لهجه‌های مختلف این زبان که هر یک متعلق به محصول خاصی می‌باشد بجز مواردی که بطور صریح خلاف این مطلب را عنوان کرده‌ام.(این نکته را در پیشگفتار این کتاب هم آورده ام ولی می‌دانم که اکثر مردم پیشگفتار را نمی‌خوانند). انتقادات من هم به همان SQL استاندارد برمی‌گردد. بنابراین اگر انتقادی مطرح گردیده به محصول پر طرفدار شما که خوشبختانه شنیده‌ام بسیار خوب و عالی است مربوط نمی‌شود.

قوانین مهم‌ند، نه محصولات

سوال مهمی که ممکن است از من بپرسید این است که چرا مدعی هستم که شما به عنوان متخصص بانک اطلاعاتی بایستی با مدل رابطه‌ای آشنایی داشته باشید. پاسخ این است که مدل رابطه‌ای وابسته به محصول نیست بلکه وابسته به قوانین است و اگر بخواهید معنی قواعد را بدانید به شرح زیر است (به نقل از لغت نامه چیمبر قرن بیستم):

قانون: ماخذ، ریشه، اصل، اصل بنیادی، اصل طبیعی، پایه تئوری، واقعیت بنیادی که دیگران آنرا کشف کرده‌اند و یا بدیهی باشد.

نکته این است که قوانین پابرجا هستند و محصولات و فن‌آوری‌ها همواره در حال تغییرند (و SQL هم از این امر مستثنا نیست). برای مثال فرض کنید که شما اوراکل را می‌شناسید و یا اصلا در اوراکل خبره هستید. با این حال دانش شما به اوراکل محدود می‌شود و قابل انتقال به محیط DB2 و یا SQLServer نیست (و ممکن است مانع پیشرفت شما در شناخت محیط جدید هم باشد)؛ ولی اگر شما قوانین اساسی –و به زبان دیگر مدل رابطه‌ای– را بدانید، دانش و مهارتی دارید که قابل انتقال است: چنین دانش و مهارتی شما را قادر به کار در هر محیطی می‌کند و هرگز قدیمی نمی‌شود.

بنابراین در کتاب حاضر قوانین اهمیت دارند و نه محصولات. اصول بنیادی موضوع بحث است و نه مدهای زودگذر. البته من می‌دانم که در دنیای واقعی گاه شما مجبور به مصالحه هستید. مثلا برخی اوقات ممکن است دلایلی عملی برای طراحی بانک اطلاعاتی بصورت غیر بهینه وجود داشته باشد. (این موضوع در فصل هفتم مورد بحث قرار گرفته است.) در مورد SQL هم می‌توان چنین شرایطی را در نظر داشت. گر چه استفاده از SQL بصورت رابطه‌ای قطعا امکان پذیر است (در برخی اجزا و به هر قیمتی)، ولی در برخی موارد به این نتیجه می‌رسید که –به این دلیل که پیاده‌سازی موجود بسیار ناقص است– چنین کاری بهره‌وری را پایین خواهد آورد. در چنین مواردی ممکن است برخی قسمت‌ها را از «کاملا رابطه‌ای» بودن معاف کنید (مثلا با نوشتن یک کوئری عجیب و غیر طبیعی برای پیاده‌سازی استفاده از یک ایندکس). به هر صورت من اعتقاد راسخ دارم که شما باید چنین مصالحه‌ای را از منظر قوانین به انجام رسانید.

- شما باید بدانید که در زمان مصالحه، دارید چه کاری انجام می‌دهید.
- شما باید رفتار صحیح و قانونی بشناسید و دلیل خوبی برای ترک آن داشته باشید.
- شما باید دلایل خود را مستند کنید، چرا که اگر در آینده قصد بهم زدن این مصالحه را داشته باشید – مثلا اگر محصول بعدی در همین رابطه ابزارهای بیشتری داشت– این کار انجام‌پذیر باشد.

جمله زیر –که منسوب به لئوناردو داوینچی است و پانصد سال قبل بیان شده!– فوت و فن دریا نوردی را بیان می‌کند:

کسی که شیفته انجام تجربه بدون پشتوانه تئوری است مانند ناخدایی است که بدون سکان یا قطب‌نما به کشتی می‌رود و هیچ‌گاه نمی‌تواند اطمینان داشته باشد که به کجا می‌رسد. تجربه بایستی همیشه بر پایه معلومات تئوری بنا شود.

مروری بر مدل اصلی

شما در زمینه بانک اطلاعاتی کار می‌کنید، بنابراین تا حدودی با مدل رابطه‌ای آشنایی دارید. هدف از این بخش شروع صحبت در مورد مباحث آتی است و مروری بر برخی از جنبه‌های ابتدایی این مدل –که بصورت اصلی تدوین شده است- خواهیم داشت. به عبارت «بصورت اصلی تدوین شده است» توجه کنید! یک تصور غلط در مورد مدل رابطه‌ای اینست که آنرا کاملا ایستا فرض می‌کنند در حالی که چنین نیست. مدل رابطه‌ای از این نظر شبیه ریاضیات است: ریاضیات هم ایستا نیست و در گذر زمان تغییر پیدا می‌کند. در حقیقت مدل رابطه‌ای می‌تواند بعنوان شاخه کوچکی از ریاضیات در نظر گرفته شود. در طی زمان همواره قضایای جدید اثبات و نتایج جدید کشف می‌شوند و همیشه افراد شایسته در این روند مشارکت دارند. و باز هم مشابه ریاضیات، مدل رابطه‌ای نیز توسط فردی ابداع شد که از زمره مردان کوشای این جهان بود.

شاید نام وی را ندانید. باید بگویم که او ای‌اف کـاد [*] نام داشت و در آن زمان یکی از محققین موسسه IBM بود. در اواخر سال 1968 کاد ریاضیدان تجربی برای نخستین بار دریافت که قواعد ریاضی می‌توانند در تدوین قوانین مربوط به پایگاه داده‌ها –که در آن زمان بسیار ناقص و ابتدایی بود- به‌کار گرفته شوند. تدوین اصلی این مدل در گزارش پژوهشی IBM در سال 1969 نمودار گشت و من مطالب بیشتری در مورد این مقاله در ضمیمه ب عنوان آورده‌ام.

ویژگی‌های ساختاری

مدل اصلی دارای سه بخش مهم است. ساختار، جامعیت و کار با داده‌ها. من بطور خلاصه هریک از آنهارا بصورت اجمالی توضیح می‌دهم و در فصل‌های آینده بطور کامل در مورد آنها بحث خواهم کرد.

[*] ای برای ادگار و اف برای فرانک، ولی او همیشه با نام مستعار امضا می‌کرد. برای دوستانش و از جمله خود من او تد بود.

اول ساختار. اصلی ترین ساختار خود رابطه است و همانطور که می‌دانید رابطه‌ها را بصورت جدول بر روی کاغذ نمایش می‌دهند (شکل 1-1 را بعنوان یک مثال بدون شرح ببینید). رابطه‌ها بر روی انواع تعریف می‌شوند (که بعنوان دامنه هم شناخته می‌شوند). یک نوع، استخری خیالی از مقادیر مختلف است که ویژگی‌های واقعی در رابطه‌های واقعی، مقادیر واقعی خود را از آن برمی‌دارند. با مراجعه به مثال بانک اطلاعاتی کارمندان و شعب که در شکل 1-1 نمایش داده شده است، مثلا نوعی که DNO («کد شعبه») نام دارد مجموعه‌ای از تمامی کدهای معتبر برای شعب می‌باشد. ویژگی DNO در رابطه DEPT و همچنین ویژگی DNO در رابطه EMP می‌توانند شامل مقادیری باشند که از استخری خیالی برداشته شده‌اند.(ولی لازم نیست نام این ویژگی‌ها و نام نوع‌شان با هم یکی باشد و معمولا هم چنین نیست. در آینده مثال‌های زیادی در این مورد خواهیم دید.)

DEPT

DNO	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

ENO	ENAME	DNO	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

DEPT.DNO *referenced by* EMP.DNO

شکل 1-1. بانک اطلاعاتی شعب و کارمندان – مقادیر نمونه

همان طور که قبلا گفتم، جدول‌هایی مانند دو جدول شکل 1-1 رابطه‌ها را به تصویر می‌کشند. یک رابطه n تایی می‌تواند با جدولی n ستونی نمایش داده شود. ستون‌های جدول با ویژگی‌ها رابطه و همچنین سطرها با تاپل‌ها معادل‌اند. n می‌تواند یک مقدار صحیح و غیر منفی باشد. یک رابطه یکتایی یونری، رابطه دوتایی باینری و رابطه سه تایی ترنری نامیده می‌شوند. مدل رابطه‌ای انواع گوناگون کلید را پشتیبانی می‌کند. هر رابطه دارای حداقل یک کلید کاندید است.* کلید کاندید فقط یک شناسه یکتا است. به زبان دیگر ترکیبی است از ویژگی‌ها -اکثرا و

* به زبان دقیق این جمله باید به این صورت گفته شود: «هر مرابطه دارای حداقل یک کلید کاندید است» (قسمت فرق رابطه‌ها و مرابطه‌ها را ببینید).

نه همیشه، این «ترکیب» فقط یک ویژگی را در برمی‌گیرد- که هر تاپل رابطه در ترکیب فوق الذکر دارای مقداری یکتا است. در شکل 1-1 برای مثال هر شعبه دارای یک کد یکتا و هر کارمند دارای یک شماره پرسنلی یکتا است. بنابراین می‌توانیم بگوییم که {DNO} یک کلید کاندید برای DEPT و {ENO} یک کلید کاندید برای EMP است. در مورد آکولاد: کلیدهای کاندید همیشه ترکیب‌ها، یا زیر مجموعه‌هایی از ویژگی‌ها هستند-حتی زمانی که فقط یک ویژگی را شامل شود- و بصورت عرفی اعضای یک مجموعه را بین آکولاد قرار می‌دهند.

و موضوع بعد کلید اصلی، کلید کاندیدی است که به دلیل دارا بودن برخی مشخصات خاص، برای این منظور انتخاب شده است. بدیهی است که اگر رابطه فقط یک کلید کاندید داشته باشد، می‌توان به راحتی آنرا کلید اصلی نامید. ولی اگر رابطه دارای دو -ویا بیشتر- کلید کاندید باشد برای انتخاب یکی از آنها بعنوان کلید اصلی مختار خواهیم بود. به این معنا که بالاخره یکی از آنها «از بقیه برابرتر است». برای مثال فرض کنید که کارمندان اداره دارای یک شماره کارمندی یکتا و یک نام یکتا هستند. در این صورت {ENO} و {ENAME} هر دو کلید کاندید EMP خواهند بود و ممکن است که ما {EMP} را بعنوان کلید اصلی تعیین کنیم.

توجه کنید که گفتم در انتخاب کلید اصلی آزاد هستید ولی اگر فقط یک کلید کاندید موجود باشد دیگر نه اختیاری وجود دارد و نه مشکلی. اما اگر بیش از یک کلید کاندید وجود داشته باشد، انتخاب از بین آنها تا حدودی حالت دل‌بخواه دارد. در بعضی موقعیت‌ها به نظر می‌رسد که هیچ دلیل منطقی برای انتخاب وجود ندارد. در این کتاب من معمولا از قاعده کلید اصلی پیروی می‌کنم. در تصاویری مانند شکل 1-1 من ویژگی کلید اصلی را با دو خط در زیر آن متمایز می‌کنم. بر این نکته تاکید می‌کنم که کلید کاندید همان کلید اصلی نیست و این موضوع از نقطه نظر رابطه‌ای اهمیت فراوان دارد. از این به بعد در برخی موارد من از کلمه کلید استفاده کرده‌ام که منظور از آن کلید کاندید است (در مواردی ممکن است متحیر شوید که کلید اصلی از کدام «مشخصات خاص» بهره‌مند بوده که در سایر کلیدهای کاندید نبوده است؟ این مشخصه معمولا به ذات ویژگی انتخاب شده باز می‌گردد. بگذریم، این موضوع چندان مهم نیست.)

و در نهایت کلید خارجی، مجموعه‌ای از ویژگی‌های یک رابطه است که لازم است با مقادیر کلید کاندید یک رابطه دیگر (و یا شاید هم همان رابطه) مطابقت داشته باشد. با مراجعه به شکل

1-1 مثلا {DNO} کلید خارجی EMP است و لازم است مقادیر آن با کلید کاندید {DNO} در رابطه DEPT مطابقت داشته باشد. (برای مشخص شدن این موضوع، آنها را در شکل با فلش به هم مرتبط کرده‌ام). منظور از لزوم مطابقت این است که اگر مثلا EMP دارای تاپلی با مقدار D2 برای DNO باشد آنگاه DEPT هم می‌تواند دارای تاپلی باشد که مقدار DNO آن D2 است در غیر اینصورت EMP دارای کارمندی خواهد بود که وجود خارجی ندارد و بانک اطلاعاتی هم «نمونه قابل اعتمادی از واقعیت» نخواهد بود.

ویژگی جامعیت

قید جامعیت در اصل فقط عبارتی منطقی است که باید همیشه درست باشد. مثلا در مورد شعب و کارمندان می‌توان یک قید عملی در مورد SALARY داشت بدین صورت که مقدار آن همیشه بزرگتر از صفر باشد. هر بانک اطلاعاتی دارای قیودی است ولی این قیود لزوما با توجه به رابطه‌های یک بانک اطلاعاتی خاص تدوین شده و ویژه همان بانک اطلاعاتی خواهند بود. در مقابل مدل رابطه‌ای (بصورت سنتی) دارای دو قانون جامعیت عمومی است. عمومی بدین معنا که بر هر بانک اطلاعاتی اعمال می‌شود. یکی از آنها در مورد کلید اصلی است و دیگری در مورد کلید خارجی.

جامعیت وجودی

کلید اصلی نمی‌تواند تهی باشد.

جامعیت ارجاعی

نبایستی هیچ مقدار غیر منطبقی در کلید خارجی وجود داشته باشد.

اجازه دهید تا اول دومی را توضیح دهم: منظورم از عبارت «کلید خارجی غیر منطبق» ، مقدار کلید خارجی است که مقدار معادل آن در کلید کاندید متناظر وجود نداشته باشد. در مثال شعب و کارمندان، اگر EMP دارای تاپلی باشد که مقدار DNO آن D2 است ولی تاپل متناظر آن در DEPT وجود نداشته باشد، جامعیت ارجاعی نقض شده است. بنابراین قانون جامعیت ارجاعی

فقط بر پیاده‌سازی صحیح کلید خارجی تاکید می‌کند و عنوان جامعیت ارجاعی از این واقعیت ناشی می‌شود که هر مقدار کلید خارجی بعنوان مرجع به تاپلی با مقدار مساوی در کلید کاندید متناظر وابسته است. در واقع این قانون می‌گوید:«اگر B به A ارجاع داشته باشد، آنگاه A باید حتما وجود داشته باشد.»

در مورد قانون جامعیت وجودی، در اینجا من اشکال دارم. من موضوع «تهی» را بطور کلی رد می‌کنم و شدیدا معتقدم که تهی جایی در مدل رابطه‌ای ندارد (کاد خلاف این نظر را داشت اما من برای این حرف دلایل محکمی دارم). برای تشریح قاعده جامعیت وجودی من مجبورم نظر خود را موقتا کنار بگذارم ولی خواهشمندم مطالبی را که در این مورد در فصل سوم بیان کرده ام را درک کنید.

«نشانگر» تهی ذاتا به معنای مقدار نامعلوم است (نکته بسیار مهم: تهی یک مقدار نیست بلکه یک نشانگر و یا پرچم است.) برای مثال فرض کنید که میزان حقوق کارمند E2 را نمی‌دانیم. بنابراین بجای وارد کردن یک مقدار SALARY واقعی در تاپل مربوط به کارمند مذکور -که نمی‌توانیم این کار را انجام دهیم چون مقدار واقعی حقوق را نمی‌دانیم- محل مربوط به SALARY را با تهی علامت‌گذاری می‌کنم.

ENO	ENAME	DNO	SALARY
E2	Cheng	D1	

همانطور که می‌بینید تاپل در محل SALARY هیچ مقداری ندارد. در این کتاب از هاشور برای نمایش مکان‌های خالی استفاده کرده‌ام و شما می‌توانید مکان‌های هاشور خورده را دارای پرچم و یا نشانگر تهی در نظر بگیرید.

قانون جامعیت وجودی از نقطه نظر رابطه EMP تقریبا چنین است که هر کارمند می‌تواند نام، شعبه محل خدمت و حقوق نامعلوم داشته باشد اما شماره کارمندی وی باید حتما معلوم باشد چرا که در غیر این صورت مشخص نیست که ما در مورد کدام کارمند (و یا کدام «موجودیت») صحبت می‌کنیم.

این تمام مطالبی بود که می‌خواستم اینجا درباره تهی بیان کنم. این موضوع را تا فصل سوم فراموش کنید.

ویژگی کار با داده‌ها

بخش کار با داده‌ها در مدل رابطه‌ای شامل قسمت‌های زیر است:

- مجموعه‌ای از عملگرهای رابطه‌ای مانند تفاضل (یا MINUS) است که تمامی آنها در مجموع جبر رابطه‌ای نامیده می‌شوند.
- عملگر انتساب رابطه‌ای که یک عبارت رابطه‌ای مانند r MINUS s (r و s رابطه هستند) را به یک رابطه دیگر منتسب می‌کند.

عملگر رابطه‌ای انتساب چگونگی به‌روزرسانی* را در مدل رابطه‌ای تشریح می‌کند و در بخش «رابطه‌ها، و یا رلورها» بیشتر به این موضوع خواهم پرداخت. و در مورد جبر رابطه‌ای،(تقریبا) مجموعه‌ای از عملگرهای رابطه‌ای است که رابطه‌های «جدید» را از روی رابطه‌های «قدیمی» بدست می‌آورد. و به عبارت دقیق‌تر هر عملگر حداقل یک رابطه را بعنوان ورودی دریافت کرده، رابطه‌ای دیگر را بعنوان خروجی تولید می‌کند. مثلا تفاضل (و یا MINUS) دو رابطه را بعنوان ورودی دریافت کرده، یکی را از دیگری «تفریق» نموده و رابطه دیگری را بعنوان خروجی بوجود می‌آورد. و این خیلی مهم است که رابطه خروجی یک رابطه دیگر است. ویژگی معروف تراگذری (تعدی) در جبر رابطه‌ای وجود دارد و باعث می‌شود که بتوان عبارات تو در تو نوشت. تا زمانی که خروجی یک عملگر با نوع ورودی مورد نیاز عملگر بعدی مطابقت داشته باشد،خروجی یک عملگر می‌تواند بعنوان ورودی عملگر دیگر مورد استفاده قرار گیرد. برای مثال می‌توان تفاضل دو رابطه r و s را بدست آورد و نتیجه را بعنوان ورودی اجتماع با u در نظر گرفت و مجددا نتیجه حاصل را بعنوان یکی از دو ورودی عملگر اشتراک با v مورد استفاده قرار داد و به همین ترتیب ...

*در این کتاب به‌روزرسانی بعنوان کلمه‌ای عمومی برای عملگرهای INSERT، DELETE و UPDATE به کار برده شده است و زمانی که منظور دقیقا UPDATE باشد آنرا با حروف بزرگ انگلیسی خواهم نوشت.

می‌توان تعدادی بیشماری عملگر تعریف کرد که همگی در تعریف «حداقل یک رابطه بعنوان ورودی و دقیقا یک رابطه خروجی» صدق می‌کنند. اما وقتی که از عملگر صحبت می‌کنیم معمولا منظور هشت عملگر سنتی است (که کاد در اولین مقاله‌اش آنها را معرفی کرد). در فصل پنجم تعدادی از عملگرهای اضافی را را معرفی و در موردشان بحث خواهم کرد. شکل 1-2 معرفی تصویر این هشت عملگر می‌باشد. توجه: اگر با برخی از این عملگرها ناآشنا هستید - خصوصا تقسیم!- و فهم این توضیحات مختصر برایتان مشکل است، نگران نباشید. من قصد دارم در آینده مجددا با مثال و توضیحات بیشتر به آنها بپردازم (عمدتا در فصل پنجم).

گزینش

تمامی تاپل‌هایی از یک رابطه که دارای شرایطی خاص باشند را برمی‌گرداند. مثلا ممکن است رابطه EMP را گزینش کنیم تا تاپل‌هایی که مقدار EMP آنها D2 است را بدست آوریم.

پرتو

رابطه‌ای را برمی‌گرداند که حاوی تمام (زیر)تاپل‌های یک رابطه، بعد از حذف برخی ویژگی‌های خاص می‌باشد. برای مثال ممکن است رابطه EMP را فقط بر روی ویژگی‌های ENO و SALARY پرتو کنیم.

ضرب

رابطه‌ای را برمی‌گرداند که حاوی تمامی تاپل‌های ممکن از ترکیب دو تاپل است که هر یک از آنها متعلق به یکی از دو رابطه مشخص است. ضرب به نام‌های ضرب دکارتی، عمل ضربدری، پیوند ضربدری و پیوند دکارتی شناخته می‌شود (در حقیقت این عمل نوع خاصی از پیوند است، که در فصل پنجم خواهید دید).

اشتراک

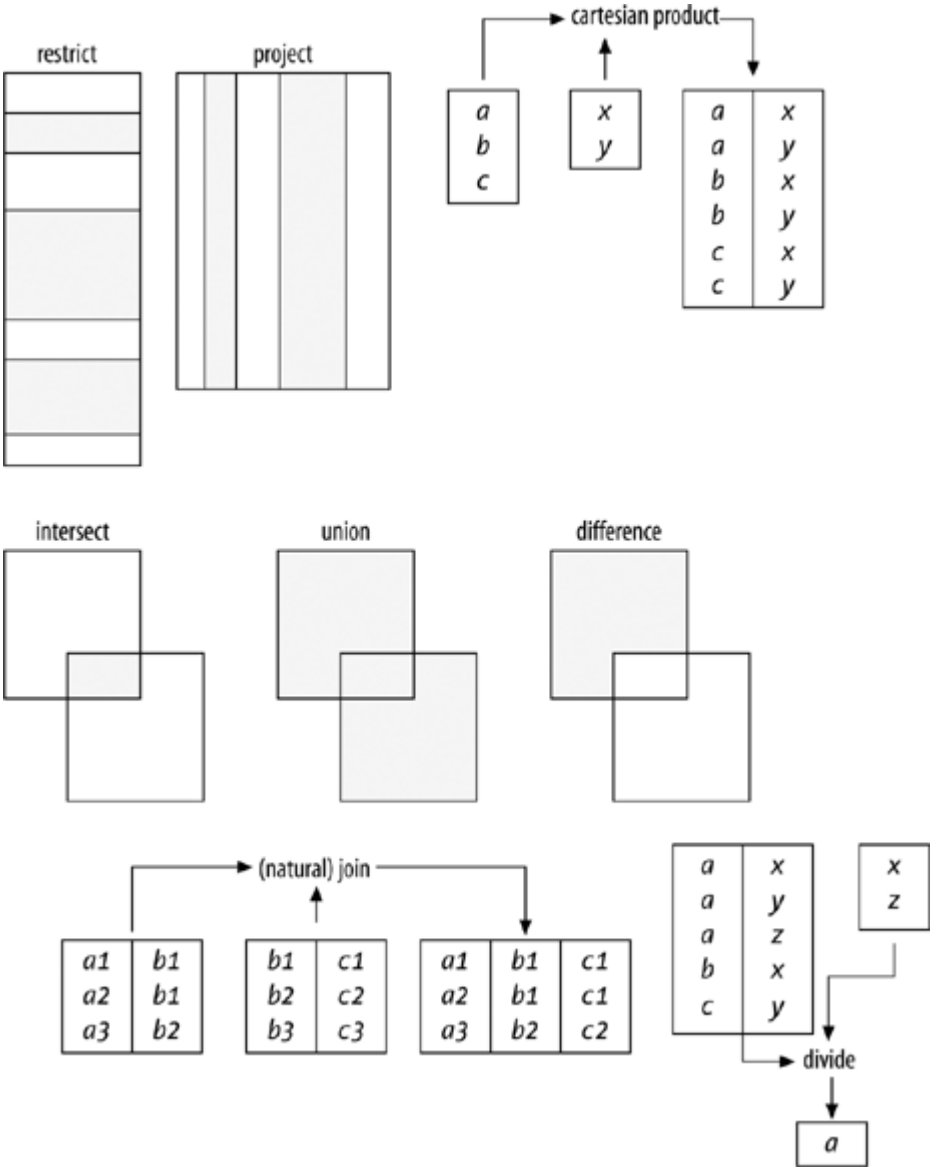
رابطه‌ای را برمی‌گرداند که حاوی تاپل‌های موجود در هر دو رابطه است (در واقع اشتراک هم نوع خاصی از پیوند است) .

اجتماع

رابطه‌ای را برمی‌گرداند که حاوی تمامی تاپل‌های موجود در یکی و یا هر دو رابطه معین است.

تفاضل

رابطه‌ای را بازمی‌گرداند که حاوی تمام تاپل‌هایی است که در رابطه اول حضور دارند ولی در رابطه دو موجود نیستند.



شکل 1-2. جبر رابطه‌ای اصلی (نتایج هاشور خورده‌اند)

پیوند

رابطه‌ای را برمی‌گرداند که حاوی تمامی تاپل‌های حاصل از ترکیب‌های ممکن دو تاپل، که هر یک از آنها متعلق به یکی از رابطه‌هاست می‌باشد با این شرط که دو تاپل شرکت کرده در هر تاپل بدست آمده، دارای مقدار مشترک برای ویژگی‌هایی خاص در هر دو رابطه باشند (و این مقدار مشترک فقط یک بار –و نه دو بار– در تاپل بدست آمده دیده شود).

توجه: این نوع پیوند بطور سنتی پیوند طبیعی نامیده می‌شود. از آنجا که پیوند طبیعی اهمیت فوق‌العاده دارد، کلمه پیوند به پیوند طبیعی اطلاق می‌شود و من هم در این کتاب از این امر پیروی خواهم کرد.

تقسیم

دو رابطه که یکی از آنها باینری و یکی یونری است را دریافت می‌کند و رابطه‌ای را برمی‌گرداند که شامل تمام مقادیر یکی از ویژگی‌های رابطه باینری است به گونه‌ای که (ویژگی دیگر آن) با تمامی مقادیر موجود در رابطه یونری مطابقت داشته باشد.

آخرین نکته قبل از خاتمه این بحث: همانطور که احتمالا می‌دانید چیز دیگری هم بنام حساب رابطه‌ای وجود دارد. حساب رابطه‌ای را می‌توان بعنوان جایگزینی برای جبر رابطه‌ای دانست. به این معنا که به جای بیان عملیات به گونه جبر رابطه‌ای (به همراه انتساب)، از بیان آنالیز رابطه‌ای(به همراه انتساب) استفاده کنیم. این دو با هم مساوی و قابل جایگزینی با یکدیگر هستند. به نظر می‌رسد که برای هر عبارت جبر رابطه‌ای یک عبارت آنالیز رابطه‌ای وجود دارد و بالعکس. در ضمیمه آ کمی بیشتر در مورد حساب رابطه‌ای صحبت خواهم کرد.

مثال اجرایی

این مرور مختصر را با معرفی مثالی به پایان می‌برم که پایه اکثر مباحث آتی کتاب خواهد بود: مثال شناخته شده توزیع‌کنندگان و قطعات (شکل 1-3 را ببینید)

توزیع‌کنندگان

رابطه S توزیع‌کنندگان را مشخص می‌کند.(به بیان دقیق‌تر توزیع‌کنندگان طرف قرارداد) هر توزیع‌کننده یک شماره توزیع‌کننده (SNO) که بصورت یکتا به وی اختصاص یافته است (و {SNO} کلید اصلی است)، یک نام (SNAME) نه لزوما یکتا (در شکل 1-3 تصادفا یکتا است)، یک رتبه و یا پایه (STATUS) و یک محل(CITY)، دارد.

قطعه‌ها

رابطه P قطعات را مشخص می‌کند. هر قطعه یک شماره قطعه (PNO) که یکتا است (و {PNO} کلید اصلی است)، یک نام (PNAME)، یک رنگ (COLOR)، یک وزن (WEIGHT) و یک مکان که قطعه در آن انبار شده(CITY)، دارد.

سفارش‌ها

رابطه SP سفارش‌ها را مشخص می‌کند (نشان می‌دهد که کدام قطعات توسط کدامیک از توزیع‌کنندگان، تامین شده است). هر سفارش یک شماره توزیع‌کننده (SNO)، یک شماره قطعه (PNO) و یک تعداد (QTY) دارد. من فرض کرده‌ام که در هر زمان حداکثر یک سفارش برای یک قطعه و یک توزیع کننده خاص وجود دارد (بنابراین {SNO,PNO} کلید اصلی و همچنین {SNO} و {PNO} هر کدام کلید خارجی هستند و با کلید اصلی S و P مطابقت دارند). توجه کنید که مطابق شکل 1-3 توزیع‌کننده‌ای وجود دارد -توزیع کننده S5- که دارای هیچ سفارشی نیست.

S

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SNO	PNO	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

P

PNO	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12.0	London
P2	Bolt	Green	17.0	Paris
P3	Screw	Blue	17.0	Oslo
P4	Screw	Red	14.0	London
P5	Cam	Blue	12.0	Paris
P6	Cog	Red	19.0	London

شکل 1-3. بانک اطلاعاتی توزیع‌کنندگان و قطعات – با مقادیر نمونه

فرق بین مدل و پیاده‌سازی

قبل از هر چیز بایستی موضوعی را روشن کنم که پایه تمامی مطالب این کتاب است. بدیهی است که مدل رابطه‌ای، یک مدل داده است. متاسفانه کلمه مدل داده در دنیای پایگاه‌داده‌ها دارای دو معنای کاملا متفاوت است. معنای بنیادی‌تر چنین است:

تعریف: «مدل داده» (اولین معنی) یک تعریف تجریدی، خودکفا و منطقی از ساختارهای داده‌ای، عملگرها و ... می‌باشد که در کنار هم ماشینی انتزاعی تشکیل می‌دهند که قابلیت تراکنش با کاربران را دارد (طراحی) .

این برداشت مستقیما به مدل رابطه‌ای برمی‌گردد. با این تعریف می‌توانیم گام بعدی را با تمایز شناخت مدل رابطه‌ای و پیاده سازی آن برداریم که بصورت زیر تعریف می‌شود:

تعریف: «پیاده‌سازی» یک مدل داده تحقق فیزیکی ماشین انتزاعی و اجزای تشکیل دهنده مدل، بر روی یک ماشین واقعی است.

من این مفاهیم را با نگاه ویژه به مدل رابطه‌ای تشریح خواهم کرد. اول مفهوم شناخته شده رابطه: استفاده کنندگان باید بدانند که رابطه‌ها چه هستند و بدانند که از تاپل‌ها و ویژگی‌ها تشکیل شده‌اند. تمامی اینها اجزایی از مدل هستند. در مقابل نیازی نیست که بدانند که چگونه رابطه‌ها

بصورت فیزیکی بر روی دیسک ذخیره می‌شوند و مقادیر منحصر به فرد داده‌ها چگونه کدگذاری می‌شوند و ایندکس‌ها و مسیرهای دستیابی به اطلاعات چگونه ایجاد می‌شوند. تمامی اینها اجزای پیاده سازی –و نه اجزای مدل- هستند.

مثلا در مورد «پیوند»:کاربران باید بدانند که پیوند چیست و چگونه باید آنرا درخواست کرد و نتیجه آن چه شکلی دارد. تمام اینها مربوط به مدل (طراحی) هستند. ولی نباید بدانند که پیوند چگونه بصورت فیزیکی پیاده‌سازی می‌شود و یا در پشت پرده چه کارهایی انجام می‌گردد و یا کدام ایندکس‌ها و مسیرهای دستیابی مورد استفاده قرار می‌گیرند و در دستگاه‌های ورودی و خروجی چه اتفاقی رخ می‌دهد. تمامی اینها به پیاده سازی – و نه طراحی – مربوط می‌شوند.

بطور خلاصه:

- طراحی چیزی است که استفاده کننده باید آنرا بداند.
- پیاده‌سازی چیزی است که استفاده کننده نباید آنرا بداند.

(البته من نمی‌گویم که کاربران حق ندارند چیزی در مورد پیاده سازی بدانند، بلکه صحبت این است که می‌توان آنها را از این کار معاف کرد. بنابراین تمام مسائل مربوط به پیاده سازی بایستی –حداقل بصورت بالقوه- از دید کاربران پنهان نگه داشته شود.)

در اینجا برخی نتایج مطلب فوق را می‌بینید: اولا توجه کنید که کارایی موضوعی است که (بر خلاف تصور عامه) به پیاده سازی مربوط می‌شود ونه به طراحی. ما اکثرا می‌گوییم که پیوند به آهستگی انجام می‌شود. ولی این جمله بی معنا است. پیوند بخشی از طراحی است و طراحی نمی‌تواند سریع یا کند باشد. بلکه این امر بایستی به پردازش‌های موجود در پیاده سازی اطلاق شود. بنابراین می‌توان گفت محصول فلان از نظر نوع خاصی پیوند، از محصول بهمان سریعتر است.

توجه

من قصد ندارم که یک عقیده غلط را جا بیاندازم. درست است که کارایی در اصل به پیاده‌سازی مربوط می‌شود ولی این به این معنا نیست که اگر از مدل(طراحی) استفاده نادرست شود پیاده‌سازی خوبی بدست خواهد آمد ! این یکی از دلایلی است که به

موجب آن لازم است مدل را بشناسید. (در پاراگراف قبل فرض من این بود که هیچکس از مدل بد استفاده نمی‌کند).

اگر عبارتی مثل S JOIN SP بنویسید، در محدوده حقوق خود عمل کرده‌اید و پیاده سازی هم درخواست شما را به بهترین نحو انجام می‌دهد. ولی اگر اصرار داشته باشید که نحوه انجام آنرا هم خودتان مشخص کنید، مانند این:

```
do for all tuples in S ;
    fetch S tuple into TNO, TN, TS, TC ;
    do for all tuples in SP with SNO = TNO ;
      fetch SP tuple into TNO, TP, TQ ;
      emit tuple TNO, TN, TS, TC, TP, TQ ;
    end ;
  end ;
```

نخواهید توانست که به کارائی مناسبی برسید. سیستم‌های رابطه‌ای نباید مانند روش‌های ذخیره و بازیابی مورد استفاده قرار گیرند.

ثانیا، همانطور که احتمالا می‌دانید، تمایز منطقی بین طراحی و پیاده‌سازی ما را قادر به دستیابی استقلال داده‌ای می‌کند. استقلال داده‌ای (کلمه چندان مناسبی نیست با این حال احتمالا آنرا رها نخواهیم کرد.) بدین معناست که دارای این آزادی باشیم که نحوه ذخیره و بازیابی فیزیکی اطلاعات بر روی رسانه را تغییر دهیم بدون اینکه نیازی به تغییرات متناظر در دید کاربران وجود داشته باشد. دلیل این که ممکن است بخواهیم روش ذخیره و بازیابی را تغییر دهیم، بهبود کارایی است. ایجاد چنین تغییراتی بدون ایجاد تغییر در دید کاربران بدین معناست که برنامه‌های کاربردی موجود، کوئری‌ها و... همچنان قادر به کار خواهند بود. نکته بسیار مهم این است که استقلال داده‌ای به معنای «حفاظت از سرمایه‌گذاری انجام گرفته برای پرورش نیروی انسانی و برنامه‌های کاربردی» می‌باشد.

همان‌طور که ملاحظه کردید تمایز طراحی و پیاده‌سازی امری منحصر به فرد و در عین حال بسیار مهم است و بیشتر به عنوان تمایز سطوح فیزیکی و منطقی شناخته می‌شود. متاسفانه اکثر سیستم‌های بانک‌اطلاعاتی امروزی – حتی آنهایی که ادعای رابطه‌ای بودن را دارند- آنطور که باید و شاید این تمایز را به وجود نیاورده‌اند و در نتیجه استقلال داده‌ای را آنطور که شایسته است

و سیستم‌های رابطه‌ای در تئوری قابلیت آنرا دارند، فراهم نمی‌کنند. من به این موضوع در بخش بعد و همین‌طور فصل هفتم خواهم پرداخت.

اکنون می‌خواهم به دومین معنای مدل داده بپردازم و معتقدم که با آن مانوس هستید:

تعریف: «مدل داده» (دومین معنی) یعنی طرحی برای نگهداری داده‌های ماندگار یک موسسه بخصوص.

به بیان دیگر مدل داده (در معنای دوم) به طرح یک بانک اطلاعاتی خاص اطلاق می‌شود. برای مثال ممکن است از مدل داده برای یک بانک، یک بیمارستان و یا یک اداره دولتی صحبت کنیم.

حال که این دو معنای متفاوت شرح داده شدند، می‌خواهم توجه شما را به مقایسه‌ای جلب کنم که تصور می‌کنم رابطه آنها را به خوبی روشن می‌کند:

- یک مدل داده در معنای اول مانند یک زبان برنامه نویسی است که ممکن است برای حل برخی مسائل مورد استفاده قرار گیرد ولی در نهایت خود مستقیما به هیچ مساله خاصی مربوط نمی‌شود.
- یک مدل داده در معنای دوم مانند برنامه به خصوصی است که به زبان فوق نوشته شده است. این برنامه از امکانات فراهم شده توسط مدل در معنای اول، برای حل یک مساله خاص استفاده می‌کند.

به هر حال، نتیجه منطقی تمام مطالب بالا این است که اگر در مورد مدل‌های داده در معنای دوم صحبت می‌کنیم منظور به طور کل مدل رابطه‌ای و یا مدل رابطه‌ای خاصی(بدون تعریف مشخص) است. ولی اگر از مدل‌های داده در معنای اول صحبت کنیم منظور فقط یک مدل رابطه‌ای (با تعریف مشخص) است. در فصل هشتم بیشتر در این باره خواهم گفت.

در باقیمانده این کتاب لفظ مدل داده – و یا فقط مدل- را در معنای اول(طراحی) به کار خواهم برد.

خصوصیات رابطه‌ها

اکنون به مفاهیم پایه رابطه‌ای بازمی‌گردیم. در این فصل می‌خواهم بحث را بر روی برخی ویژگی‌های رابطه متمرکز کنم. قبل از هر چیز هر رابطه یک عنوان و یک بدنه دارد. عنوان

مجموعه‌ای از ویژگی‌هاست (منظور از ویژگی زوج ویژگی نام:نوع است) و بدنه مجموعه‌ای از تاپل‌ها می‌باشد که با عنوان مطابقت دارند. برای مثال در رابطه توزیع‌کنندگان شکل 1-3 چهار ویژگی در عنوان و پنج تاپل در بدنه وجود دارد. توجه داشته باشید که رابطه در واقع دارای تاپل نیست، بلکه دارای بدنه است و این بدنه است که تعدادی تاپل دارد. ولی ما معمولا برای سادگی چنین می‌گوییم.

کاملا صحیح است که بگوییم عنوان از زوج‌های ویژگی نام: نوع تشکیل شده است. با این وجود معمولا نوع‌ها از تصاویر (مانند شکل 1-3) حذف می‌شوند و چنین به نظر می‌رسد که عنوان فقط مجموعه‌ای از نام ویژگی‌ها است. مثلا ویژگی STATUS دارای یک نوع است (مثلا INTEGER) ولی من آن را در شکل 1-3 نشان نداده‌ام ولی شما هم نباید این موضوع را فراموش کنید!

تعداد ویژگی‌های عنوان درجه (و گاهی اریتی) و تعداد تاپل‌های بدنه کاردینالیتی خوانده میشود. برای مثال در رابطه‌های S، P و SP در شکل 1-3 دارای درجه به ترتیب 4، 5 و 3 بوده کاردینالیتی آنها به ترتیب 5، 6، و 12 می‌باشد.

رابطه‌ها هرگز دارای تاپل‌های تکراری نیستند. این ویژگی قابل استنباط است چرا که بدنه مجموعه‌ای از تاپل‌ها است و مجموعه‌ها در ریاضیات دارای اعضای تکراری نیستند. و اما SQL اینجا درمی‌ماند: همانطور که می‌دانید جداول SQL وجود سطرهای تکراری را مجاز می‌شمارند در نتیجه نمی‌توان جدول‌های SQL را همواره رابطه به حساب آورد. لطفا درک کنید که من در این کتاب کلمه «رابطه» را فقط به معنای رابطه، بدون تاپل‌های تکراری و عینا طبق تعریف به کار برده‌ام. همچنین متوجه باشید که عملگرهای رابطه‌ای نیز –باز هم طبق تعریف– نتایجی با تاپل‌های تکراری تولید نمی‌کنند برای مثال پرتو رابطه توزیع‌کنندگان بر روی CITY در شکل 1-3 نتیجه سمت چپ –و نه سمت راست– را تولید می‌کند.

CITY
London
Paris
Athens

CITY
London
Paris
Paris
London
Athens

(نتیجه سمت چپ ممکن است بوسیله این کوئری SQL بدست آید:

SELECT DISTINCT S.CITY FROM S

با حذف DISTINCT نتیجه‌ای غیر رابطه‌ای سمت راست حاصل خواهد شد. توجه کنید که در جدول سمت راست از دو خط زیر عنوان استفاده نکرده‌ام به این دلیل که این جدول کلید اصلی ندارد.)

موضوع دیگر اینکه تاپل‌های رابطه از بالا تا پایین بدون ترتیب هستند. این ویژگی نیز قابل استنباط است چرا که بدنه یک مجموعه است و اعضای مجموعه ترتیبی ندارند (مثلا {a,b,c} و {c,a,b} در ریاضیات با هم مساویند و این امر طبعا در مدل رابطه‌ای نیز صادق است). البته وقتی ما یک جدول را بر روی کاغذ رسم می‌کنیم مجبوریم سطرها را به ترتیب نشان دهیم ولیکن این ترتیب نشانه هیچ چیز در مدل رابطه‌ای نیست. برای مثال من می‌توانستم سطرهای جدول توزیع‌کنندگان را در شکل 1-3 به هر ترتیب دیگری از جمله S3 بعد S1 بعد S5 بعد S4 بعد S2 در شکل نشان دهم و شکل همچنان نشان دهنده همان رابطه باشد.

توجه

واقعیت این است که تاپل‌های رابطه نامرتب‌اند ولی این به این معنا نیست که کوئری‌ها نمی‌توانند دارای ORDER_BY باشند. بلکه بدین معناست که نتایج تولید شده از چنین کوئری‌هایی رابطه نیست. ORDER_BY برای نمایش نتایج سودمند است ولی در عین حال یک عملگر رابطه‌ای نیست.

به همین صورت، ویژگی‌های رابطه نیز از چپ به راست بدون ترتیب اند چرا که عنوان نیز یک مجموعه است ولی وقتی یک جدول را بر روی کاغذ رسم می‌کنیم مجبوریم ستون‌ها را به ترتیب از چپ به راست نشان دهیم ولیکن این ترتیب نیز نشانه هیچ چیز در مدل رابطه‌ای نیست. در جدول توزیع‌کنندگان شکل 1-3 می‌توانستم ستون‌ها را به هر ترتیبی از چپ به راست –مثل STATUS, SNAME, CITY, SNO– بچینم و شکل همچنان نشان دهنده همان رابطه باشد. اتفاقا SQL در اینجا هم ناتوان است. ستون‌ها در SQL حتما دارای ترتیب هستند (دلیل دیگر برای اینکه

جدول‌های SQL رابطه نیستند). مثلا دو شکل زیر نمایشگر یک رابطه و در عین حال دو جدول SQL متفاوت هستند:

SNO	CITY
S1	London
S2	Paris
S3	Paris
S4	London
S5	Athens

CITY	SNO
London	S1
Paris	S2
Paris	S3
London	S4
Athens	S5

(کوئری‌های ایجاد کننده این دو جدول عبارت است از:

```
SELECT S.SNO, S.CITY FROM S
SELECT S.CITY, S.SNO FROM S
```

ممکن است تصور کنید که این تفاوت اهمیت چندانی ندارد، ولی در واقع این امر نتایج و مسائل مهمی را دربردارد که در فصل‌های بعد بدان خواهیم پرداخت.)

مطلب بعدی اینکه رابطه‌ها همیشه نرمال شده‌اند. (یعنی در صورت اول نرمال یا 1NFهستند) به زبان ساده یعنی در نمایش تصویری جدول، در محل تلاقی هر سطر و ستون تنها یک مقدار دیده می‌شود و به زبان رسمی‌تر می‌توان گفت که هر یک از ویژگی‌های متعلق به هر تاپل رابطه دارای یک مقدار منفرد –و متناسب با نوع آن– می‌باشد. من در این مورد مطالب زیادی دارم که در فصل بعد خواهم گفت.

نکته دیگر اینکه ما بین رابطه‌های پایه و رابطه‌های ناشی‌شده تفاوت قائل می‌شویم. همانطور که قبلا گفته شد عملگرهای جبر رابطه‌ای ما را قادر می‌سازند که با استفاده از چند رابطه موجود – مانند آنچه در شکل 1-3 وجود دارد– رابطه‌های جدیدی بدست آوریم. رابطه‌های داده شده پایه و بقیه ناشی‌شده هستند. پس یک سیستم رابطه‌ای بایستی ابزارهایی جهت تعریف روابط پایه در اختیار داشته باشد. در SQL این کار با دستور CREATE_TABLE انجام می‌شود (در SQL جدول پایه معادل رابطه پایه است) و روابط پایه بایستی نام‌گذاری شوند.

```
CREATE TABLE SP ... ;
```

رابطه‌های ناشی‌شده که «دید» نامیده می‌شوند نیز دارای نام هستند. یک دید (که بعنوان رابطه مجازی هم شناخته می‌شود) رابطه نام‌داری است که مقدار آن در هر لحظه از زمان، با ارزیابی یک عبارت رابطه‌ای است که در همان لحظه بدست می‌آید. مانند مثال SQL زیر:

```
CREATE VIEW SST_PARIS AS
    SELECT S.SNO, S.STATUS
    FROM   S
 WHERE  S.CITY = 'Paris' ;
```

اگر دیدها رابطه‌های پایه باشند شما می‌توانید با آنها کار کنید. ولی آنها رابطه‌های پایه نیستند. با این وجود می‌توانید تصور کنید که دیدها تبدیل به جدول شده‌اند و یا فرض کنید که دیدها رابطه‌های پایه‌ای هستند که بصورت پویا و در زمان مراجعه ایجاد می‌شوند (در هر صورت مجبوریم که این طور تصور کنیم که دیدها به هنگام مراجعه جدول هستند چرا که راه دیگری وجود ندارد. ولی این اتفاقی نیست که در واقعیت رخ می‌دهد. اینکه دیدها واقعا چگونه کار می‌کنند موضوعی است که در فصل چهارم مورد بحث قرار خواهد گرفت).

در اینجا بایستی به نکته مهمی اشاره کنم. اکثر مردم تفاوت بین رابطه‌های پایه و دیدها را به این صورت بیان می‌کنند.

- رابطه‌های پایه واقعا وجود دارند و بصورت فیزیکی در بانک اطلاعاتی ذخیره شده‌اند.
- دیدها وجود خارجی ندارند آنها فقط راه‌های مختلفی برای نگریستن به رابطه‌های پایه در اختیار ما قرار می‌دهند.

ولی مدل رابطه‌ای هیچ صحبتی در مورد اینکه چه چیزی بصورت فیزیکی ذخیره می‌شود، ندارد و نمی‌گوید که رابطه‌های پایه باید به صورت فیزیکی ذخیره می‌شوند. تنها باید تناظری بین رابطه‌های پایه و آنچه بصورت فیزیکی ذخیره می‌شود وجود داشته باشد و رابطه‌های پایه هم بایستی در زمان نیاز به طریقی ایجاد شوند. بنابراین رابطه‌های پایه می‌توانند به این طریق (تناظر با داده‌های ذخیره شده) و یا هر روش دیگری تولید شوند. برای مثال می‌توان پیوند دو رابطه توزیع‌کنندگان و سفارش‌ها را بصورت فیزیکی ذخیره کرد و یا بجای این کار روابط S و SP را بطور جداگانه ذخیره نمود و پیوند را در زمان نیاز بصورت مجازی ایجاد کرد. به بیان دیگر، مطابق مدل رابطه‌ای، رابطه‌های پایه از دیدها فیزیکی‌تر نیستند.

مدل رابطه‌ای عمدا هیچ چیز در مورد ذخیره فیزیکی نمی‌گوید تا سازندگان سیستم‌های بانک اطلاعاتی بتوانند آزادانه و از هر راهی که می‌خواهند–و تصور می‌کنند از آن راه به کارایی بهتری می‌رسند، البته بدون از دست دادن استقلال داده‌ها – به پیاده‌سازی آن بپردازند. واقعیت تاسف‌بار این است که اکثر تولیدکنندگان SQL این موضوع را درک نکرده‌اند. آنها جدول‌های پایه را

مستقیما به دستگاه‌های ذخیره سازی فیزیکی* منتسب می‌کنند و (همانطور که در قسمت قبل گفته شد) از این رو محصولات‌شان از استقلال داده‌ای کافی-در حدی که تئوری رابطه‌ای تعیین می‌کند- برخوردار نیست. این مشکل در خود SQL استاندارد -و همچنین اکثر نگارش‌های SQL- که از عباراتی نظیر «جدول‌ها و دیدها» استفاده کرده اند وجود دارد. بدیهی است کسی که با چنین عباراتی سروکار دارد تصور می‌کند که جدول‌ها و دیدها دو چیز متفاوت هستند و بر این اساس به این نتیجه می‌رسد که جدول‌ها فیزیکی و دیدها غیر فیزیکی هستند. اما نکته‌ای که در مورد دیدها پوشیده مانده این است که آنها هم جدول (و یا من ترجیح میدهم بگویم رابطه) هستند. به همین علت است که ما میتوانیم عملگرهای رابطه‌ای را بر آنها (مانند رابطه‌های معمولی) اعمال کنیم. چرا که دیدها «رابطه‌های معمولی» اند. در این کتاب من از لغت رابطه به معنای رابطه استفاده خواهم کرد (خواه رابطه‌ای پایه باشد، خواه یک دید، خواه نتیجه یک کوئری و یا هر چیز دیگر) و اگر منظورم رابطه پایه باشد آنگاه کلمه «رابطه پایه» را به کارخواهم برد و به شما هم شدیدا توصیه می‌کنم که از این مقررات تبعیت کنید و دچار عوام‌زدگی نشوید و تصور نکنید کلمه «رابطه» فقط به معنای رابطه پایه است.

فرق بین رابطه‌ها و مرابطه‌ها

این احتمال وجود دارد که از مطالبی که تا کنون مطرح گردید تا حدودی مطلع باشید و امیدوارم که اینطور باشد (این امیدواری به این معنا نیست که مباحث برای شما ملال‌آور بوده است). در هر صورت هم اکنون قصد دارم مطلبی را ارائه کنم که احتمالا آن را نمی‌دانید. واقعیت این است که در مفاهیم رابطه‌ها -در معنایی که بیان شد- و «متغیر»های رابطه‌ای اختلاط وجود دارد.

برای چند لحظه بانک اطلاعاتی را فراموش کنید و مثال برنامه‌نویسی زیر را در نظر بگیرید. فرض کنید این دستور در یک زبان برنامه‌نویسی داده شده باشد:

```
DECLARE N INTEGER ... ;
```

* همه می‌دانند که محصولات SQL امروزی خودشان امکاناتی همچون پارتیشن‌بندی، ایندکس‌بندی، کلاستربندی و در کل سازمان‌دهی داده‌های ذخیره شده در دیسک را فراهم می‌کنند. با این وجود من همچنان معتقدم که نگاشت به دستگاه‌های ذخیره‌سازی فیزیکی در این محصولات هنوز «تا حدودی مستقیم» است.

در این صورت N یک integer نیست. بلکه یک متغیر است که مقادیر integerی را می‌پذیرد (در دفعات متفاوت مقادیر صحیح مختلف) موافقید؟ مطمئنم که شما هم این مطلب را تایید می‌کنید. دقیقا به همین صورت اگر در SQL بگوییم:

CREATE TABLE T ... ;

در اینصورت T یک جدول نیست بلکه یک متغیر جدول و یا یک متغیر رابطه‌ای است که مقادیر از نوع رابطه را می‌پذیرد (در دفعات متفاوت رابطه‌های مختلف).

شکل 1-3 را مجددا در نظر بگیرید. این شکل سه مقدار رابطه‌ای را نشان می‌دهد: بطور مشخص وضعیت بانک اطلاعاتی در یک زمان خاص نشان داده شده است. ولی اگر در زمان دیگری به بانک اطلاعاتی نگاه می‌کردیم، سه مقدار رابطه‌ای دیگر را در آنجا می‌دیدیم. به زبان دیگر S، P و SP در این بانک‌اطلاعاتی واقعا متغیرند: دقیقا متغیرهای رابطه‌ای. برای مثال فرض کنید که متغیر رابطه‌ای S هم اکنون دارای مقداری است- مقدار رابطه‌ای- که در شکل 1-3 نشان داده شده است. فرض کنید که برخی تاپل‌ها (در واقع فقط یک تاپل) که مربوط به توزیع کنندگان مستقر در آتن است را حذف می‌کنیم:

DELETE S WHERE CITY = 'Athens' ;

نتیجه چنین خواهد بود:

S

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London

می‌توان چنین تصور کرد که مقداری جدید جایگزین مقدار قبلی S شده است. البته مقدار قدیمی(با پنج تاپل) و مقدار جدید با هم شباهت زیادی دارند ولی با این وجود آنها دو مقدار متفاوت اند. در حقیقت DELETE معادل منطقی و خلاصه شده در دستور انتساب رابطه‌ای زیر است*:

* من نمی‌توانم این دستور را به SQL بنویسم چرا که SQL دستور انتساب را مستقیما پشتیبانی نمی‌کند.در این کتاب من مثال‌ها را تا جایی که ممکن است با SQL نشان می‌دهم و در جایی که به هر دلیل این کار ممکن نیست-مانند همین جا- از یک زبان کمابیش بدون شرح(و واقعا رابطه‌ای) بنام تاتوریال‌دی استفاده خواهم کرد.این زبان را هیو

```
S := S WHERE NOT ( CITY = 'Athens' ) ;
```

در تمامی انتساب‌ها، «عبارت مبدا» (a) در سمت راست ارزیابی شده، و نتیجه این ارزیابی به «متغیر مقصد» (b) در سمت چپ منتسب می‌شود.

عبارت‌های مانوس INSERT و DELETE نیز خلاصه نویسی شده دستورات انتساب رابطه‌ای هستند. همانطور که در قسمت «مروری بر مدل اصلی» عنوان شد انتساب رابطه‌ای عملگر به‌روزرسانی اصلی مدل رابطه‌ای و در حقیقت تنها عملگر به‌روزرسانی مورد نیاز است.

بنابراین ما به دو موضوع مختلف سروکار داریم، مقادیر رابطه‌ای و متغیرهای رابطه‌ای. مشکل این است که در گذشته به دلیل اختصار، برای هر دو آنها از کلمه «رابطه» استفاده شده است و این روش قطعا موجب سردرگمی می‌شود. بنابراین در این کتاب از این پس من بین این دو مفهوم با دقت تمایز قائل شده‌ام و اگر منظور مقدار رابطه‌ای است از کلمه مقدار رابطه‌ای و اگر منظور متغیر رابطه‌ای است از کلمه متغیر رابطه‌ای استفاده کرده‌ام. هر چند که من هم اکثر اوقات مقدار رابطه‌ای را، رابطه (همان طور که مقدار بیشتر اوقات مقدار integerی (اینتیجری) را integer می‌نامند) و متغیر رابطه‌ای را مرابطه نامیده‌ام. برای مثال می‌گویم که بانک اطلاعاتی توزیع‌کنندگان و قطعات دارای سه مرابطه است.

برای تمرین می‌توانید به مطالب قبلی این فصل بازگردید و ببینید من در کجاها از کلمه رابطه استفاده کرده‌ام در حالی که بایستی بجای آن از مرابطه استفاده می‌کردم.

فرق بین مقدارها و متغیرها

تفاوت بین رابطه‌ها و مرابطه‌ها حالت خاصی از تفاوت عمومی بین مقدارها و متغیرها است و من می‌خواهم زمان کوتاهی را صرف این تفاوت عمومی‌تر کنم. (شاید کمی انحراف از موضوع باشد ولی تصور می‌کنم که ارزش آنرا دارد. چرا که روشن شدن این امر می‌تواند به فهم بسیاری از مطالب بعدی کمک زیادی نماید.) به چند تعریف توجه کنید:

تعریف: یک مقدار یک «ثابت منحصر بفرد» است: برای مثال عدد صحیح منحصر بفرد 3. یک مقدار جایی در زمان و مکان ندارد. مقدارها می‌توانند در حافظه، معنای خاصی را به ذهن

داروین و من در کتاب پایگاه‌های داده‌ها، نوع‌ها و مدل رابطه‌ای مورد استفاده قرار داده‌ایم و شما میتوانید برای درک مفاهیم انتزاعی مدل رابطه‌ای آن را مطالعه کنید (چیزی که متاسفانه SQL فاقد آن است).

بیاورند، و این به ذهن آوردن هم جایی در زمان و مکان ندارد. در واقع هر مقدار می‌تواند برای نمایش چیزهای مختلفی در زمان و مکان بکار رود. این تقریبا به این معناست که تعدادی متغیرمختلف –طبق تعریف زیر– می‌توانند، در یک زمان یا زمان‌های مختلف دارای مقداری یکسان باشند. نکته مهم این که یک مقدار قابل به‌روزرسانی نیست و اگر به فرض محال این کار انجام شود، بعد از به‌روزرسانی دیگر یک مقدار نخواهد بود.

تعریف: یک متغیر نگهداری کننده یک مقدار است. یک متغیر در ظرف زمان و مکان جا دارد و همچنین یک متغیر بر خلاف یک مقدار قابل به‌روزرسانی است. مقدار فعلی متغیر می‌تواند با مقدار دیگری جایگزین شود.

لطفا دقیقا توجه کنید که تنها چیزهای ساده مانند عدد صحیح 3 ، «مقدار» های قابل‌قبول نیستند. مقدارها می‌توانند تا حد دلخواه پیچیده باشند. برای مثال یک نقطه هندسی، یک چند ضلعی، یک تابش اشعه ایکس، یک نوشتار XML، یک اثر انگشت، یک آرایه، یک پشته، یک لیست و یک رابطه همگی مقدار هستند. مقیاس‌ها برای معنادهی به متغیرها بوجود آمده‌اند. من در مورد این موضوع در دو فصل آینده توضیحات بیشتری می‌دهم.

ممکن است برایتان عجیب باشد که چگونه مردم در تفاوت بین مقدارها و متغیرها دچار اشتباه می‌شوند. ولی واقعیت این است که این مساله برای بسیاری از افراد پیش می‌آید. برای روشن شدن این موضوع ، متن زیر را از کتاب خودآموز بانک‌های اطلاعاتی شی‌گرا نقل قول کرده‌ام. (مطالب داخل کروشه را خودم اضافه کرده‌ام):

ما بین دو مفهوم متغیر و.. تمایز قائل می‌شویم. شی به مقدار فعلی متغیر گفته می‌شود [شی یک مقدار است]. ما بین اشیا و مقدارها تمایز قائل می‌شویم[پس شی مقدار نیست] یک «تغییر دهنده» می‌تواند بر برخی اشیا اثر بگذارد [پس شی یک متغیر است].

خلاصه

بیشتر در این فصل مقدماتی را بیان کردم و امیدوارم اکنون آنها را فراگرفته باشید.(ممکن است این مطالب برایتان آسان باشد). به هر حال خلاصه این مطالب به شرح زیر است:

- توضیح دادم که چرا باید به قواعد توجه کنیم و نه به محصولات. همچنین چرا از اصطلاحات رسمی مانند رابطه، تاپل و ویژگی استفاده می‌کنیم و نه از معادل‌های دوستانه آنها در SQL.
- ادعا کردم که مدل رابطه‌ای و SQL یکی نیستند. برخی از این تفاوت‌ها را هم اکنون می‌دانیم-مثلا SQL سطرهای تکراری را مجاز می‌شمارد.- و بسیاری تفاوت‌های دیگر را هم در فصل‌های بعد خواهیم دید.
- آشنایی مختصری با مدل اصلی ارائه شد و این موضوعات مورد بررسی قرار گرفت: نوع، رابطه n اری، تاپل، ویژگی، کلید کاندید، کلید اصلی، کلید خارجی، جامعیت وجودی، جامعیت ارجاعی، انتساب رابطه‌ای و جبر رابطه‌ای. به خاصیت شرکت‌پذیری نیز اشاره کردم و بصورت بسیار مختصر عملگرهای گزینش، پرتو، ضرب، اشتراک، اجتماع، تفاضل، پیوند و تقسیم را شرح دادم.
- در مورد مشخصات رابطه بحث کردم و عبارات عنوان، بدنه، کاردینالیتی و درجه را معرفی کردم. رابطه‌ها تاپل تکراری، ترتیب تاپل‌ها از بالا به پایین و ترتیب ویژگی‌ها از چپ به راست را ندارند. همچنین در مورد دیدها بحث کردم.
- تفاوت‌های میان مدل و پیاده‌سازی، رابطه‌ها و مرابطه‌ها و مقادیر و متغیرها را روشن کردم. فرق بین مدل و پیاده سازی بحثی است که ما را به استقلال داده‌ها می‌رساند.

و آخرین نکته (که تاکنون آنرا مطرح نکرده‌ام ولی امیدوارم از مباحث گذشته به این نتیجه رسیده باشید): در مجموع مدل رابطه‌ای طبیعتی اعلان‌گرا -و نه روال گرا- دارد. ما روش‌های اعلان‌گرا را بر روش‌های روندگرا ترجیح میدهیم چرا که ملموس‌تر هستند. دلیل این امر هم روشن است: اعلان‌گرا به این معناست که سیستم کارها را انجام می‌دهد و روال‌گرا به این معناست که کاربر کارها را انجام می‌دهد. از همین رو مدل رابطه‌ای از کوئری‌های اعلانی، به‌روزرسانی‌های اعلانی، تعریف دید بصورت اعلانی و تعریف قواعد جامعیت بصورت اعلانی، پشتیبانی می کند.*

* وقتی که این کتاب در دست چاپ بود مطلع شدم که حداقل یکی از محصولات شناخته شده SQL کلمه «اعلانی» را به معنای عدم انجام کار بوسیله سیستم استفاده کرده است! به این معنا که کاربر مجاز است دستورات را بصورت

تمرین‌ها

1- (تکراری از متن کتاب با اندکی تغییر) اگر تاکنون آنرا انجام نداده‌اید، مطالب این فصل را مرور کنید و در هر کجا که کلمه رابطه را به کار برده‌ام، بگویید که صحیح است و یا باید بجای آن مرابطه را به کار می‌بردم.

2- ای اف کاد که بود؟

3- دامنه چیست؟

4- از جامعیت ارجاعی چه فهمیدید؟

5- لغات عنوان، بدنه، ویژگی، تاپل، کاردینالیتی و درجه که در قسمت مقادیر رابطه‌ای تعریف شدند، به همان صورت قابل تعمیم به مرابطه‌ها هم هست. مطمئن شوید که این جمله را فهمیده‌اید.

6- دو معنای متفاوت مدل داده را شرح دهید.

7- تفاوت بین مدل و پیاده‌سازی را به زبان خودتان بیان کنید.

8- در متن گفتم که جدول‌های رسم شده مانند 1-1 و 3-1 رابطه نیستند، بلکه تصویر شده رابطه هستند. تفاوت‌هایی که بین این تصاویر و روابط متناظر با آنها وجود دارد چه هستند؟

9- استقلال داده‌ای را به زبان خودتان تعریف کنید.

10- (سعی کنید این تمرین را بدون نگاه کردن به متن کتاب انجام دهید) چه مرابطه‌هایی در بانک اطلاعاتی توزیع‌کنندگان و قطعات وجود دارد؟ چه ویژگی‌هایی در هر یک آنها وجود دارد؟ کلیدهای اصلی و خارجی کدامند؟ (نکته این تمرین این است که خود را با این مثال آشنا می‌کنید. واضح است که نبایستی جزئیات داده‌های واقعی را حفظ کنید.)

اعلانی بگوید ولی مجبور به این کار نیست. چنین استفاده‌های نابجایی از اصطلاحات فنی، هیچ کمکی به فهم بهتر مطلب نمی‌کند.

11- «فقط یک مدل رابطه‌ای وجود دارد.» این جمله را توضیح دهید.

12- مطلب زیر از یک کتاب جدید پایگاه داده‌ها نقل قول شده است: «مهم است که بین رابطه‌های ذخیره شده –جدول‌ها- و رابطه‌های مجازی –دیدها- تمایز قائل شویم. ما بایستی کلمه رابطه را برای یک جدول و یا دید قابل استفاده بکار بریم و زمانی که قصد داریم بر رابطه‌های ذخیره شده تاکید کنیم از لغات رابطه پایه و یا جدول پایه استفاده می‌کنیم.» این متن دارای تناقضات و اشتباهات زیادی در مورد مدل رابطه‌ای است. هر تعداد از آنها را که می‌توانید پیدا کنید.

13- متن زیر از کتاب جدید دیگری در رابطه با پایگاه‌داده‌ها استخراج شده است:«مدل رابطه‌ای ... برای هر رابطه، جدول‌هایی ساده و روابط چند به چند تعریف می‌کند. کلیدهای ارجاعی جدول‌ها را به هم متصل می‌کنند و رابطه بین آنها را نشان می‌دهند. ایندکس‌های اولیه و ثانویه راه سریعی برای دستیابی به اطلاعات فراهم می‌کنند.» این متن تعریف مدل رابطه‌ای است... چه اشکالی در آن وجود دارد؟

14- چند دستور SQL بصورت CREATE TABLE بنویسید که بانک اطلاعاتی توزیع‌کنندگان و قطعات را تعریف کند.

15- دستور INSERT زیر در SQL و مربوط به بانک‌اطلاعاتی توزیع‌کنندگان و قطعات است.

```
INSERT INTO SP ( SNO, PNO, QTY )
    VALUES ( SNO('S5'), PNO('P6'), QTY(250) ) ;
```

معادل این دستور را با انتساب رابطه‌ای بنویسید. توجه: فرض کرده‌ام ویژگی‌های SNO، PNO و QTY از نوع SNO، PNO و QTY هستند و به ترتیب عبارات SNO('S5')، PNO('P6') و QTY(250) با این انواع مطابقت دارند. من مطالب بیشتری در این مورد در دو فصل آینده دارم. من می‌دانم که هنوز نحوه و جزئیات دستور انتساب رابطه‌ای را شرح نداده‌ام. زیاد نگران درست

نوشتن پاسخ نباشید و فقط نهایت سعی خود را بکنید.

16- دستور UPDATE زیر مربوط به پایگاه توزیع کنندگان و قطعات است.

```
UPDATE S
SET    STATUS = 25
WHERE  S.CITY = 'Paris' ;
```

معادل این دستور را با انتساب رابطه‌ای بنویسید.(هدف از این تمرین آن است که در مورد آن تفکر کنید. من هنوز آنقدر در این مورد برای شما توضیح نداده‌ام که بتوانید پاسخ کاملا صحیح به این سوال بدهید. فصل 5 را جهت مباحث آتی ببینید.)

17- در این فصل گفتم که SQL بطور مستقیم دستور انتساب را پشتیبانی نمی‌کند. آیا بطور غیر مستقیم این کار ر ا انجام می‌دهد؟ اگر بلی چگونه؟

18- از نقطه نظر کاربردی فکر می‌کنید چرا تاپل‌های تکراری، ترتیب بالا به پایین تاپل‌ها و ترتیب چپ به راست ویژگی‌ها ایده‌های بسیار بدی هستند؟ (این سه سوال در متن پاسخ داده نشده‌اند و این تمرین بیشتر برای بحث در گروه‌ها مناسب است. ما در آینده بیشتر به این قبیل مطالب خواهیم پرداخت.

فصل دوم

تفاوت

رابطه‌ها و نوع‌ها

عنوان این فصل تفاوت رابطه‌ها و نوع‌ها است اما بیشتر مطالب آن به نوع‌ها مربوط است. نکته اینجاست که مدل رابطه‌ای نیازمند به سیستمی برای پشتیبانی از نوع‌هاست ولی مدل چندان به طبیعت این سیستم نپرداخته است. چرا مدل رابطه‌ای به نوع نیاز دارد؟ زیرا رابطه‌ها (و مرابطه‌ها) با توجه به نوع‌ها تعریف می‌شوند. *هر ویژگی از هر رابطه (و هر مرابطه) به هنگام تعریف باید دارای یک نوع باشند.* برای مثال ویژگی STATUS متعلق به توزیع‌کنندگان و از مرابطه S می‌تواند دارای نوع INTEGER باشد. اگر چنین باشد هر رابطه مانند s که یک مقدار احتمالی برای مرابطه S است باید دارای یک ویژگی STATUS از نوع INTEGER باشد. این بدین معناست که هر تاپل رابطه s باید دارای STATUSی باشد که مقدار آن یک عدد صحیح است.

من این مطلب را با جزئیات بیشتر در این فصل شرح خواهم داد. برای شروع فقط می‌گویم –با استثناهای مهمی که بعدا خواهم گفت–که ویژگی‌های رابطه می‌توانند دارای هر *نوعی* باشند (نوع‌ها می‌توانند هر قدر که ما می‌خواهیم پیچیده باشند، همانطور که در آینده خواهیم دید). هر ویژگی می‌تواند دارای نوع تعریف‌شده در سیستم (توکار) و یا نوع تعریف‌شده بوسیله کاربر باشد. در مورد مثال اجرایی خودمان، من فرض کرده‌ام که ویژگی‌ها دارای نوع‌های زیر هستند (توجه داشته باشید که برخی ویژگی‌ها دارای نام یکسان با نوع‌شان هستند).

Suppliers	Parts	Shipments
SNO : SNO	PNO : PNO	SNO : SNO
SNAME : NAME	PNAME : NAME	PNO : PNO
STATUS : INTEGER	COLOR : COLOR	QTY : QTY
CITY : CHAR	WEIGHT : WEIGHT	
	CITY : CHAR	

من همچنین فرض کرده‌ام که نوع‌های INTEGER (عدد صحیح) و CHAR (رشته حرفی با طول دلخواه) تعریف شده در سیستم و سایر نوع‌ها تعریف شده بوسیله کاربر هستند.

SQL دارای نوعی به نام INTEGER است –مطمئنم این را می‌دانستید–. همچنین نوع CHAR در آن وجود دارد ولیکن رشته‌ای با طول ثابت است که این طول درون پرانتز به همراه آن ذکر می‌شود مانند: CHAR(*n*). (CHAR بدون مشخص کردن طول مختصر شده CHAR(1) است ولی این کار زیاد معمول نیست.) SQL همچنین تعریف نوع توسط کاربر را مجاز می‌شمارد.[*]

طبق مستندات تاریخی وقتی کاد برای اولین بار مدل رابطه‌ای را تعریف کرد، گفت که رابطه‌ها بر روی دامنه‌ها تعریف می‌شوند و نه بر روی نوع‌ها. با این حال حقیقت این است که دامنه‌ها و نوع‌ها *دقیقا یکی هستند*. شما می‌توانید این ادعا را نظر من بدانید و من قصد دارم در دو بخش آینده دلیل خود را بیان کنم. من بحث را با مدل اصلی کاد شروع می‌کنم و تا اطلاع ثانوی از کلمه دامنه –و نه نوع– استفاده خواهم کرد. دو موضوع عمده برای طرح در هر قسمت وجود دارد:

مقایسه با دامنه تحمیلی و «عدم تطبیق دامنه»

امیدوارم این بحث شما را قانع کند که دامنه‌ها واقعا همان نوع‌ها هستند.

اتمی بودن مقدار داده‌ها و فرم اول نرمال

و امیدوارم مطالعه این قسمت شما را قانع کند که نوع‌ها می‌توانند تا حد دلخواه پیچیده باشند.

مقایسه با دامنه تحمیلی

هر کس می‌داند (و شما هم باید بدانید!) که در مدل رابطه‌ای می‌توان دو مقدار را از نظر برابر بودن با هم مقایسه کرد تنها اگر هر دو مقدار از یک دامنه باشند. در مورد توزیع‌کنندگان و قطعات، عبارت زیر –که WHERE بر روی قطعات اعمال شده است– کاملا معتبر است:

```
SP.SNO = S.SNO        /* OK    */
```

[*] در SQL نوع دیگری بنام varchar وجود دارد که در آن طول رشته‌ها متغیر است با این حال بایستی حداکثر طول رشته معین شود.

در مقابل این عبارت معتبر نیست:

```
SP.PNO = S.SNO        /* not OK */
```

به این دلیل که شماره قطعه و شماره توزیع‌کننده دو چیز متفاوت هستند و از دو دامنه مختلف ناشی شده‌اند و سیستم مدیریت پایگاه داده‌ها (DBMS)* قاعدتا بایستی تقاضاهایی از قبیل پیوند، اجتماع، تقسیم و هر چیز دیگر که در آن، بصورت صریح یا ضمنی، مقایسه بین مقدارهایی از دامنه‌های متفاوت وجود دارد را رد کند. برای مثال در عبارت SQL زیر کاربر سعی دارد که توزیع‌کنندگانی که هیچ قطعه‌ای ارائه نکرده‌اند را پیدا کند:

```
SELECT S.SNO, S.SNAME, S.STATUS, S.CITY
FROM   S
WHERE  NOT EXISTS
   ( SELECT SP.PNO
     FROM   SP
     WHERE  SP.PNO = S.SNO )      /* not OK */
```

(به سمیکالن پایانی نیازی نیست چرا که این یک عبارت است نه یک دستور. تمرین 24 را ببینید کنید.)

همان‌طور که در توضیح می‌بینید این کوئری صحیح *نیست*. علت آن است که در سطر آخر احتمالا کاربر می‌خواسته عبارت WHERE SP.SNO = S.SNO را بنویسد ولی به دلیل اشتباه تایپی آن را به صورت WHERE SP.PNO = S.SNO نوشته است. در مورد این اشتباه کوچک تایپی DBMS می‌تواند با عملکردی دوستانه، کار را متوقف نموده و با نشان دادن محل اشتباه از کاربر بخواهد که آنرا اصلاح کند.

من نمی‌دانم که کدامیک از محصولات تجاری چنین عملکردی دارند. در محصولات امروزی –بسته به اینکه شما چگونه پایگاه را ایجاد کرده باشید– یا کوئری با شکست مواجه

---

* تفاوت بین DBMS و پایگاه داده‌ها چیست؟ این سوال بیهوده‌ای نیست، چرا که تولیدکنندگان –مانند اوراکل و برخی محصولات دیگر، هنگامی که یک نسخه از آنها در حال نصب بر روی کامپیوتر است– معمولا از لفظ بانک‌اطلاعاتی بجای DBMS استفاده می‌کنند. اشکال این کار این است که اگر به DBMS پایگاه داده‌ها (دیتابیس) بگوییم، آنوقت به پایگاه داده‌ها چه بگوییم؟

خواهد شد و یا پاسخ اشتباهی به شما نشان خواهد داد. نه دقیقا پاسخ اشتباه، بلکه پاسخی است صحیح به یک درخواست اشتباه. (آیا این کار مزیتی برای شما دارد؟)

مجددا تاکید می‌کنم که DBMS باید درخواست‌های مقایسه غیرمجازی مانند SP.PNO=S.SNO را مردود اعلام کند. کاد عقیده داشت که باید راهی برای کاربر وجود داشته باشد که DBMS را مجبور به انجام مقایسه‌ها –حتی مقایسه‌های غیر مجاز– کند. چرا که در بسیاری موارد عقل کاربر بیشتر از عقل DBMS می‌رسد. برای من مشکل است که درستی این عقیده را تایید کنم –چون آن را قبول ندارم– ولی سعی خودم را می‌کنم.

فرض کنید که کار شما طراحی بانک‌اطلاعاتی است و مشغول طراحی پایگاه مشتریان و توزیع‌کنندگان هستید و قصد دارید دو دامنه، یکی برای شماره مشتری و دیگری برای شماره توزیع‌کننده داشته باشید. شما پایگاه را می‌سازید و به کار می‌اندازید و تا یکی دو سال همه چیز به خوبی کار می‌کند تا اینکه یک روز یکی از کاربران می‌خواهد یک کوئری را برای نخستین بار به اجرا درآورد، به این منظور که: «آیا هیچکدام از مشتریان هستند که توزیع‌کننده هم باشند؟» ملاحظه می‌شود که این کوئری یک درخواست عاقلانه است. همچنین ملاحظه می‌شود که *ممکن است* درگیر مقایسه‌ای بین شماره مشتری و شماره فروشنده شویم به این منظور که معلوم شود این دو با هم یکی هستند و یا خیر و سیستم هم نبایستی مانع از انجام این کار شود و از یک کوئری عاقلانه جلوگیری کند.

بعد از این مقدمه، کاد راه‌حلی بنام *نقض تطبیق دامنه* یا (DCO) را پیشنهاد می‌کند که تفسیر جدیدی از عملگرهای جبری‌اش است. مثلا در نگارش DCO از پیوند، حتی اگر ویژگی‌هایی که پیوند بر روی آن انجام می‌شود از یک دامنه نباشند مشکلی پیش نمی‌آید. این ایده را می‌توان با کمک افزودن یک بند جدید به دستور SQL عملی کرد. مانند این:

```
SELECT ...
FROM   ...
WHERE  CUSTNO = SNO
IGNORE DOMAIN CHECKS
```

این دستور اختیاری است و اکثر کاربران نبایستی مجاز به استفاده از آن باشند و احتمالا فقط مدیر پایگاه (DBA) باید قادر به استفاده از آن باشد.

قبل از بررسی جزئیات ایده DCO می‌خواهم مثال مشابهی را بیازماییم. به دو کوئری زیر دقت کنید:

```
SELECT ...                          | SELECT ...
FROM   P, SP                        | FROM   P, SP
WHERE  P.WEIGHT = SP.QTY            | WHERE  P.WEIGHT - SP.QTY = 0
```

اگر وزن و تعداد از دو دامنه مجزا تعریف شوند، عاقلانه خواهد بود. در این صورت کوئری چپ بطور واضح غیر مجاز است ولی در مورد کوئری سمت راست چه می‌توان گفت؟ طبق نظر کاد این کار مجاز است! در کتاب وی تحت عنوان *مدل رابطه‌ای و مدیریت پایگاه داده‌ها نگارش*2،1990، او درباره چنین موقعیتی می‌گوید « DBMS [فقط] بر یکی بودن نوع‌های داده‌ای پایه نظارت می‌کند» در این مورد «نوع داده‌ای پایه» هر دو طرف عددی است و این بررسی با موفقیت انجام خواهد شد.

از نظر من چنین چیزی غیر قابل قبول است. معنای یک عبارت نباید به شکل نوشتن آن وابسته باشد. از این رو من فکر می‌کنم دو عبارت P.WEIGHT=SP.QTY و P.WEIGHT-SP.QTY=0 بایستی یا هر دو مجاز و یا هر دو غیر مجاز باشند و به نظر می‌رسد که هر دو این حالت‌ها باید غیر مجاز باشند. احساس می‌شود که روش تطبیق دامنه کاد قدری عجیب است. (در واقع روش کاد در بررسی تطبیق دامنه فقط برای حالت خاصی مناسب است که هر دو طرف بعنوان ویژگی‌های رابطه – نه هیچ چیز دیگر– در نظر گرفته شوند.)

چند مثال ساده دیگر در مورد تطبیق نوع. (هر یک از آنها می‌تواند در WHEREهای SQL به کار رود.)

```
S.SNO = 'X4'        P.PNO = 'X4'        S.SNO = P.PNO
```

امیدوارم با این نظر موافق باشید که دوتای اول مجاز و سومی غیر مجاز است. اگر چنین است باز هم امیدوارم موافق باشید که این وضعیت قدری عجیب است. فرض کنید سه متغیر *a*، *b* و *c* داشته باشیم. اگر $a=c$ صحیح و $b=c$ صحیح باشد در مورد $a=b$ ..... این مقایسه امکان پذیر نیست! چه می‌توان کرد؟

در اینجا به این حقیقت اشاره می‌کنم که S.SNO و P.PNO به ترتیب در دامنه‌های SNO و PNO تعریف شده‌اند و بنا بر ادعای من دامنه‌ها همان نوع‌ها هستند. پس SNO و PNO *نوع‌های تعریف‌شده بوسیله کاربر* می‌باشند. احتمالا هر دو آنها به صورت فیزیکی توسط نوع

داخلی CHAR نمایش داده می‌شوند ولی می‌دانیم که نمایش فیزیکی به پیاده‌سازی مربوط است و نه به مدل. همانطور که در فصل یک دیدیم، این مساله به کاربر مربوط نیست و در واقع از چشم وی پنهان است (یا اینکه باید اینطور باشد). خصوصا عملگرهایی که برای کار با شماره قطعه و شماره توزیع‌کننده تعریف می‌شوند بایستی از نوع آنها باشند و نه نوع CHAR. برای مثال می‌توان دو رشته حرفی را بدنبال هم الحاق نمود ولی احتمالا نمی‌توان دو شماره توزیع‌کننده را به یکدیگر الحاق کرد (این کار فقط در صورتی ممکن است که یک عملگر به همین منظور برای نوع SNO تعریف شده باشد).

وقتی یک نوع تعریف می‌کنیم بایستی یک عملگر به نام *انتخابگر* نیز برای آن تعریف کنیم، تا ما را قادر به انتخاب و یا تعیین یک مقدار دلخواه برای نوع مورد نظر نماید.* مثلا انتخابگر نوع SNO (همانطور که در فصل ششم خواهیم دید می‌تواند فقط SNO نامیده شود) ما را قادر می‌سازد که مقدارهای SNO را از میان مقدارهایی که به صورت CHAR نمایش داده می‌شوند، انتخاب کنیم. برای مثال:

```
SNO('S1')
```

این عبارت انتخابگر SNO را به اجرا در می‌آورد و یک شماره توزیع‌کننده خاص را بازمی‌گرداند. توزیع‌کننده‌ای که شماره آن با رشته حرفی 'S1' نمایش داده شده است. همچنین عبارت زیر:

```
PNO('P1')
```

این عبارت نیز با اجرای انتخابگر PNO شماره قطعه خاصی را برمی‌گرداند. قطعه‌ای که شماره آن با رشته کاراکتری 'P1' نمایش شده است. همانطور که مشاهده می‌کنید انتخابگرهای SNO و PNO در عمل کار تبدیل یک مقدار از نوع CHAR به مقادیری از نوع‌های PNO یا SNO را انجام می‌دهند.

به مقایسه S.SNO = 'X4' برگردیم. اتفاقی که می‌افتد این است که سیستم متوجه می‌شود که طرفین از یک نوع نیستند (دو نوع CHAR و SNO). با توجه به این که این دو از یک نوع نیستند، و مادامی که طرفین مقایسه هم‌نوع نباشند نمی‌توانند با هم مساوی باشند. در عین حال سیستم می‌داند عملگری وجود دارد–به نام انتخابگر SNO– که می‌تواند نوع CHAR را به

*این دیدگاه علیرغم نحوه رفتار SQL صحیح‌تر است. از آنجا که انتخابگرهای SQL تا حدودی آشفته‌اند، در مورد جزئیات آنها بحث نکرده‌ام و در اینجا و تمامی کتاب فرض می‌کنم که این عملگرها تعریف شده‌اند.

نوع SNO تبدیل کند. پس سیستم می‌تواند با اجرای *تلویحی* این عملگر مقایسه را به درستی انجام دهد. در عمل عبارت زیر جایگزین مقایسه اصلی می‌شود.

S.SNO = SNO('X4')

حال دو شماره توزیع‌کننده را مقایسه می‌کنیم. کاری که انجام آن مشروع است.

به همین ترتیب سیستم در عمل عبارت زیر را جانشین 'P.PNO = 'X4 می‌کند.

P.PNO = PNO('X4')

ولی در مورد مقایسه S.SNO = P.PNO هیچ عملگری برای تبدیل شماره قطعه به شماره توزیع‌کننده و یا بالعکس در سیستم وجود ندارد (دست‌کم فرض کنیم وجود ندارد). در نتیجه این مقایسه منجر به وقوع *خطای نوع* خواهد شد: عملگرها از یک نوع نیستند و هیچ راهی برای یکسان‌سازی نوع آنها وجود ندارد.

واژگان: برای تبدیل معمولا از اصطلاح *اجبار* استفاده می‌شود. در مثال اول رشته حرفی 'X4' به زور و اجبار به نوع SNO تبدیل شد و در مثال دوم همین رشته به PNO اجبار شد.

در رابطه با همین مثال عملگر دیگری که بایستی به هنگام تعریف SNO و PNO معرفی شود، *عملگر* _THE است. کار این عملگر تبدیل انواع SNO و PNO به رشته حرفی (و یا هر چیز دیگر) می‌باشد.* در این مثال فرض را بر این بگذارید که مبدل هر دو نوع SNO و PNO، THE_CHAR نامیده می‌شود. اگر واقعا نیازمند مقایسه S.SNO و P.PNO باشیم، تنها راهی که به نظر می‌رسد نمایش هر دو آنها بصورت رشته حرفی است و این کار می‌تواند بصورت زیر انجام پذیرد:

THE_CHAR (S.SNO) = THE_CHAR (P.PNO)

به عبارت دیگر تبدیل شماره توزیع‌کننده به رشته، تبدیل شماره قطعه به رشته و در نهایت مقایسه دو رشته.

مطمئنا می‌بینید که این ساز و کار به گونه‌ای موثر الف)بررسی مطابق دامنه را میسر می‌سازد و ب)در صورتی که چاره دیگری نبود از امکان نقض بررسی دامنه استفاده می‌کند. بعلاوه این کار را بصورت صریح، کاملا مستقل و با روشی چند منظوره انجام می‌دهد. در مقابل روش «نقض بررسی دامنه» یک کار ملموس و واقعی نیست و در واقع کاری بی‌معناست که باعث

*در این مورد هم رفتار SQL متفاوت است و اساسا اصطلاح _THE در آن بی‌معناست (در عمل انتخابگر هم در آن بی‌معناست).

اختلاط نوع‌ها با روش نمایش داده‌ها می‌شود (همانطور که قبلا گفته شد موضوع نوع‌ها به مدل مربوط می‌شود در حالی که نمایش داده‌ها مربوط به پیاده‌سازی است).

تاکنون احتمالا دریافته‌اید که صحبت من در موردی چیزی است که در حوزه زبان‌ها، *قویا نوع‌دار* نامیده می‌شود. نویسندگان مختلف تعریف‌های گوناگونی در مورد این کلمه ارائه کرده‌اند، ولی بطور کلی معنی آن این است که الف)هر چیز –مقدار یا متغیر– دارای نوع است. ب)هر کجا که بخواهیم عملگری را به اجرا در آوریم، سیستم درست بودن نوع عملوندها را برای انجام عملیات بررسی می‌کند.* این روال در مورد تمام عملگرها –نه فقط عملگرهای مقایسه‌ای– انجام می‌شود. تاکید بر مقایسه به هنگام طرح موضوع بررسی دامنه جنبه تاریخی دارد ولی این کار اشتباه است. برای نمونه عبارت‌های زیر را در نظر بگیرید:

```
P.WEIGHT * SP.QTY
P.WEIGHT + SP.QTY
```

اولی احتمالا مجاز است (این عبارت خود وزن دیگری را برمی‌گرداند: وزن کل سفارش را). در مقابل دومی احتمالا غلط است (جمع تعداد و وزن چه معنایی می‌تواند داشته باشد؟).

اتمی بودن مقدار داده‌ها

امیدوارم بخش قبل شما را قانع کرده باشد که دامنه‌ها دقیقا همان نوع‌ها هستند نه کمتر و نه بیشتر. اکنون می‌خواهم بحث را به *اتمی بودن داده‌ها* و ارتباط آن با *فرم اول نرمال* (بطور خلاصه 1NF) بکشانم. در فصل اول گفتم که 1NF به معنای آنست که هر ویژگی متعلق به هر تاپل رابطه باید یک مقدار واحد (البته از یک نوع خاص) داشته باشد و اضافه می‌کنم که این «مقدارهای واحد»، *اتمی* به حساب می‌آیند. برای ادامه این بحث باید به این سوال پاسخ داده شود؛ اتمی بودن داده‌ها به چه معناست؟

در فصل اول گفتم که کاد اتمی بودن داده‌ها را تعریف کرد. این تعریف می‌گوید: «بوسیله DBMS قابل تجزیه به قطعات کوچکتر نباشند (به استثنای برخی توابع خاص)» اگر از

* و یا آنها را به نوع درست اجبار می‌نماید. به دلایلی که مستقیما مربوط به این بحث نیست، من بر این عقیده‌ام که تمامی تبدیلات بایستی بطور صریح صورت گیرند. اجبار عامل بسیاری از خطاهاست.

استثنای داخل پرانتز هم صرف‌نظر کنیم این تعریف نادقیق و تا حدودی گیج‌کننده است. برای مثال آیا می‌توان گفت که رشته‌های حرفی اتمی هستند؟ تمام محصولاتی که من دیده‌ام عملگرهای متعددی برای کار با رشته‌ها مانند LIKE یا SUBSTR (زیر رشته) یا «||» (اتصال رشته) و غیره را می‌شناسند و به کاملا نشان می‌دهند که رشته‌های حرفی بوسیله DBMS قابل تجزیه‌اند. با این وجود آیا رشته‌های حرفی قابل تجزیه‌اند؟ شما چه فکر می‌کنید؟

چند مثال دیگر در رابطه با اتمی بودن ویژگی‌های مربوط به تاپل‌ها در برخی رابطه‌ها:

- اعداد صحیح می‌توانند به عامل‌های اولیه تجزیه شوند (می‌دانم که این نوع تجزیه در این بحث چندان مورد نظر نیست، فقط سعی دارم نشان دهم که خود تجزیه ممکن است صورت‌های مختلف داشته باشد).
- اعداد اعشاری (با تعداد رقم اعشار ثابت)، می‌توانند به دو بخش صحیح و اعشار تجزیه شوند.
- تاریخ و زمان می‌توانند به بخش‌های روز/ماه/سال و یا ثانیه/دقیقه/ساعت تجزیه شوند.

حال می‌خواهم به یک مثال مشکل‌تر بپردازیم. به شکل 2-1 نگاه کنید. رابطه R1 در این شکل کاهش یافته رابطه سفارش‌ها از مثال اجرایی ماست و فقط نشان می‌دهد که کدام توزیع‌کننده، کدام قطعه را تهیه می‌کند. طبق این مثال برای هر ترکیب صحیح SNO-PNO یک تاپل وجود دارد. شماره توزیع‌کننده و شماره قطعه را «اتمی» در نظر می‌گیریم، بنابراین احتمالا تایید می‌کنید که R1 یک رابطه 1NF است.

R1

SNO	PNO
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

R2

SNO	PNO
S2	P1, P2
S3	P2
S4	P2, P4, P5

R3

SNO	PNO_SET
S2	{P1, P2}
S3	{P2}
S4	{P2, P4, P5}

شکل 2-1. رابطه‌های R1، R2 و R3

حال فرض کنید به جای R1 از R2 استفاده می‌کنیم تا نشان دهیم که هر توزیع‌کننده *گروه* مشخصی از قطعات را توزیع می‌کند (ویژگی PNO در R2 *چند مقداری* است و مقدارهای موجود در این ویژگی نشان دهنده شماره گروهی از قطعات است). اکثر افراد با دیدن R2 می‌گویند که این رابطه 1NF نیست. در حقیقت این مثال «گروه تکرار شونده» را نشان می‌دهد و تقریبا همه موافق‌اند که در گروه‌های تکرار شونده 1NF از بین رفته است (چرا که گروه‌ها اتمی نیستند).

به این نتیجه می‌رسیم که R2، 1NF نیست. در ادامه این بحث فرض کنید که R3 را جایگزین R1 کرده‌ایم. من ادعا می‌کنم که *R3، 1NF است!*[*] برای بحث:

- اولا، توجه کنید که من –عمدا– ویژگی نام PNO_SET را انتخاب کرده‌ام و گروهی از شماره قطعه‌ها را درون آکولاد قرار داده‌ام تا نشان دهم هر یک از گروه‌ها در واقع یک مقدار واحد دارد: یک مقدار مجموعه‌ای. مجموعه در سطح بالایی از تجرید قرار دارد و یک مقدار واحد به حساب می‌آید.
- ثانیا (صرفنظر از اینکه در مورد استدلال اول چه عقیده‌ای دارید): حقیقت این است که مجموعه‌ای مانند {P2,P4,P5} نه بیشتر از یک رشته حرفی توسط DBMS قابل تجزیه است و نه کمتر از آن. مجموعه‌ها همچون رشته‌های حرفی دارای یک ساختار داخلی هستند که می‌توان آن را برای مقاصد خاصی می‌توان شکست. به زبان دیگر اگر یک رشته کاراکتری شرایط لازم برای 1NF بودن را داراست –یعنی رشته کاراکتری اتمی است– پس مجموعه‌ها هم چنین خاصیتی را دارا هستند.

نکته‌ای که اکنون به آن می‌رسیم این است که اتمی بودن معنای کاملا مشخصی ندارد (اتمی مطلق وجود ندارد) و این امر وابسته به کاری است که ما می‌خواهیم با داده‌ها انجام دهیم. گاهی اوقات با مجموعه‌ای از تمام شماره قطعه‌ها به عنوان یک چیز واحد سروکار داریم و گاهی می‌خواهیم با یکی از شماره قطعاتی که درون مجموعه است، به تنهایی کار کنیم که در این

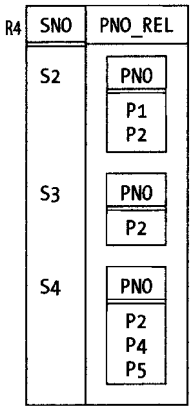
[*] من ادعا ندارم که این طراحی به خوبی انجام شده است.–به راستی شاید چنین نباشد– بحث بر سر چیز دیگری است. چیزی که در اینجا مهم است این است که چه کاری مجاز است نه اینکه چه کاری خوب است. طراحی R3 مجاز است.

صورت به سطح پایینتری از جزئیات تنزل کرده‌ایم (سطح پایینتری از تجرید). این مقایسه ممکن است به فهم مطلب کمک کند. در فیزیک (که لغت اتمی بودن از آنجا آمده است) وضعیت کاملا مشابهی وجود دارد: گاهی به اتم‌های منفرد بعنوان یک چیز واحد نگاه می‌شود و گاهی می‌خواهیم در مورد پروتون‌ها، نوترون‌ها و الکترون‌ها که اجزای تشکیل دهنده اتم هستند صحبت می‌کنیم. به همین ترتیب پروتون و نوترون‌ها هم منفرد نیستند و از اجزایی بنام کوارک تشکیل شده‌اند و احتمالا این روند ادامه دارد.*

بیایید کمی دیگر به R3 بپردازیم. در شکل 2-1 مقادیر PNO_SET را بصورت مجموعه‌های عمومی نشان دادم ولی اگر دقیقتر و به صورت رابطه به آنها نگاه شود، این مثال سودمندتر خواهد بود (شکل 2-2 را ببینید که در آن نام ویژگی به PNO_REL تغییر یافته است). چرا این تغییر موجب بهبود شده است؟ چون رابطه‌ها –نه مجموعه‌های عمومی– تمامی آن چیزی است که مدل رابطه‌ای در مورد آن است.† در نتیجه جبر قدرتمند رابطه‌ای در اختیار رابطه یاد شده قرار می‌گیرد و گزینش، پرتو، پیوند و غیره برای آن قابل استفاده خواهند بود. در مقابل اگر از مجموعه‌های عمومی بجای رابطه‌ها استفاده نماییم نیازمند تعریف عملگرهای جدید (اجتماع مجموعه‌ای، اشتراک مجموعه‌ای و غیره) خواهیم بود. پس چه بهتر که از عملگرهایی که داریم استفاده کنیم!

* بنده در انجام یک پروژه پژوهشی دقیقا با همین مشکل مواجه شدم. در طراحی‌ام یکی از ویژگی‌ها از ترکیب دو رشته حرفی به وجود آمده بود و داوران عقیده داشتند به همین علت رابطه نرمال نیست. من تمام مطالب این قسمت کتاب را بعنوان استدلال مطرح کردم ولی نپذیرفتند و در نهایت این استدلال ساده موثر واقع شد که چون در هیچ مرحله‌ای سیستم پایگاه داده اقدام به تجزیه این مقدار نمی‌کند –با وجود آنکه ذاتا تجزیه‌پذیر است– بنابراین باید آن را بعنوان ویژگی اتمیک بپذیرید. همانطور که در علم شیمی ما هرگز اتمی را تجزیه نمی‌کنیم ولو آنکه همان اتم در علم فیزیک هسته‌ای تجزیه‌پذیر است. مترجم.

† تعجب نکنید. تفاوت اساسی مجموعه‌های عمومی که می‌توانند شامل همه چیز باشند با رابطه‌ها که فقط شامل تاپل‌ها هستند در همین است. توجه داشته باشید که رابطه‌ها همانند مجموعه‌های عمومی ممکن است فقط یک عضو داشته باشند.



شکل 2-2. رابطه R4 (اصلاح شده R3)

ویژگی PNO_REL در شکل 2-2 نمونه‌ای است از *ویژگی با مقدار رابطه* (RVA). دامنه مورد نظر هم دارای مقدار رابطه‌ای است (یعنی مقدارهای آن رابطه‌اند). من ادامه صحبت در مورد RVAها را به فصل 5 و 7 واگذار می‌کنم و در اینجا فقط این نکته را متذکر می‌شود که SQL از آنها پشتیبانی نمی‌کند. (و بطور دقیقتر از هیچ چیزی مشابه RVA یعنی *ستون‌های با مقدار جدول* نیز پشتیبانی نمی‌کند و عجیب اینجاست که از این دو پشتیبانی می‌کند. الف) ستون‌هایی که مقدارشان آرایه باشد. ب) ستون‌هایی که می‌تواند مقدارشان یک «چندمجموعه‌ای از سطرها» باشد. *چند مجموعه‌ای* -که به نام *خورجین* هم شناخته می‌شود- شبیه یک مجموعه است با این تفاوت که اعضای آن می‌توانند تکراری باشد. چندمجموعه‌ای‌ها از برخی جهات مشابه «ستون‌های با مقدار جدول» هستند. ولی این دو با هم برابر نیستند چرا که عملگرهای معمولی SQL نمی‌توانند بر روی آنها عمل کنند.)

من مثال قبل را عمدا و به دلیل مقدارهای خاص آن انتخاب کردم. رابطه‌های دارای RVA مشابه رابطه‌های دارای گروه‌های تکرار شونده به نظر می‌رسند و می‌دانیم که گروه‌های تکرار شونده جایی در دنیای رابطه‌ای ندارند. ولی من می‌توانم مثال‌های متعددی برای نشان دادن نقطه نظرم ارائه کنم: من می‌توانم ویژگی‌هایی(و در نتیجه دامنه‌هایی) را نشان دهم که شامل آرایه‌ها، توده‌ها، لیست‌ها، عکس‌ها، صدا و تصویر، تصاویر رادیولوژی، آثار انگشت، متون XML و یا مقادیر دیگری را ذکر کنم که بسته به نگاه شما می‌توانند «اتمی» یا «غیراتمی» باشند. ویژگی‌ها و در نتیجه دامنه‌ها می‌توانند پذیرای هر چیزی (هر *مقداری*) باشند. این بحث طولانی

نشان میدهد که یک سیستم «شی‌گرا–رابطه‌ای» واقعی چیزی بیشتر یا کمتر از یک سیستم رابطه‌ای واقعی نیست. سیستمی که از مدل رابطه‌ای با تمامی جزئیات پشتیبانی می‌کند. نکته اصلی سیستم‌های «شی‌گرا–رابطه‌ای» این است ویژگی‌ها، رابطه‌هایی با درجه دلخواه از پیچیدگی باشند. شاید بهتر باشد که این گونه بگوییم: یک سیستم شی‌گرا–رابطه‌ای خوب، یک سیستم رابطه‌ای است که از نوع‌ها به خوبی پشتیبانی کند و این تعریف یک سیستم *رابطه‌ای* خوب است. نه بیشتر و نه کمتر.

نوع چیست؟

در ادامه من کلمه *نوع* را بجای *دامنه* به کار خواهم برد. نوع دقیقا یعنی چه؟ بطور خلاصه نوع، *یک مجموعه نامدار و متناهی از مقدارها است*[*]، تمامی مقدارهای ممکن برای یک چیز خاص: مثل تمامی اعداد صحیح ممکن، تمامی رشته‌های کاراکتری ممکن، تمامی شماره توزیع‌کنندگان ممکن، تمامی متن‌های XML ممکن، یا تمامی رابطه‌های ممکن با عنوان مشخص و غیره. علاوه بر این:

- هر مقدار دارای یک نوع است، دقیقا یک نوع مگر زمانی که در خاصیت ارث‌بری در آن نوع پشتیبانی شود و این کتاب تاب بحث پیرامون چنین موضوعی را ندارد. توجه داشته باشید که نوع‌ها *جدا از هم* و *بدون همپوشانی* هستند. (برای درک بهتر: یکی از بازبین‌های کتاب گفت قطعا نوع‌های حیوانات خونگرم و چهارپایان همپوشانی دارند. واقعا هم چنین است ولی من گفتم اگر همپوشانی داشته باشیم به دلایلی وارد قلمرو ارث‌بری –در واقع ارث‌بری *چندگانه*– شده‌ایم و این دلایل و موضوع ارث‌بری –چه رابطه‌ای و چه غیر رابطه‌ای– با موضوع این بحث ارتباط چندانی ندارد و من قصد ندارم که در این کتاب در مورد آن صحبت کنم.)
- هر متغیر، هر ویژگی، هر عملگر که نتیجه‌ای را باز می‌گرداند، و هر یک از پارامترهای یک عملگر، همگی دارای نوع هستند. برای مثال متغیر V از نوع T

[*] متناهی به این دلیل که ما با کامپیوترهایی با امکانات متناهی سروکار داریم. همچنین در مورد نام‌دار توجه داشته باشید که نوع‌هایی با نام‌های متفاوت با هم متفاوت‌اند.

تعریف شده است بدین معنا که هر مقدارv که به V نسبت داده می‌شود باید از نوع T باشد.

- هر عبارت، یک مقدار را برمی‌گرداند و از این رو دارای نوع است. نوع عبارت وابسته به خارجی‌ترین عملگر عبارت است. (منظور آخرین عملگری است که عمل می‌کند) برای مثال نوع عبارت $(A+B)*(X-Y)$ وابسته به نوع عملگر "*" می‌باشد.

این واقعیت که پارامترها دارای نوع هستند بطور کلی مورد اشاره قرار گرفت ولی هنوز این مطلب را به صورت کامل تشریح نکرده‌ام: این واقعیت که هر نوع دارای مجموعه‌ای از عملگرها برای عمل بر روی تعدادی متغیر و مقدار است، مورد بحث است. مثلا اعداد صحیح دارای عملگرهای ریاضی معمول هستند؛ تاریخ و ساعت عملگرهای محاسباتی خود را دارند؛ متون XML چیزی بنام عملگرهای XPath دارند، رابطه‌ها عملگرهای جبر رابطه‌ای را دارند و تمامی نوع‌ها دارای عملگرهای(":=") جهت انتساب و ("=") برای مقایسه هستند. پس هر سیستمی که از تعریف نوع پشتیبانی می‌کند حتما راهی برای تعریف عملگر نیز در اختیار کاربران قرار می‌دهد چرا که نوع‌ها بدون داشتن عملگر، به هیچ دردی نمی‌خورند.

دانستن این نکته نیز مهم است که مقدارها و متغیرها فقط و فقط با عملگرهایی که بر روی نوع آنها تعریف شده‌اند، کار می‌کنند. برای مثال نوع عدد صحیح را که بوسیله سیستم (و نه کاربر) تعریف شده است را در نظر بگیرید:

- سیستم عملگر انتساب ":=" را که برای نسبت دادن مقدارهای عددی صحیح به متغیرهای عدد صحیح را دارد.
- همچنین دارای عملگرهای مقایسه‌ای "="، "<" و... برای مقایسه مقدارهای عددی صحیح است.
- و نیز دارای عملگرهای ریاضی "+"، "×" و... برای انجام عمیات ریاضی روی مقدارهای عددی صحیح است.
- ولی عملگرهای رشته‌ای "||"(اتصال رشته‌های حرفی) و SUBSTR (زیر رشته) و... را برای عمل بر روی مقدارهای عددی صحیح *ندارد*. بیان دیگر: عملگرهای رشته‌ای روی مقدارهای عددی کار نمی‌کنند.

در مقابل نوع SNO که بوسیله کاربر تعریف شده است قطعا دارای عملگرهای انتساب و مقایسه (":="و"="و"≠"و احتمالا"<" و...) می‌باشد. با این وجود ممکن است "+"،" ×" و... بر روی آن تعریف نشده باشد و معنای آن این است که از عملیات ریاضی بر روی شماره توزیع‌کننده پشتیبانی نمی‌شود (چرا ممکن است نیاز باشد که دو شماره توزیع‌کننده را جمع یا ضرب کنیم؟).

از آنچه گفته شد معلوم می‌شود که تعریف هر نوع جدید دست‌کم شامل تمامی موارد زیر می‌شود:

1- تعیین یک نام برای نوع (بدیهی است)
2- تعیین مقدارهایی که در این نوع می‌گنجند. این موضوع با جزئیات بیشتر در فصل ششم مورد بحث قرار گرفته است.
3- تعیین نحوه نمایش فیزیکی مقدارهای این نوع. همانطور که قبلا گفتم این موضوع به پیاده‌سازی مربوط است و نه به مدل، و من قصد ندارم در این کتاب به آن بپردازم.
4- تعیین عملگرهایی که بر مقدارها و متغیرهای این نوع اعمال می‌شوند (بحث بعدی)
5- تعیین نوع نتیجه، در مورد عملگرهایی که مقدار بر می‌گردانند. (مجددا بحث بعدی)

از بندهای 4 و 5 نتیجه می‌گیریم که سیستم دقیقا می‌داند که کدام عبارت‌ها مجاز هستند و همچنین می‌داند که نتیجه هر عبارت از چه نوعی خواهد بود.

برای مثال تصور کنید که یک نوع تعریف شده بنام POINT داریم که نمایش دهنده نقطه هندسی در یک فضای دو بعدی است. تعریف آنرا با توتوریال‌دی می‌بینید.* عملگر REFLECT نقطه P با مختصات دکارتی (x,y) را می‌گیرد و وارون آن با مختصات $(-x,-y)$ را برمی‌گرداند:

```
1   OPERATOR REFLECT ( P POINT ) RETURNS POINT ;
2   RETURN ( POINT ( - THE_X ( P ) , - THE_Y ( P ) ) ) ;
3   END OPERATOR ;
```

* از SQL هم میتوانستم به این منظور استفاده کنم ولی عملگر تعریف در SQL دارای جزئیاتی است که نمی‌خواهم در اینجا وارد آن شوم.

توضیح:

- خط 1 نشان می‌دهد که نام عملگر REFLECT است و فقط یک پارامتر P و از نوع POINT را دریافت می‌کند و نتیجه‌ای از نوع POINT را هم برمی‌گرداند (بنابراین عملگر از نوع POINT تعریف شده است).
- خط 2 حاوی کد مربوط به پیاده‌سازی عملگر است. عملگر دارای فقط یک مقدار برگشتی است که نوع آن POINT است که بوسیله عملگر *انتخابگر* POINT بدست آمده است که دو آرگومان X و Y را دارد. هر یک از این آرگومان‌ها درگیر اجرای عملگر *THE_* هستند. این اجرا X و Y مربوط به P را می‌دهند و منفی کردن آنها ما را به نتیجه مورد نظر می‌رساند.
- خط 3 نشان دهنده پایان تعریف است.

در اینجا منظور من بیشتر نوع‌های تعریف شده بوسیله کاربر بوده‌اند. نوع‌های تعریف شده در سیستم هم مشابه آنها هستند با این تفاوت که بجای این که بوسیله کاربر ساخته شوند، از قبل درون سیستم مهیا هستند. مثلا اگر INTEGER یک نوع تعریف شده در سیستم باشد، آنگاه سیستم نام آن، مقدارهای مجاز، پیاده سازی پوشیده و عملگرهای آنرا معین می‌کند. البته برای کسی که فقط استفاده کننده است تفاوتی ندارد که یک نوع بوسیله شخص دیگری ساخته شده باشد و یا از اول درون سیستم موجود باشد.

من احساس نمی‌کنم که نیازی باشد بیش از این وارد جزئیات تعریف نوع شویم چرا که این مطالب ربط زیادی به مدل رابطه‌ای ندارند.

فرق بین نوع‌های اسکالر و نوع‌های غیراسکالر

این تفکر وجود دارد که نوع‌ها *اسکالر* و یا *غیراسکالر* هستند. با یک تعریف تقریبی می‌توان گفت یک نوع اسکالر است اگر هیچ جزء قابل مشاهده‌ای توسط کاربر نداشته باشد و در غیر این صورت غیراسکالر است. مقدارها، متغیرها، ویژگی‌ها، عملگرها، پارامترها و عبارات‌هایی که از نوع T هستند اسکالر (و یا غیراسکالر) هستند در صورتی که نوع T اسکالر (و یا غیراسکالر) باشد. مثلا:

- نوع INTEGER اسکالر است پس مقدارها، متغیرها و.... که از این نوع باشند همگی اسکالراند و هیچ جزء قابل مشاهده‌ای برای کاربر ندارند.
- نوع‌های تاپل و رابطه غیراسکالراند و جزء قابل مشاهده آنها ویژگی‌های‌شان است. بنابراین مقدارها، متغیرهایی که از نوع تاپل و یا رابطه باشند، غیراسکالر خواهند بود.

بر این نکته تاکید می‌کنم که تعریف فوق کاملا غیر رسمی است. دیدیم که موضوع *اتمی‌بودن* به صورت مطلق معنایی ندارد. «اسکالربودن» هم همین موضوع را با نام دیگری مطرح می‌کند. بنابراین مدل رابطه‌ای در هیچ موردی با موضوع اسکالر بودن و نبودن، به طور رسمی برخورد نکرده است. در این کتاب من به زبان غیر رسمی اکتفا کرده‌ام، *اسکالر* را برای نوع‌هایی که نه رابطه و نه تاپل هستند به کار برده‌ام و *غیراسکالر* را به نوع‌هایی که تاپل یا رابطه هستند اطلاق کرده‌ام. حال نگاهی به یک مثال بیاندازیم. در اینجا تعریف توتوریال‌دی مربوط به مرابطه S (توزیع کنندگان) را ملاحظه می‌کنید:

```
1  VAR S BASE
2     RELATION {SNO SNO, SNAME NAME, STATUS INTEGER,
CITY CHAR}
3   KEY { SNO } ;
```

توضیح:

- کلمه کلیدی VAR در خط 1 نشان می‌دهد که در حال تعریف متغیر هستیم و کلمه کلیدی BASE به این معناست که متغیر از نوع مرابطه پایه می‌باشد.
- خط 2 نوع متغیر را مشخص می‌کند. کلمه کلیدی RELATION نشان می‌دهد که متغیر از نوع رابطه است و ادامه سطر مجموعه ویژگی‌هایی مشخص می‌شوند که در کنار هم، عنوان رابطه را می‌سازند. (همانطور که از فصل اول به یاد دارید یک ویژگی شامل زوج نام ویژگی و نوع ویژگی است). بدیهی است که این نوع غیراسکالر است و ترتیب ویژگی‌ها هیچ چیز را نشان نمی‌دهد.
- خط 3 {SNO} را به عنوان کلید کاندید این مرابطه معرفی می‌کند.

این مثال نکته دیگری را هم روشن می‌کند. این نوع:

```
RELATION {SNO SNO, SNAME NAME, STATUS INTEGER, CITY
CHAR}
```

نمونه‌ای از نوع‌های *ساختگی* است. بطور کلی یک نوع ساختگی نوعی است که از طریق اجرای *نوع‌ساز* به وجود می‌آید. (در این مثال سازنده نوع RELATION) می‌توانید نوع‌ساز را به صورت یک عملگر خاص تصور کنید. خاص به این دلیل که الف)یک نوع را برمی‌گرداند و نه (مثلا) یک مقدار اسکالر را، ب) فراخوانی آن در زمان کامپایل است و نه زمان اجرا. برای مثال بیشتر زبان‌های برنامه‌نویسی از نوع‌سازی بنام ARRAY پشتیبانی می‌کنند که کاربر را قادر به تعریف انواع گوناگونی از نوع آرایه می‌نماید. تنها نوع‌سازهایی که در راستای اهداف این کتاب مورد نیازاند عبارتند از TUPLE و RELATION. اینجا مثالی را از فراخوانی نوع‌ساز تاپل می‌بینید:

```
VAR SINGLE_SUPPLIER
    TUPLE { STATUS INTEGER, SNO SNO, CITY CHAR, SNAME
NAME } ;
```

مقدار متغیر SINGLE_SUPPLIER در هر لحظه از زمان یک تاپل با عنوان یکسان با مرابطه S خواهد بود. (من عمدا ترتیب ویژگی‌ها را تغییر دادم تا نشان دهم این ترتیب اهمیتی ندارد.) اگر تصور کنیم که این کد چند قسمت شده است اول رابطه تک تاپلی (تصور کنید رابطه فقط یک تاپل دارد که مربوط به S1 است.) را از مقدار فعلی مرابطه S استخراج می‌کند و سپس یک تاپل را از رابطه تک تاپل استخراج می‌کند و در نهایت تاپل را به متغیر SINGLE_SUPPLIER منتسب می‌کند. در توتوریال‌دی:

```
SINGLE_SUPPLIER := TUPLE FROM ( S WHERE SNO = SNO('S1') )
```

به هرحال توجه داشته باشید که تاپل t و رابطه r که فقط همین یک تاپل t را دارد با هم یکی نیستند. آنها از دو نوع مختلف‌اند. t از نوع تاپل و r از نوع رابطه (گرچه عنوان هر دو باهم یکی است).

توجه

نمی‌خواهم دچار سوتفاهم شوید. به این دلیل که متغیری مانند SINGLE_SUPPLIER ممکن است مورد نیاز برخی برنامه‌هایی باشد که به پایگاه توزیع‌کنندگان و قطعات دسترسی دارند، *نمی‌تواند* گفت که چنین متغیری می‌تواند خودش درون پایگاه داده‌ها وجود داشته باشد. پایگاه داده رابطه‌ای فقط یک نوع متغیر دارد، متغیر رابطه‌ای (مرابطه). مرابطه تنها نوع متغیری است که در پایگاه مجاز است. من در فصل هشتم و در مبحثی تحت عنوان *اصل اطلاعات* مجددا به این موضوع خواهم پرداخت.

فقط یک نکته دیگر درباره تاپل، رابطه و نوع باقی مانده است: به رغم اینکه نوع رابطه دارای اجزا قابل مشاهده توسط کاربران است (یعنی ویژگی‌هایش)، هیچ اظهار نظری در مورد نحوه ذخیره سازی فیزیکی آنها صورت نمی‌گیرد. در حقیقت نمایش فیزیکی نوع‌ها بایستی از دید کاربر پنهان بماند همانطور که در مورد نوع‌های اسکالر چنین است.

خلاصه

اشتباه رایجی که در مورد مدل رابطه‌ای وجود دارد این است که تصور می‌شود مدل رابطه‌ای فقط با نوع‌های ساده‌ای همچون اعداد، رشته‌ها و احتمالا تاریخ و ساعت سروکار دارد و دیگر هیچ. در این فصل سعی کرده‌ام شما را از این اشتباه درآورم. ویژگی‌های رابطه‌ها می‌توانند از هر *نوعی* باشند. در هیچ کجای مدل رابطه‌ای تعیین نشده است که این نوع‌ها باید چه باشند و در حقیقت نوع‌ها می‌توانند هر قدر که ما بخواهیم پیچیده باشند (بجز موردی که ذکر می‌کنم). به بیان دیگر این سوال که سیستم از چه نوع‌هایی پشتیبانی می‌کند، و این که آیا سیستم از مدل رابطه‌ای پشتیبانی می‌کند، دو سوال مستقل هستند. و یا (با دقت کمتر و بیان شیواتر): *نوع‌ها از جدول‌ها مستقل‌اند*.

من همچنین یادآوری کردم که این قضیه راهی برای فرار از رعایت فرم اول نرمال نیست. فرم اول نرمال فقط به این معناست که درون هر یک از ویژگی‌های هر تاپل رابطه، فقط یک مقدار تکی (از نوع متناسب) وجود دارد. اکنون می‌دانیم که نوع‌ها می‌توانند هر چیزی باشند و همچنین می‌دانیم تمامی رابطه‌ها طبق تعریف در فرم اول نرمال هستند.

و در نهایت، در مقدمه این فصل گفتم که در مورد این موضوع که نوع‌ها می‌توانند هر چیزی باشند دو استثنا بسیار مهم وجود دارد. اول -که برای آموزش آن را ساده کرده‌ام- اگر رابطه r از نوع T باشد آنگاه هیچیک از ویژگی‌های این رابطه نمی‌توانند از نوع T باشند (در این مورد فکر کنید!). دوم اینکه هیچیک از ویژگی‌های رابطه نبایستی از نوع اشاره‌گر باشند. همانطور که احتمالا می‌دانید پایگاه‌هایی که قبل از مدل رابطه‌ای وجود داشتند پر از اشاره‌گر بودند، و دسترسی به چنین پایگاه‌هایی مستلزم تعقیب اشاره‌گرها بود. مسلما در مورد چنین پایگاه‌هایی برنامه‌نویسی بسیار پراشتباه و دسترسی مستقیم کاربر نهایی به پایگاه غیر ممکن بود. هدف کاد از ارائه مدل رابطه‌ای، دوری از این گونه مشکلات بود و الحق که در این کار موفق شد.

تمرین‌ها

1- نوع چیست؟ چه تفاوتی بین نوع و دامنه وجود دارد.

2- از کلمه *انتخابگر* چه فهمیدید؟

3- *عملگر* _*THE* چیست؟

4- نمایش فیزیکی همیشه از دید کاربر پنهان است: درست یا غلط؟

5- موشکافی کنید: فرق آرگومان و پارامتر، فرق پایگاه داده‌ها و DBMS، فرق نوع ساختگی و نوع غیرساختگی، فرق اسکالر و غیراسکالر، فرق نوع و نمایش، فرق نوع تعریف شده بوسیله کاربر و نوع تعریف شده در سیستم، فرق عملگر تعریف شده بوسیله کاربر و عملگر تعریف شده در سیستم.

6- از کلمه *اجبار* چه می‌فهمید؟ چرا اجبار بد است؟

7- چرا «نقض بررسی دامنه» قابل توجیه نیست؟

8- نوع‌ساز چیست؟

9- *فرم اول نرمال* را تعریف کنید.

10- فرض کنید X یک عبارت باشد. منظور از نوع X چیست؟ اینکه X حتما دارای یک نوع *باشد*، چه اهمیتی دارد؟

11- تعریف عملگر REFLECT در این فصل را الگو قرار دهید و در توتوریال‌دی عملگری تعریف کنید که یک عدد صحیح را دریافت می‌کند و مکعب (به توان سه) آنرا برمی‌گرداند.

12- با استفاده از توتوریال‌دی عملگری تعریف کنید که نقطه‌ای با مختصات دکارتی x و y را دریافت کند و نقطه دیگری با مختصات $f(x)$ و $g(y)$ را برگرداند. عملگرهای f و g از قبل در سیستم تعریف شده‌اند.

13- مثالی از نوع رابطه بزنید. تفاوت بین نوع رابطه، مقدار رابطه و متغیر رابطه‌ای را بیان کنید.

14- مرابطه‌های P و SP از پایگاه توزیع‌کنندگان و قطعات را با استفاده از SQL یا توتوریال‌دی (و یا هر دو) تعریف کنید. اگر از هر دو استفاده کردید، چند تفاوت SQL یا توتوریال‌دی را عنوان کنید. مقصود از اینکه (مثلا) مرابطه P دارای از یک نوع رابطه‌ای مشخص است، چیست؟

15- مطابق نوع‌های داده شده در فصل مقدمه کتاب برای پایگاه توزیع‌کنندگان و قطعات، کدامیک از مقدارهای اسکالر زیر مجاز هستند؟ برای آنها که مجازاند نوع عبارت و برای آنها که نیستند، عبارتی که بطور صحیح این کار را برای ما انجام می‌دهد را بنویسید.

الف) CITY = 'London'

ب) SNAME || PNAME

ج) QTY × 100

د) QTY + 100

ه) STATUS + 5

و) 'ABC' < CITY

16- گاهی پیشنهاد می‌شود که بهتر است نوع‌ها متغیر باشند. مثلا شماره پرسنلی کارمندان ممکن است هنگام گسترش سازمان از سه رقم به چهار رقم افزایش یابد و در این هنگام لازم است در «مجموعه تمام شماره پرسنلی‌های ممکن» تجدید نظر شود. در این مورد بحث کنید.

17- نوع مجموعه‌ای از مقدارها است و تهی هم یک مجموعه قانونی است. پس ما می‌توانیم نوع *خالی* را تعریف کنیم که مجموعه مقدارهای آن خالی است. آیا می‌توانید کاربردی برای این نوع تصور کنید؟

18- در این فصل گفتم که عملگر مقایسه‌ای تساوی "=" در تمامی نوع‌ها وجود دارد. (با این وجود معنای آنرا شرح ندادم، اگر *v1* و *v2* دو مقدار باشند آنگاه *v1=v2* مقدار صحیح خواهد داشت اگر و فقط اگر مقدارهای *v1* و *v2* کاملا مشابه باشند). همانطور که با جزئیات بیشتر در فصل 8 خواهم گفت، SQL نه نیازی به عملگر "=" برای تمامی نوع‌ها دارد و نه استفاده از این مفهوم را توصیه می‌کند. دلیل این امر را توضیح دهید.

19- تمرین قبل را می‌توان بصورت دیگری مطرح کرد، *v1=v2* مقدار صحیح را برمی‌گرداند اگر و تنها اگر هر چه باشد عملگر *Op* ، اجرای *Op* بر روی *v1* و *v2* اثر یکسانی داشته باشد. ولی این عبارت هم یکی دیگر از قواعد SQL را نقض می‌کند. آیا می‌توانید مثالی در این مورد بزنید؟ دلیل آن را شرح دهید؟

20- چرا اشاره‌گرها جایی در مدل رابطه‌ای ندارند؟

21- *اصل انتساب* – که ساده و در عین حال اساسی است- می‌گوید که بعد از انتساب مقدار *v* به متغیر *V*، مقایسه *V=v*، مقدار صحیح را برخواهد گرداند. این امر هم یک حکم SQL را نقض می‌کند. (زیاد اتفاق می‌افتد) آیا می‌توانید مثالی در این مورد ارائه دهید؟ دلیل آن را شرح دهید؟

22- آیا تصور می‌کنید که نوع‌ها «متعلق به» پایگاه داده‌ها هستند؟ مرابطه‌ها چطور؟

23- در قسمت «مقایسه با دامنه تحمیلی»، یک عبارت SELECT_FROM_WHERE از SQL که محتوی عبارت دیگری بصورت تودرتو است را نمایش دادم، حال هر SELECTی می‌تواند جایگزین *SELECT شود. ولی "*SELECT" دارای مشکلاتی اساسی است و به همین دلیل من سعی کرده‌ام تا حد ممکن از آن اجتناب کنم. تا جایی که می‌توانید این اشکالات را بیان کنید. آیا می‌توانید به مشکلات دیگری از این دست که در SQL وجود دارند اشاره کنید؟

24- در اولین مثال از SELECT زبان SQL در این فصل، اشاره کردم که نیازی به سمیکالن پایانی وجود ندارد چرا که SELECT یک عبارت است نه یک دستور. تفاوت این دو در چیست؟

فصل سوم

تفاوت

تاپل‌ها و رابطه‌ها

با مطالعه دو فصل اول بایستی –دست‌کم از دیدگاه غیر رسمی– به درک مناسبی از تاپل‌ها و رابطه‌ها رسیده باشید. حال می‌خواهم این مفاهیم را دقیق‌تر تعریف کنم و از این تعریف دقیق به نتایج جدیدی برسم. شاید بهتر است اخطار بدهم که این تعریف‌ها ممکن است کمی ترسناک به نظر برسند ولی در مقایسه با به سایر مفاهیم رسمی هیچ چیز غیرعادی ندارند. تاپل و رابطه مفاهیم سرراستی هستند و شما بایستی حداقل یک بار با تعریف رسمی آنها روبرو شوید تا به درستی با آنها آشنا شوید.

تاپل چیست؟

آیا این یک تاپل است؟

SNO | SNO	SNAME | NAME	STATUS | INTEGER	CITY | CHAR
S1	Smith	20	London

نه، البته که نیست. این فقط *عکس* یک تاپل است. (فقط همین یک بار در تصویر من نوع‌ها ویژگی‌ها را در کنار نام آنها نوشته‌ام). همانطور که در فصل یک دیدیم بین خود یک چیز و تصویر آن تفاوت وجود دارد و این تفاوت می‌تواند بسیار مهم باشد. مثلا ویژگی‌های تاپل ترتیبی از راست به چپ ندارند، و این یک حسن (یا عیب؟) است. شکل زیر همان تاپل را نشان می‌دهد.

STATUS | INTEGER	SNAME | NAME	CITY | CHAR	SNO | SNO
20	Smith	London	S1

پس استفاده از تصویر برای درک بهتر است. به یاد داشته باشید که آنها *فقط* تصویر هستند و ممکن است مطالب نادرستی را به شما القا کنند.

پس از این تذکر اکنون می‌توانم بگویم مفهوم تاپل دقیقا عبارت است از:

تعریف: اگر داشته باشیم $T_1, T_2, \ldots, T_n$ و $n <= 0$ که هر کدام یک نام نوع باشند که لزوما یکتا (غیر تکراری) نیستند و متناظر با هر Ti یک نام ویژگی لزوما یکتا A_i وجود داشته باشد. هر یک از n زوج (نام‌ویژگی:نوع‌ویژگی) یک *ویژگی* خواهد بود. متناظر با هر ویژگی یک مقدار v_i از نوع T_i وجود دارد. هر یک از این n (ویژگی:مقدار) یک *جزء* است. مجموعه تمام n جزء که آنرا t می‌نامیم یک *مقدار تاپلی* (یا بطور خلاصه *تاپل*) بر روی ویژگی‌های $A_1, A_2, \ldots, A_n$ است. مقدار n *درجه* t است. تاپلی با درجه یک *یونری*، با درجه دو *باینری*، با درجه سه *ترنری*،... و تاپلی با درجه n، n-*اری* نامیده می‌شود. مجموعه تمامی n ویژگی، *عنوان* t است.

برای مثال و مطابق دو تصویر قبل (تاپل مربوط به توزیع‌کننده S1)، داریم:

درجه

چهار. عنوان هم از درجه چهار است.

نام نوع‌ها

SNO, SNAME, INTEGER و CHAR

نام ویژگی‌های متناظر

SNO, SNAME, STATUS و CITY

مقدارهای ویژگی‌های متناظر

SNO('S1'), NAME('Smith'), 20 و 'London'

توجه

من اکثر این ویژگی‌ها را بصورت ساده شده در تصویر نشان داده‌ام (و این روش را در تمام این کتاب ادامه خواهم داد). اما این دقیقا درست نیست که بگوییم مقدار SNO در تاپل فقط 'S1' (و یا با شلختگی بیشتر فقط S1) است. مقداری از نوع SNO، مقداری است از نوع SNO و نه مقداری از نوع CHAR! و SNO('S1') مقداری ثابت (لیترال) از نوع SNO است. (بطور دقیق‌تر، *حاصل اجرای انتخابگر SNO* همان‌گونه که در فصل دوم دیدیم. با بیانی مختصر و مفید، یک مقدار ثابت حاصل اجرای انتخابگری است که تمام آرگومانهای آن، خودشان مقدارهای ثابت هستند.)

عنوان

ساده‌ترین کار در اینجا نمایش یک شکل دیگر است:

SNO	SNO	SNAME	NAME	STATUS	INTEGER	CITY	CHAR

این شکل نشان دهنده یک مجموعه است و همانطور که می‌دانیم ترتیب اعضای یک مجموعه دلخواه است. در اینجا تصویر دیگری از همان عنوان را می‌بینید:

STATUS	INTEGER	SNAME	NAME	CITY	CHAR	SNO	SNO

تمرین: چند تصویر مختلف می‌توانیم برای نمایش این عنوان می‌توانیم رسم کنیم. (*جواب*: $4\times3\times2\times1=24$)

تاپل یک مقدار است؛ و مثل همه مقدارها دارای نوع است (همان‌طور که در فصل دوم دیدیم) و مثل همه نوع‌ها این نوع هم دارای یک نام است. در توتوریال‌دی چنین اسامی بصورت TUPLE{H} نوشته می‌شود که در آن {H} یک عنوان مشخص است. در مثال ما این نام عبارت است از:

TUPLE { SNO SNO, SNAME NAME,STATUS INTEGER, CITY CHAR }

(ترتیب ویژگی‌ها دلخواه است). تکرار می‌کنم تاپل یک مقدار است. مثل همه مقدارها، از این رو بایستی با اجرای انتخابگر بازگردانده شود.(حال که مقدار از نوع تاپلی است، انتخابگر تاپل). در اینجا اجرای انتخابگر تاپل را در مورد مثال خودمان مشاهده می‌کنید (باز هم در توتوریال‌دی):

```
TUPLE { SNO SNO('S1'), SNAME NAME('Smith'),
        STATUS 20, CITY 'London' }
```

(ترتیب جزءها دلخواه است). مشاهده می‌کنید که در توتوریال‌دی هر جزء با زوج (نام‌ویژگی:مقدارویژگی) مشخص می‌شود و نوع حذف شده است چرا که همواره می‌تواند از مقدار ویژگی بدست‌آید.

توجه

در مورد نحوه نوشتن: همانطور که می‌بینید کلمه کلیدی TUPLE در توتوریال‌دی دو کاربرد متفاوت دارد. این کلمه هم برای اجرای انتخابگر تاپل و هم برای نام نوع تاپل بکار می‌رود. این امر در مورد کلمه کلیدی RELATION نیز صدق می‌کند. (بزودی در قسمت رابطه چیست خواهید دید.)

و در پایان یک کلام هم از SQL. البته SQL بجای تاپل از سطر پشتیبانی می‌نماید؛ و به بیان دقیق‌تر از نوع سطری، سازنده نوع سطری و سازنده مقداری سطری پشتیبانی می‌کند که تا حدودی شبیه نوع تاپل، نوع‌ساز تاپل و انتخابگر تاپل است که این شباهت حتی در بهترین حالت بسیار تقریبی است.

```
ROW ( 1, 2 )
```

در این مثال سازنده مقدار سطری الف)دارای اجزایی بی‌نام است. ب)همان سطری که سازنده سطری ROW (2,1) می‌سازد را نشان نمی‌دهد (تفاوت‌های دیگری نیز بین سطرهای SQL و تاپل‌ها وجود دارد که در حال حاضر از محدوده بحث ما خارج است).

توجه

استفاده از کلمه کلیدی ROW در سازنده مقدار سطری SQL اختیاری است.

چند دست‌آورد مهم

اکنون می‌خواهم برخی دست‌آوردهای تعریف فوق را برشمرم. اول اینکه: *هیچ تاپلی محتوی هیچ تهی‌ای نیست*. چرا که طبق تعریف هر یک از ویژگی‌های تاپل دارای مقدار است (متناسب با نوع) و در فصل اول دیدیم که تهی مقدار نیست.* اگر تاپل‌ها دارای مقدار تهی نباشند، هیچ رابطه‌ای هم شامل مقدار تهی نخواهد بود. پس ما حداقل یک دلیل رسمی برای نفی تهی داریم. و اما در ادامه این فصل و در بخش «چرا تهی منع می‌شود» دلایل واقع بینانه‌تری در این مورد خواهم آورد.

دست‌آورد بعدی: *هر زیرمجموعه از یک تاپل، خود یک تاپل است و هر زیرمجموعه از عنوان، خود یک عنوان است*. برای مثال در مورد تاپل همیشگی خودمان که مربوط به توزیع‌کننده S1 است، ممکن است «مقدار {SNO,CITY}» را نیاز داشته باشیم. این تاپل درون تاپل دیگری وجود دارد.

SNO ¦ SNO	CITY ¦ CHAR
S1	London

ساختار عنوان و نوع این تاپل بدین گونه است:

TUPLE { SNO SNO, CITY CHAR }

در نتیجه این هم یک تاپل است.(از نوع TUPLE {SNO SNO}) :

SNO ¦ SNO
S1

اگر بخواهیم به *مقدار* واقعی یک ویژگی دست پیدا کنیم (مثل SNO('S1')) بایستی آن را به طریقی از درون تاپل بیرون بکشیم. توتوریال‌دی از ساختار زیر برای این منظور استفاده می‌نماید.

SNO FROM *t*

* با این وجود SQL معمولا (و نه همیشه) از مقدارهای تهی صحبت می‌کند.

(t هر عبارتی است، نشانه یک تاپل، که ویژگی‌ای به نام SNO در آن وجود دارد). SQL از نشانگر نقطه هم استفاده می‌کند: t.SNO

توجه

در فصل دوم دیدیم که تاپل t و رابطه r که تنها دارای تاپل t است با هم مساوی نیستند. به همین ترتیب مقدار v و تاپل t که فقط شامل مقدار v است هم یکی نیستند. این دو به خصوص از نظر نوع با یکدیگر متفاوت‌اند.

مطمئنم که می‌دانید مجموعه خالی(تهی) زیرمجموعه همه مجموعه‌هاست. به همین ترتیب تاپلی با عنوان خالی –و مجموعه‌ای تهی از ویژگی‌ها– یک تاپل معتبر است. رسم تصویر چنین تاپلی بر روی کاغذ قدری مشکل است و من هم سعی نمی‌کنم این کار را انجام دهم. یک تاپل با عنوان خالی از نوع TUPLE{} (و بدون ویژگی) می‌باشد. ما برخی اوقات برای تاکید بر صفر بودن درجه، آن را *تاپل صفر* و گاهی هم *تاپل خالی* می‌نامیم. ممکن است تصور کنید چنین تاپلی در دنیای واقعی استفاده چندانی ندارد. ولی اتفاقا این موضوع اهمیت ویژه‌ای دارد. من در این فصل و در بخش «DUM و DEE» در این مورد بیشتر خواهم گفت.

آخرین «دستاورد مهم» که قصد دارم در اینجا در مورد آن صحبت کنم مربوط به *تساوی تاپل‌ها* است. (یادآوری از فصل دوم، عملگر تساوی «=» برای تمام نوع‌ها از جمله نوع تاپل تعریف شده است.) اساسا دو تاپل مساوی‌اند اگر و تنها اگر با هم دقیقا یکسان باشند (مانند دو عدد صحیح که با هم مساوی‌اند اگر و تنها اگر با هم دقیقا یکسان باشند). ولی بهتر است مفهوم تساوی تاپل‌ها را دقیق‌تر تعریف کنیم چرا که در مدل رابطه‌ای بسیاری مفاهیم بدان وابسته‌اند. برای مثال کلید کاندید، کلید خارجی، و اکثر عملگرهای جبری رابطه‌ای با استفاده از آن تعریف می‌شوند. این تعریف مختصر و مفید چنین است:

تعریف: تاپل‌های t_1 و t_2 مساوی‌اند اگر و تنها اگر دارای ویژگی‌های یکسانی در $A_1, A_2,..., A_n$ باشند. به زبان دیگر هم‌نوع باشند و هر چه باشد i، ($i = 1, 2,..., n$) مقدار v_1 موجود در A_i واقع در t_1 مساوی مقدار v_2 موجود در A_i واقع در t_2 باشد.

همچنین (بدیهی است باید باز هم گفته شود)، دو تاپل در یک رابطه تکراری‌اند، اگر و تنها اگر با هم مساوی باشند.

نتیجه فوری تعریف فوق این است که تمامی تاپل صفرها تکرار یکدیگر هستند. به همین دلیل ما می‌توانیم به جای اصطلاح «یک تاپل صفر» از «تاپل صفر» (بصورت عام) استفاده کنیم و اغلب هم همین کار را انجام می‌دهیم.

و در نهایت اینکه عملگرهای مقایسه‌ای > و < در مورد تاپل‌ها به کار نمی‌روند (تمرین 11 انتهای این فصل را ببینید).

رابطه چیست؟

من از رابطه توزیع‌کنندگان خودمان بعنوان مثال اصلی این بخش استفاده می‌کنم. این تصویر را ببینید:

SNO ¦ SNO	SNAME ¦ NAME	STATUS ¦ INTEGER	CITY ¦ CHAR
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

و این تعریف را:

تعریف: فرض کنید $\{H\}$ یک عنوان باشد و همچنین $t_1, t_2,..., t_m$ برای $m \geq 0$ تاپل‌هایی متمایز و همگی با عنوان $\{H\}$ باشند. به تلفیق $\{H\}$ و مجموعه تاپل‌ها $\{t1, t2,...tm\}$ یک *مقدار رابطه‌ای* (یا بطور خلاصه *رابطه*) –مثلا **r**– بر روی ویژگی‌های $A_1, A_2,...A_n$ می‌گویند. که $A_1, A_2,...A_n$ ویژگی‌های $\{H\}$ هستند. *عنوان* r، $\{H\}$ است. r همان ویژگی‌ها (و از این رو همان نام‌ها و نوع‌ها) و همان درجه عنوان را داراست. بدنه r مجموعه‌ای از تاپل‌هاست و m *کاردینالیتی* r است.

من تفسیر رابطه توزیع‌کنندگان از نقطه نظر تعریف فوق را به عنوان یک تمرین به شما واگذار می‌کنم. اما روشن می‌کنم که چرا به چنین چیزی رابطه می‌گویند. اساسا هر تاپل در رابطه نماینده یک ارتباط nتایی است (مثلا ارتباط مقدارهای S1، اسمیت، 20 و لندن با همدیگر). به زبان ساده، یک مجموعه با n مقدار (که هر مقدار یک ویژگی تاپل است) داریم. مجموعه تمام تاپل‌های یک رابطه خاص که نماینده تمام ارتباط‌های موجود در یک لحظه از زمان هستند، به

زبان ریاضی رابطه نامیده می‌شود. بنابراین «تفسیری» که بارها شنیده‌اید مبنی بر اینکه مدل رابطه‌ای چنین نام‌گذاری شده است زیرا ما را قادر به برقراری «ارتباط جدول‌ها با یکدیگر» می‌نماید، ناشی از عدم فهم نکته اساسی موضوع می‌باشد. مدل رابطه‌ای به این دلیل چنین نامیده می‌شود که در سطح بالایی از تجرید قرار دارد به گونه‌ای که می‌توانیم در آن رابطه‌های ریاضی را به صورت «جدول‌های» معمولی تصور کنیم.

اکنون می‌دانیم که رابطه هم –مانند تاپل– خود یک مقدار است و نوع دارد و نوع آن هم نام دارد. در توتوریال‌دی این نام‌ها به صورت RELATION{*H*} هستند که {H} عنوان می‌باشد. به مثال زیر توجه کنید:

```
RELATION { SNO SNO, SNAME NAME, STATUS INTEGER, CITY
CHAR }
```

(ترتیب نوشتن ویژگی‌ها تفاوتی ندارد.) هر مقدار رابطه‌ای می‌تواند با اجرای انتخابگر رابطه بازگردانده شود. برای مثال:

```
RELATION {
    TUPLE { SNO SNO('S1'), SNAME NAME('Smith'),
                STATUS 20, CITY 'London' } ,
    TUPLE { SNO SNO('S2'), SNAME NAME('Jones'),
                STATUS 10, CITY 'Paris'  } ,
    TUPLE { SNO SNO('S3'), SNAME NAME('Blake'),
                STATUS 30, CITY 'Paris'  } ,
    TUPLE { SNO SNO('S4'), SNAME NAME('Clark'),
                STATUS 20, CITY 'London' } ,
    TUPLE { SNO SNO('S5'), SNAME NAME('Adams'),
                STATUS 30, CITY 'Athens' } }
```

ترتیب تاپل‌ها هم دلخواه است.

SQL تا چه حد این مفاهیم را پشتیبانی می‌کند؟ تقریبا هیچ. در حقیقت SQL اصلا چیزی به نام نوع رابطه ندارد. یک جدول SQL برای داشتن چند *سطر* طراحی شده است (یک *مجموعه چندتایی* یا خورجین از سطرها) که طبیعتاً از نوع سطر هستند (البته چیزی به نام نوع جدول نام‌دار هم در آن وجود دارد که هیچ شباهتی به نوع رابطه ندارد و من هم قصد ندارم آن را

در اینجا شرح دهم). در نتیجه هیچ چیز به عنوان سازنده نوع RELATION وجود ندارد و مشابه آن سازنده جدول را در اختیار دارد و به همین ترتیب در مورد انتخابگر رابطه. یک مثال:

VALUES (1, 2), (2, 1), (1, 1), (1, 2)

این عبارت بعنوان یک جدول با چهار –و نه سه!– سطر و دو ستون (بدون نام) ارزیابی می‌شود.

دست‌آوردهای مهم دیگر

بیشتر ویژگی‌های رابطه که در فصل اول در مورد آنها صحبت کردم نتیجه مستقیم تعاریف بخش قبل هستند. ولی دو ویژگی دیگر هستند که تاکنون در مورد آنها صحبت نکرده‌ام و در اینجا قصد دارم آنها را موشکافی کنم.

اول اینکه هر زیرمجموعه از بدنه خود یک بدنه است. (تقریبا به این معنا که، هر زیرمجموعه از رابطه خود یک رابطه است). از آنجا که تهی زیرمجموعه تمام مجموعه‌هاست پس یک رابطه می‌تواند بدنه‌ای داشته باشد که حاوی مجموعه‌ای خالی از تاپل‌ها است (و ما چنین رابطه‌ای را *رابطه خالی* می‌نامیم). مثلا فرض کنید که هم اکنون هیچ سفارشی وجود ندارد. چنین چیزی را می‌توان به این صورت نشان داد.*

SNO	PNO	QTY

توجه داشته باشید که برای هر نوع خاص از رابطه، دقیقا یک رابطه خالی وجود دارد ولی رابطه‌های خالی از انواع مختلف با هم مساوی نیستند، دقیقا به همین دلیل که نوع‌شان متفاوت است. مثلا رابطه خالی توزیع‌کنندگان با رابطه خالی قطعات مساوی نیست. بدنه آنها مساوی است ولی عنوان آنها چنین نیست.

و دومین نکته که عمدا تاکنون در مورد آن صحبت نکرده‌ام. فصل اول را به یاد بیاورید، زمانی که یک رابطه بصورت یک جدول ترسیم شود واقعا جدول *نیست* (یکبار دیگر می‌گویم

* می‌دانم این بحث در حالی صورت می‌گیرد که نام نوع‌ها را از عنوان –بصورت غیر رسمی– حذف کرده‌ایم. در ادامه این کتاب بسته به موقعیت و منظوری که دارم هر وقت که بخواهم آنها را ذکر می‌کنم.

تصویر یک چیز با خود آن یکی نیست). البته *تصور* رابطه‌ها بصورت جدول راحت‌تر است. در این حال جدول‌ها «دوستانه»تر هستند و ما می‌توانیم بصورت غیر رسمی رابطه‌ها را بصورت جدول تصور کنیم –اکثرا جدول‌های مسطح یا دوبعدی–. چنین کاری فهم و استفاده از سیستم‌های رابطه‌ای را آسان‌تر و رفتار آنها را قابل درک‌تر می‌کند. به زبان دیگر خصوصیت بسیار خوب مدل رابطه‌ای این است که زیربنای ساختار داده‌ای آن –رابطه– دارای یک نمایش تصویری جذاب و قابل درک است.

متاسفانه ظاهر زیبا چشم بسیاری را کور کرده است به گونه‌ای که تصور می‌کنند رابطه‌ها همانطور که به نظر می‌رسد «مسطح» و «دوبعدی» هستند در حالی که چنین نیست. اگر r رابطه‌ای با n ویژگی باشد آنگاه هر تاپل r نشان‌دهنده یک نقطه در فضایی n بعدی است (هر ویژگی دارای یک دامنه بوده و یک رابطه مجموعه‌ای از چنین نقاطی است). برای نمونه هر یک از پنج تاپل موجود در رابطه شناخته شده S که مربوط به توزیع‌کنندگان است، نشان‌دهنده یک نقطه مهم در فضایی چهاربعدی است و بطور کلی می‌توان این رابطه را چهاربعدی به حساب آورد. بنابراین رابطه‌ها nبعدی هستند نه دوبعدی![*] همانطور که در جاهای دیگر هم نوشته‌ام (در بسیاری موارد و در واقع): *بیایید به خودمان قول بدهیم که هیچگاه دوباره لفظ «رابطه‌های مسطح» را به کار نبریم.*

اکنون به سوی موضوعاتی می‌روم که قصد موشکافی آنها را دارم. سه موضوع که عبارتند از: تاپل‌های تکراری، تهی و رابطه‌ای که هیچ ویژگی‌ای ندارد. من هر یک از این‌ها را در یک بخش توضیح می‌دهم.

چرا تاپل‌های تکراری غیرمجازند

دلایل عملی زیادی برای جلوگیری از تاپل‌های تکراری (یا بطور خلاصه تکرار) وجود دارد و در اینجا فقط می‌توانم در مورد یکی از آنها صحبت کنم که تصور می‌کنم مهمترین دلیل باشد. اما این موضوع نیازمند پیش‌نیازهایی است که تاکنون در این کتاب مطرح نشده‌اند. از همین این رو دو پیش فرض اولیه را عنوان می‌کنم.

[*] من عقیده دارم موضوع نیاز به «پایگاه‌های داده چندبعدی» (خصوصا جهت نرم‌افزارهای پشتیبانی از تصمیم)، بایستی مورد بررسی قرار گیرد. چرا که هنوز بسیاری تصور می‌کنند رابطه‌ها چندبعدی نیستند.

- فرض می‌کنم می‌دانید که DBMSهای رابطه‌ای قسمتی بنام بهینه‌ساز دارند که وظیفه آن تلاش برای ایجاد بهترین شکل از کوئری‌های کاربران است (اینجا بهترین یعنی بهترین کارایی).
- فرض می‌کنم این را هم می‌دانید که یکی از کارهای بهینه‌ساز بازنویسی کوئری‌ها است. در عمل بازنویسی، کوئری X1 که توسط کاربر وارد شده است به کوئری X2 تبدیل می‌شود به گونه‌ای که کوئری‌های X1 و X2 پس از اجرا دقیقا یک نتیجه را به بار می‌آورند ولی کوئری X2 نسبت به X1 سرعت و کارایی بهتری دارد (یا حداقل چنین انتظار می‌رود).

اکنون دلیل خود را عنوان می‌کنم. نکته بنیادی این است که تبدیل عبارت (و در نتیجه بهینه‌سازی) که در سیستم‌های رابطه‌ای مجاز است، در صورت وجود تکرار غیر مجاز می‌شود. برای مثال پایگاه داده (غیر رابطه‌ای) تصویر شده در شکل 3-1 را در نظر بگیرید.

P

PNO	PNAME
P1	Screw
P1	Screw
P1	Screw
P2	Screw

SP

SNO	PNO
S1	P1
S1	P1
S1	P2

شکل 3-1. یک پایگاه غیر رابطه‌ای با تکرار

قبل از ادامه مطلب سوالی می‌پرسم: وجود سه تا <P1,Screw> در جدول P چه معنایی می‌دهد و چه تفاوتی با دوتا، چهارتا یا هفده تا دارد؟* این موضوع بایستی بالاخره معنایی داشته باشد، و اگر ندارد اصلا چرا تکرار در اینجا وجود دارد. همانطور که فقط یک بار این جمله را از کاد شنیدم، «اگر حرفی درست باشد، دوباره گفتن آن را درست‌تر نمی‌کند.»

من فرض می‌کنم که تکرار بی‌دلیل نیست. این دلیل هر چه که باشد، نمی‌تواند به آسانی قابل تشخیص باشد. (قبلا بطور غیر مستقیم عنوان کردم که تکرار یکی از اهداف استفاده از مدل رابطه‌ای را زیرپا می‌گذارد: *وضوح* را. بدین معنا که داده‌ها باید تا آنجا که ممکن است واضح، صریح و بی نیاز از توضیح باشند. وقتی از پایگاه‌های داده مشترک صحبت می‌کنیم وجود تکرار به معنای پوشیده ماندن بخشی از محتوا خواهد بود). به بیان دیگر اگر این تکرار معنا دارد، چه واقعیتی را می‌رساند و چه تفاوتی بین دو، چهار یا هفده وجود دارد؟ و اگر معنا ندارد چرا اصلا وجود دارد؟

در اینجا برای کوئری «شماره قطعاتی که پیچ screw هستند یا توسط توزیع‌کننده S1 تهیه می‌شوند و یا هر دو حالت، را بده» چند عبارت SQL ذکر شده است. به خروجی‌های متفاوتی که از آنها بدست می‌آید توجه کنید.

```
1  SELECT P.PNO
   FROM   P
   WHERE  P.PNAME = NAME('Screw')
   OR     P.PNO IN
      ( SELECT SP.PNO
        FROM   SP
        WHERE  SP.SNO = SNO('S1') )

   Result: P1 * 3, P2 * 1.
```

* در اینجا از لغت جدول استفاده کرده‌ام زیرا چیزهایی که در موردشان صحبت می‌کنم نه رابطه هستند و نه متغیر رابطه‌ای. آنها کلید ندارند. (مشاهده کنید که در شکل 3-1 زیر هیچیک از نام ستون‌ها خط کشیده نشده است.)

```
2   SELECT SP.PNO
    FROM   SP
    WHERE  SP.SNO = SNO('S1')
    OR     SP.PNO IN
       ( SELECT P.PNO
         FROM   P
         WHERE  P.PNAME = NAME('Screw') )
```

Result: P1 * 2, P2 * 1.

```
3   SELECT P.PNO
    FROM   P, SP
    WHERE  ( SP.SNO = SNO('S1') AND
           SP.PNO = P.PNO )
    OR     P.PNAME = NAME('Screw')
```

Result: P1 * 9, P2 * 3.

```
4   SELECT SP.PNO
    FROM   P, SP
    WHERE  ( SP.SNO = SNO('S1') AND
           SP.PNO = P.PNO )
    OR     P.PNAME = NAME('Screw')
```

Result: P1 * 8, P2 * 4.

```
5   SELECT P.PNO
    FROM   P
    WHERE  P.PNAME = NAME('Screw')

    UNION  ALL
    SELECT SP.PNO
    FROM   SP
    WHERE  SP.SNO = SNO('S1')
```

Result: P1 * 5, P2 * 2.

```
6   SELECT DISTINCT P.PNO
    FROM   P
    WHERE  P.PNAME = NAME('Screw')
    UNION  ALL
    SELECT SP.PNO
    FROM   SP
    WHERE  SP.SNO = SNO('S1')
```

Result: P1 * 3, P2 * 2.

```
7   SELECT P.PNO
    FROM   P
    WHERE  P.PNAME = NAME('Screw')
    UNION  ALL
    SELECT DISTINCT SP.PNO
    FROM   SP
    WHERE  SP.SNO = SNO('S1')
```

Result: P1 * 4, P2 * 2.

```
8   SELECT DISTINCT P.PNO
    FROM   P
    WHERE  P.PNAME = NAME('Screw')
    OR     P.PNO IN
       ( SELECT SP.PNO
         FROM   SP
         WHERE  SP.SNO = SNO('S1') )
```

Result: P1 * 1, P2 * 1.

```
9   SELECT DISTINCT SP.PNO
    FROM   SP
    WHERE  SP.SNO = SNO('S1')
    OR     SP.PNO IN
       ( SELECT P.PNO
         FROM   P
         WHERE  P.PNAME = NAME('Screw') )
```

Result: P1 * 1, P2 * 1.

```
10  SELECT P.PNO
    FROM   P
    GROUP  BY P.PNO, P.PNAME
    HAVING P.PNAME = NAME('Screw')
    OR     P.PNO IN
       ( SELECT SP.PNO
         FROM   SP
         WHERE  SP.SNO = SNO('S1') )
```

Result: P1 * 1, P2 * 1.

```
11  SELECT P.PNO
    FROM   P, SP
    GROUP  BY P.PNO, P.PNAME, SP.SNO, SP.PNO
    HAVING ( SP.SNO = SNO('S1') AND
           SP.PNO = P.PNO )
    OR     P.PNAME = NAME('Screw')
```

Result: P1 * 2, P2 * 2.

```
12  SELECT P.PNO
    FROM   P
    WHERE  P.PNAME = NAME('Screw')
    UNION
    SELECT SP.PNO
    FROM   SP
    WHERE  SP.SNO = SNO('S1')
```

Result: P1 * 1, P2 * 1.

توجه

اعتبار این عبارت‌ها جدا مورد تردید است (کدامیک درست‌اند؟). چرا که آنها باید شامل هر پیچی باشند که حداقل به یک توزیع‌کننده سفارش داده شده‌اند ولی این کار به دلیل زیر به درستی انجام نگرفته است.

اولین نکته بدیهی این است که دوازده عبارت مختلف نُه نتیجه متفاوت به بار آورده‌اند و این تفاوت فقط به درجه *تکرار* آنها مربوط می‌شود (در این مورد کاری از دست کسی بر

نمی‌آید. این دوازده عبارت و نه نتیجه همگی یکسان هستند و تازه این همه حالات ممکن نیست). بنابراین اگر وجود تکرار واقعا برای کاربر اهمیت داشته باشد، برای رسیدن به یک نتیجه خاص بایستی زحمت و مشقت فراوانی را برای نوشتن یک کوئری تحمل نماید.

این مشکلات همچنین در مورد خود سیستم هم وجود دارند؛ از آنجا که عبارت‌نویسی‌های مختلف موجب نتایج متفاوت می‌شود و بهینه‌ساز هم به هنگام تبدیل کوئری‌ها دچار سردرگمی می‌شود. مثلا بهینه‌ساز حتی اگر چنین صلاح ببیند مجاز نیست عبارت 1 را به عبارت 3 تبدیل کند. به عبارت دیگر تکرار سطرها موجب از کار افتادن بهینه‌ساز می‌شود. در اینجا برخی نتایج این مساله آورده شده‌اند:

- بهینه‌ساز با دشواری زیادی به هنگام ایجاد یا پشتیبانی کدها روبرو می‌شود و احتمال وجود اشکال افزایش می‌یابد. این عوامل دست به دست هم می‌دهند تا محصولی با هزینه بیشتر و قابلیت اطمینان کمتر تولید گردد و با تاخیر زمانی بیشتر روانه بازار پر رقابت شود.
- کارایی سیستم از آن چیزی که می‌توانست باشد، کمتر خواهد شد.
- کاربران درگیر مسائل مربوط به کارایی می‌شوند. به بیان دقیق‌تر آنها وقت و نیروی خود را صرف محاسبه اینکه کدام روش نوشتن کوئری بیشترین کارائی را دارد خواهند نمود و در چنین شرایطی مدل رابطه‌ای غیر قابل استفاده می‌شود.

دلیلی که موجب از کار افتادن بهینه‌سازها می‌شود در بیشتر موارد بیهوده است. به احتمال زیاد برای کاربر اهمیتی ندارد که چند تکرار در نتیجه ظاهر شود. به بیانی دیگر:

- عبارت نویسی متفاوت موجب نتایج متفاوت می‌شود.
- ولی این تفاوت به احتمال زیاد از نظر کاربر بی‌اهمیت است.
- با این وجود بهینه‌ساز از این امر بی‌اطلاع است و از این رو محدودیتی غیر ضروری را رعایت می‌کند و نمی‌تواند تبدیل عبارات را آن گونه که مایل است، انجام دهد.

پس از ذکر این مثال‌ها، فکر کردم بهتر است نتیجه‌گیری دیگری را نیز مطرح کنم. شما باید مطمئن شوید که در نتیجه کوئری‌تان تکرار وجود ندارد. مثلا در کوئری‌های SQL همیشه از

DISTINCT استفاده نمایید و به سادگی خود را از شر این مشکل رها کنید. اگر از این توصیه پیروی کنید آنگاه اصلا دلیلی برای مجاز شمردن تکرار وجود نخواهد داشت.

استفاده از SELECT DISTINCT

در نسخه دست‌نویس کتاب نوشتم که اگر نوشتن DISTINCT در همه جا برای شما سخت است، غرولند خود را بجای من متوجه تولیدکنندگان محصولات SQL کنید. ولی تقریبا تمامی بازبین‌های کتاب از پیشنهاد استفاده همیشگی از DISTINCT، برآشفتند. یکی از آنها نوشت: «تمام کسانی که SQL را به خوبی می‌شناسند از تصور استفاده از SELECT-DISTINCT بصورت پیش‌فرض جا می‌خورند.» من قصد دارم مودبانه عرض کنم که اولا کسانی که «از این پیشنهاد جاخورده‌اند» احتمالا پیاده‌سازی را خوب می‌شناسند و نه SQL را. ثانیا جاخوردن آنها احتمالا به این دلیل است که پیاده‌سازی‌ها به دلیل استفاده نابجا از DISTINCT دچار افت کیفیت می‌شوند. اگر من بنویسم

SELECT DISTINCT S.SNO FROM S ...

می‌توان از این DISTINCT به صورت کاملا بی‌ضرر صرف‌نظر کرد. اگر بنویسم

IN (SELECT DISTINCT ...) و یا EXISTS (SELECT DISTINCT ...)

می‌توان از این DISTINCT به صورت کاملا بی ضرر صرف‌نظر کرد. اگر بنویسم

SELECT DISTINCT SP.SNO FROM SP ... GROUP BY SP.SNO

می‌توان از این DISTINCT به صورت کاملا بی ضرر صرف‌نظر کرد. اگر بنویسم

SELECT DISTINCT ... UNION SELECT DISTINCT

می‌توان از این DISTINCT به صورت کاملا بی ضرر صرف‌نظر کرد و.... چرا بایستی من –به عنوان یک کاربر– وقت و نیرویم را به هنگام استفاده از DISTINCT صرف محاسبه کارایی کنم آن هم در حالی حذف آن از نظر منطقی مجاز است و چرا باید جزئیات قواعد ضد و نقیض SQL را به خاطر بسپارم تا بدانم کجاها تکرارها بصورت خودکار حذف می‌شوند و کجاها نمی‌شوند؟ به هرحال من هنوز سر حرف خودم ایستاده‌ام ولی برای مساعدت به رابطه‌های خوب در ادامه این کتاب از توصیه من پیروی نکنید و درخواست حذف تکرار را فقط در جاهایی به کار ببرید که از نظر منطقی حضور آن لازم به نظر می‌رسد. تصمیم‌گیری در این مورد همیشه آسان نیست. حداقل الان تصور می‌کنم که صدای شکایتم را به گوش تولیدکنندگان محصولات رسانده‌ام.

نکات بسیاری در این مورد وجود دارد که من فقط به چهار نکته دیگر اکتفا می‌کنم. اولا، نقطه مقابل SELECT_DISTINCT در زبان SQL دستور SELECT_ALL است که بدبختانه پیش فرض می‌باشد. مشکل اینجاست که SELECT_DISTINCT احتمالا به زمانی اجرای طولانی‌تری از SELECT_ALL نیاز دارد حتی اگر حضور DISTINCT واقعا هیچ اثری نداشته باشد. در این مورد توضیح بیشتری نمی‌دهم بجز اینکه سیستم‌های SQL به دلیل حمایت نکردن از *ارث‌بری کلید* نوعا قابلیت بهینه‌سازی مناسب برای ادغام تکرارها را ندارند (ضعف آنها این است که نمی‌توانند کلید یک عبارت رابطه‌ای دلخواه را پیدا کنند).

ثانیا، شما ممکن است –چندان نامعقول نیست– مدعی باشید که جدول‌های پایه در دنیای واقعی فاقد تکرار هستند و از این رو مثال من نمی‌تواند چیزی را نشان دهد. ولی SQL ممکن است خود *تولیدکننده* تکرار در نتایج کوئری‌ها باشد. عبارت‌نویسی‌های متفاوت برای یک کوئری ممکن است نتایجی با درجات تکرار متفاوت را به بار آورد حتی اگر جدول‌های ورودی فاقد تکرار باشند. برای مثال در پایگاه توزیع‌کنندگان و قطعات به دو عبارت زیر که برای کوئری «شماره توزیع‌کنندگانی که حداقل یک قطعه را توزیع نموده‌اند را بده» نوشته شده‌اند، توجه کنید:

```
SELECT S.SNO                 |  SELECT S.SNO
FROM   S                     |  FROM   S, SP
WHERE  S.SNO IN              |  WHERE  S.SNO = SP.SNO
   ( SELECT SP.SNO           |
     FROM   SP )             |
```

(با استفاده از مقدارهای همیشگی این دو عبارت چه نتیجه‌هایی ایجاد خواهند کرد؟) اگر نخواهید تصاویر شکل 3-1 جدول‌های پایه باشند، و خروجی این کوئری‌ها را بعنوان ورودی کوئری‌های بعدی در نظر بگیرید. ادامه تحلیل همانند مثال قبل خواهد بود.

ثالثا، یک دلیل روان‌شناسی وجود دارد که تصور می‌کنم متقاعد کننده باشد (از جاناتان جنیک برای این نکته تشکر می‌کنم): اگر مطابق بخش قبل رابطه‌ها را بصورت nبعدی در نظر بگیریم یک جدول بصورت نقاطی رسم شده در فضای nبعدی تصور می‌شود. پس تکرار چیزی را تغییر نمی‌دهد و صرفا به معنای دو بار رسم یک نقطه در یک مکان از فضا خواهد بود.

رابعا، تصور کنید در جدول T تکرار مجاز باشد. آنگاه ما نمی‌توانیم تفاوت بین تکرارهای «واقعی» و تکرارهایی که بر اثر اشتباه هنگام ورود اطلاعات بوجود آمده‌اند، را تشخیص دهیم!

مثلا چه اتفاقی می‌افتد اگر متصدی وارد کردن اطلاعات به اشتباه یک سطر را دوبار وارد کند (چیزی که زیاد اتفاق می‌افتد). آیا راهی برای حذف «دومی» بدون حذف «اولی» وجود دارد؟ توجه داشته باشید که قصد داریم «دومی» ای را حذف کنیم که اصلا نبایستی وارد می‌شد.

چرا تهی غیر مجاز است

اولین پاراگراف بخش قبل، اینجا هم مورد استفاده قرار گرفته است (فقط با یک تغییر کوچک): دلایل عملی زیادی برای جلوگیری از تهی وجود دارد و در اینجا فقط می‌توانم در مورد یکی از آنها صحبت کنم که تصور می‌کنم مهمترین دلیل باشد. ولی طرح این موضوع نیازمند پیش نیازهایی است که تاکنون در این کتاب مطرح نشده‌اند. از همین این رو دو پیش فرض اولیه را عنوان می‌کنم.

- فرض را بر این می‌گذارم که شما می‌دانید مقایسه‌ای که یکی از طرفین یا هر دو طرف آن تهی باشند از نظر صحیح یا غلط بودن، نامعلوم ارزیابی می‌شود. دلیل این امر آن است که تهی *مقداری نامعلوم* است. اگر مقدار A نامعلوم باشد، آنگاه بدیهی است درستی عباراتی مثل $A>B$ هم بدون توجه به مقدار B،نامعلوم است (حتی اگر B هم نامعلوم باشد). این واقعیت منشا *منطق سه‌حالته* (3VL) است. وجود تهی به ناچار ما را به سمت منطقی می‌برد که در آن سه حالت درستی (بجای دو حالت معمول) وجود دارد (مدل رابطه‌ای بر اساس منطق شناخته شده دوحالته 2VL بنا شده است).
- فرض می‌کنم جداول درستی منطق 3VL را برای عملگرهای شناخته شده NOT، AND و OR را می‌شناسید (T برای صحیح، F برای غلط و U برای نامعلوم). همانطور که مشاهده می‌کنید NOT نامعلوم برمی‌گرداند اگر ورودی آن نامعلوم باشد. AND نامعلوم برمی‌گرداند اگر یک ورودی آن نامعلوم باشد و ورودی دیگر صحیح یا نامعلوم باشد و OR نامعلوم برمی‌گرداند اگر یک ورودی آن نامعلوم باشد و ورودی دیگر غلط یا نامعلوم باشد.

p	NOT p
T	F
U	U
F	T

p	q	p AND q
T	T	T
T	U	U
T	F	F
U	T	U
U	U	U
U	F	F
F	T	F
F	U	F
F	F	F

p	q	p OR q
T	T	T
T	U	T
T	F	T
U	T	T
U	U	U
U	F	U
F	T	T
F	U	U
F	F	F

اکنون دلیل خود را ارائه می‌دهم. نکته بنیادی این است که عبارت‌های بولی معین–و به دنبال از آن کوئری‌های معین– نتایجی تولید می‌کنند که با منطق سه‌حالته مطابقت دارند ولی در دنیای واقعی درست نیستند. برای مثال مطابق پایگاه داده (غیر رابطه‌ای) قابل مشاهده در شکل 3–2، CITY برای قطعه P1 تهی است. توجه داشته باشید که فضای خالی موجود در قسمت CITY قطعه P1 به معنای هیچ چیز است. تصور کنید هیچ چیز، حتی یک رشته از کاراکتر بلنک یا یک رشته تهی در این مکان وجود ندارد (بدین معنا که «تاپل» مربوط به قطعه P1 یک تاپل واقعی نیست، نکته‌ای که قبل از پایان فصل به آن خواهم پرداخت).

S

SNO	CITY
S1	London

P

PNO	CITY
P1	

شکل 3–2. یک پایگاه داده‌های غیر رابطه‌ای دارای تهی

اکنون کوئری زیر که مربوط به پایگاه شکل 3–2 است را ببینید: «زوج‌های SNO و PNOای را بده که قطعه و توزیع‌کننده از شهرهای متفاوتی هستند و یا قطعه متعلق به شهر پاریس نیست (یا هر دو).» عبارت SQL این کوئری چنین است*:

```
SELECT S.SNO, P.PNO
FROM   S, P
WHERE  S.CITY <> P.CITY
OR     P.CITY <> 'Paris'
```

* به عنوان یک تمرین نتیجه این کوئری را برای داده‌های همیشگی (در شکل 3–1 از فصل 1) بدست آورید. توجه: علامت «<>» در SQL معادل «≠» است و متذکر می‌شوم که علامت های «=<» و «>=» هم معادل «≤» و «≥» هستند. چیزی که احتمالا می‌دانستید.

بیایید بر روی عبارت بولی موجود در WHERE تمرکز کنیم (پرانتزها برای وضوح اضافه شده‌اند):

(S.CITY <> P.CITY) OR (P.CITY <> 'Paris')

برای داده‌هایی که ما در این پایگاه داریم این عبارت بصورت «نامعلومORنامعلوم» ارزیابی می‌شود که به نامعلوم می‌رسد. کوئری‌های SQL داده‌هایی را بازمی‌گردانند که عبارت WHERE برای آنها صحیح ارزیابی شده باشد و نه غلط یا نامعلوم. پس در مثال قبل هیچ نتیجه‌ای برگردانده نمی‌شود.

ولی بالاخره P1 در دنیای واقعی دارای شهری هست. به بیان دیگر «تهی بودن CITY» برای قطعه P1 دلالت بر وجود یک مقدار واقعی می‌کند که آنرا x می‌نامیم. بدیهی است که x یا پاریس هست و یا نیست. اگر پاریس باشد آنگاه عبارت

(S.CITY <> P.CITY) OR (P.CITY <> 'Paris')

به این صورت در می‌آید (برای داده‌هایی که داریم):

('London' <> 'Paris') OR ('Paris' <> 'Paris')

این عبارت صحیح ارزیابی می‌شود چرا که اولین بخش آن صحیح است. همچنین اگر x پاریس نباشد آنگاه عبارت بصورت زیر در می‌آید (بازهم برای داده‌هایی که داریم):

('London' <> *xyz*) OR (*xyz* <> 'Paris')

این عبارت هم صحیح است چرا که بخش دوم آن صحیح است. پس این عبارت بولین همیشه صحیح است و کوئری بایستی S1-P1 را -صرفنظر از مقدار تهی در دنیای واقعی- برگرداند. با بیانی دیگر نتیجه‌ای که طبق این منطق (که 3VL است) بدست می‌آید با آنچه از دنیای واقعی بدست می‌آید تفاوت دارد!

و در مثالی دیگر مطابق این کوئری (من این مثال را خیلی مهم نمی‌دانم چرا که چیزی بیش از مثال قبل را مشخص نمی‌کند ولی در همان راستا نکته مورد بحث را روشن‌تر نشان می‌دهد):

```
SELECT P.PNO
FROM   P
WHERE  P.CITY = P.CITY
```

در دنیای واقعی جواب، مجموعه شماره تمامی قطعاتی است که در P وجود دارند (اگر شکل 3-2 ملاک باشد فقط P1). اما SQL هیچ شماره قطعه‌ای را برنمی‌گرداند.

جمع‌بندی: اگر شما درون پایگاه خود تهی داشته باشید، در جواب برخی کوئری‌ها به پاسخ نادرست خواهید رسید و هیچ راهی برای پیداکردن این کوئری‌ها وجود ندارد، بنابراین کل نتایج بدست آمده زیر سوال می‌روند. *هرگز نمی‌توان به نتایج بدست آمده از پایگاهی که دارای تهی است اعتماد کرد.* به نظر من این امر کاملا بازدارنده است.

می‌توانم همان معامله‌ای که در بخش قبل با «تکرار» کردم را در اینجا هم انجام دهم و دلایل دیگری بیاورم، ولی ترجیح می‌دهم این بحث را با دلایل *رسمی* بر ضد تهی خاتمه دهم. به یاد آورید که تهی مقدار نیست و به همین ترتیب:*

- نوعی که دارای تهی باشد اصلا نوع نیست (چرا که نوع محتوی مقدار است).
- تاپلی که دارای تهی باشد اصلا تاپل نیست (چرا که تاپل محتوی مقدار است).
- رابطه‌ای که دارای تهی باشد اصلا رابطه نیست (چرا که رابطه محتوی تاپل و تاپل محتوی مقدار است).

خلاصه اینکه اگر تهی باشد آنگاه چیزی برای گفتن درباره مدل رابطه‌ای نداریم، (من نمی‌دانم در مورد چه چیز حرف می‌زنیم اما می‌دانم آن چیز مدل رابطه‌ای نیست). همین. والسلام.

جدول DUM و جدول DEE

دو بخش قبل احتمالا کمی ناراحت کننده بودند ولی در عوض این بخش تا حدودی مفرح خواهد بود. بخش «چند دست‌آورد مهم» را به یاد آورید که گفته شد مجموعه خالی (تهی) زیرمجموعه تمام مجموعه‌هاست و چیزی بنام تاپل خالی (یا تاپل صفر) وجود دارد که عنوان آن تهی است. نکته‌ای که پیش از این توجه شما را به آن جلب نکردم –و البته به اندازه کافی بدیهی است– که یک رابطه می‌تواند دارای عنوان تهی باشد (عنوان مجموعه‌ای از صفات است و هیچ‌کس تا به حال نگفته است که مجموعه نمی‌تواند خالی باشد). چنین رابطه‌ای از نوع RELATION{} بوده، درجه آن صفر است. این رابطه *نالری* هم نامیده می‌شود، درست همانطور که رابطه‌ای با درجه دو باینری خوانده می‌شود (نالری هیچ ربطی به مقدار تهی یا نال ندارد. تهی همچنان چیز بدی است در حالی که نالری چیز خوبی است. با این وجود من ترجیح

* و با سختگیری کمتر در مورد «رابطه‌ها» و مواردی چون تکرار، ترتیب بالا به پایین تاپل‌ها یا ترتیب راست به چپ ویژگی‌ها می‌توان به همین صورت نتیجه‌گیری نمود.

می‌دهم این اصطلاح را بکار نبرم چون ممکن است بخاطر تشابه اسمی چنین به نظر برسد که مربوط به تهی است).

اگر r یک رابطه با درجه صفر باشد، چند رابطه با این شرایط وجود دارد. پاسخ این است که: فقط دو رابطه. اول رابطه r به گونه‌ای که کاملا خالی است و هیچ تاپلی ندارد. به یاد آورید که از هر نوع فقط یک رابطه خالی می‌توان داشت. دوم r به گونه‌ای که خالی نیست. در این صورت تاپل‌های آن فقط می‌توانند از نوع تاپل صفر باشند. ولی فقط یک تاپل صفر وجود دارد. به عبارت دیگر تاپل‌های صفر تکرار یکدیگر هستند و r نمی‌تواند بیش از یکی از آنها را داشته باشد. پس معلوم شد که فقط دو رابطه بی‌عنوان وجود دارد: یکی بدون تاپل و دیگری با یک تاپل. به دلایل بدیهی سعی نمی‌کنم تصویر این رابطه‌ها را رسم کنم. این یکی از جاهایی است که ایده تصویرسازی رابطه‌ها به صورت جدول با شکست مواجه می‌شود.

ممکن است به این فکر بیافتید که: «خب که چه؟ رابطه‌ای که هیچ ویژگی‌ای ندارد به چه درد می‌خورد.» حتی اگر چنین چیزی از نظر ریاضی مهم باشد (که هست)، ولی این امر هیچ اهمیتی از نظر کاربردی ندارد؟ واقعیت این است که این مفهوم‌ها باید از نظر کاربردی نیز دارای اهمیت شایانی باشند چرا که آن را نام‌گذاری کرده‌ایم و نام‌های TABLE_DUM و TABLE_DEE (یا بطور مختصر DUM وDEE) را روی آنها گذاشته‌ایم (DUM جدول خالی و DEE جدول دارای یک تاپل است).* اهمیت آنها در این است که DUM معادل غلط (یا خیر) و DEE صحیح (یا بله) می‌باشند. این‌ها از جمله بنیادی‌ترین مباحث هستند.

* اجازه دهید مطلبی را در مورد این نام‌گذاری بگویم. اولا جهت اطلاع خوانندگان غیر انگلیسی زبان، این کلمات حاصل شوخی با استفاده از کلمات Tweedledum و Tweedledee هستند. این‌ها دو شخصیت مربوط به یک داستان و ترانه کودکانه‌اند (مثل شنگول و منگول). دوما، شاید صدا زدن دو رابطه‌ای که حتی حاضر نیستند بصورت جدول نمایش داده شوند، با این نام‌ها کمی سبک باشند! ولی ما مدتهاست که آنها را به همین صورت صدا زده‌ایم و حالا هم قصد تغییر آن را نداریم.

توجه

من در مورد این رابطه‌ها در فصل بعد صحبت خواهم کرد. به هر حال برای از بر کردن آنها می‌توانید به یاد بسپارید که YES و DEE هر دو E دارند و NO و DUM هیچیک E ندارند.

من در این کتاب هنوز به مثال‌های مناسبی در مورد عملکرد DUM و DEE نپرداخته‌ام ولی در صفحات آینده مثال‌های زیادی را در این مورد به شما نشان خواهم داد ولی در این رویارویی زود هنگام، نکته‌ای را عنوان می‌کنم تا احساس مناسبی نسبت به این موضوع پیدا کنید: نقشی که DEE در جبر رابطه‌ای بازی می‌کند همانند نقشی است که در حساب عادی بر عهده صفر است. همه ما می‌دانیم که صفر در ریاضیات چه اهمیتی دارد. در حقیقت تصور ریاضیات بدون صفر بسیار مشکل است (رومی‌های باستان سعی در انجام این کار داشتند ولی پیشرفت زیادی نکردند). از همین روی تصور جبر رابطه‌ای بدون TABLE_DEE هم مشکل است.

خلاصه

در این فصل تعریفی جامع در مورد مفاهیم بنیادی تاپل و رابطه ارائه نموده‌ام. همانطور که در فصل مقدمه گفتم این تعریف‌ها ممکن است در ابتدا کمی ترسناک به نظر برسند ولی امیدوارم با خواندن دو فصل اول کتاب، آنها را فهمیده باشید. من همچنین در مورد نوع تاپل‌ها و نوع رابطه‌ها بحث کردم و مثال‌هایی از اجرای انتخابگر به شما نشان دادم و نتایجی که از این تعریف‌ها بدست می‌آید را مورد بحث قرار دادم:

- هیچ تاپل و در نتیجه هیچ رابطه‌ای نمی‌تواند دارای مقدار تهی باشد.
- هر زیر مجموعه از تاپل، یک تاپل است، هر زیرمجموعه از عنوان، یک عنوان است و هر زیر مجموعه از بدنه یک بدنه است (هر کدام از این زیرمجموعه‌ها می‌توانند خالی باشند).
- دو تاپل مساوی‌اند اگر از هر نظر یکسان باشند.

دیگر نتایجی که در فصل‌های قبل بیان شدند:

- در محل هر یک از ویژگی‌های یک تاپل دقیقا یک مقدار وجود دارد. در نتیجه تمام رابطه‌ها در فرم اول نرمال هستند.

- ویژگی‌های تاپل و در نتیجه رابطه از چپ به راست ترتیبی ندارند.
- تاپل‌های رابطه هم ترتیبی از بالا به پایین ندارند.
- رابطه‌ها هیچ‌گاه تاپل‌های تکراری ندارند.

همچنین مطالب واقع‌بینانه‌ای را در مورد غیرمجاز بودن تکرار و تهی بیان کردم و بحث را با بیان خلاصه‌ای در مورد جدول DUM و جدول DEE به پایان بردم. DUM به معنای نه یا غلط و DEE به معنای بله یا صحیح است.

تمرین‌ها

1- اصطلاحات زیر را به مختصرترین شکل ممکن شرح دهید: ویژگی، بدنه، کاردینالیتی، درجه، عنوان، رابطه، مرابطه و تاپل. همچنین این‌که هر رابطه و هر مرابطه دارای یک نوع (رابطه) هستند. همچنین اصطلاح نوع رابطه را باز هم بسیار مختصر.

2- یک ثابت (لیترال) چیست؟

3- تساوی دو تاپل را به اختصار شرح دهید.

4- در بخش «تاپل چیست؟» دیدیم این کار کاملا اشتباه است که مثلا بگوییم S1 و حتی 'S1' یک شماره توزیع‌کننده است. («یک مقدار از نوع SNO، یک مقدار از نوع SNO است نه یک مقدار از نوع CHAR»). در نتیجه شکل 1-3 در فصل 1 تا حدودی نادرست است با این وجود این شکل و S1 بصورت شماره توزیع‌کننده در ظاهر درست به نظر می‌رسند. مقدارهای اسکالر شکل پایگاه توزیع‌کنندگان و قطعات را بصورت صحیح، همراه با نوع مربوطه و مطابق مقدمه فصل 2 رسم کنید.

5- به زبان توتوریال‌دی انتخابگر تاپل الف) مرابطه قطعات. ب)مرابطه سفارش‌ها. را بنویسید. همچنین معادل این عبارت‌ها را به زبان SQL (در صورت وجود) بنویسید.

6- به زبان توتوریال‌دی انتخابگر یک تاپل معمولی را بنویسید. همچنین معادل این عبارات را به زبان SQL (در صورت وجود) بنویسید.
7- آیا موضوع ثابت تاپلی قابل درک است؟ ثابت رابطه‌ای چطور؟

8- (این تمرین مشابه تمرین 1-8 از فصل 1 است ولی اکنون می‌توانید با تسلط بیشتری مجددا به آن پاسخ دهید.) تفاوت‌های زیادی بین یک رابطه و یک جدول وجود دارد. تفاوت‌هایی که می‌دانید را نام ببرید.

9- ویژگی‌های تاپل بدون محدودیت هستند و می‌توانند از هر نوعی باشند (بدون استثنا). مثالی بزنید که ویژگی تاپل الف)از نوع تاپل باشد. ب)از نوع رابطه باشد (RVA)

10- تساوی تاپل‌ها بطور دقیق در این فصل شرح داده شد. در مورد تساوی رابطه‌ها چه می‌توان گفت؟ (من این مفهوم را در فصل 5 شرح داده‌ام ولی بد نیست در اینجا این مفهوم را حدس بزنید.)

11- چرا عملگرهای مقایسه‌ای «<» و «>» در مورد تاپل‌ها به کار نمی‌روند؟ این عملگرها بر روی سطرهای SQL قابل استفاده‌اند. این امر را چگونه توجیه می‌کنید؟ (اشاره: چه تفاوتی بین قواعد SQL در مورد تساوی سطرها و قواعد رابطه‌ای در مورد تساوی تاپل‌ها وجود دارد؟)

12- مثالی از یک رابطه الف)با یک RVA ب)با دو RVA، بیاورید. همچنین دو رابطه دیگر ذکر کنید که همین اطلاعات را بدون استفاده از RVA در خود داشته باشند. سپس مثالی بزنید که یک رابطه با یک RVA که دقیقا اطلاعات رابطه بدون RVA را در خود داشته باشد. (شاید بهتر باشد که پس از خواندن فصل 5 دوباره به این تمرین مراجعه نمایید.)

13- برخی اوقات پیشنهاد می‌شود که بجای کلمات رابطه (یا مرابطه)، تاپل و ویژگی از فایل معمولی کامپیوتری، رکورد و فیلد استفاده شود. در این مورد بحث کنید.

14- رابطه‌های TABLE_DUM و TABLE_DEE را به زبان خودتان تعریف کنید. آیا SQL می‌تواند از آنها پشتیبانی کند؟

15- یک عبارت معتبر SQL که مربوط به ساخت سطر است را در اینجا مشاهده می‌کنید.
ROW (1,NULL) آیا این سطر نشان دهنده تهی است؟

16- «تکرار ایده مناسبی در پایگاه داده‌ها است چرا که وجود تکرار در جهان واقعی کاملا طبیعی است. مثلا تمام بیست‌وپنج‌تومانی‌ها تکرار یکدیگراند.» چگونه به چنین استدلالی پاسخ می‌دهید؟

17- چرا پایگاه‌های مربوط به شکل‌های 3-1 و 3-2 غیر رابطه‌ای هستند؟

18- آیا تصور می‌کنید تهی بصورت طبیعی در جهان وجود دارد؟
19- تهی در اصل بعنوان راه‌حلی برای مشکل داده‌های گم شده ارائه گردید. این درست است که در دنیای واقعی، بسیاری اطلاعات گم می‌شوند. (مواردی مثل «بزرگی گفته است...» و یا «نشانی فعلی نامعلوم ») پس اگر تهی غیر مجاز باشد با مواردی از این دست چگونه در پایگاه داده برخورد کنیم؟ (مطمئنا متوجه شده‌اید که پاسخ این سوال را در این فصل نداده‌ام. این تمرین بهتر است در گروه به بحث گذاشته شود. من در فصل 7 اندکی بیشتر در این باره صحبت خواهم کرد.

20- TABLE_DEE به معنای درست و TABLE_DUM به معنای غلط است. آیا این به معنای عدم نیاز به نوع داده‌ای بولین است؟ می‌دانیم DEE و DUM مرابطه نیستند. فکر می‌کنید که نیازی به تعریف یک متغیر رابطه‌ای با درجه صفر وجود دارد؟

21- در متن این فصل عبارت SQL زیر را مشاهده کردید:

```
VALUES ( 1, 2 ), ( 2, 1 ), ( 1, 1 ), ( 1, 2 )
```

که به معنای یک جدول با چهار سطر است. عبارت زیر به چه معنا است:

```
VALUES ( ( 1, 2 ), ( 2, 1 ), ( 1, 1 ), ( 1, 2 ) )
```

فصل چهارم

متغیرهای رابطه‌ای

در فصل اول دیدیم که متغیر رابطه‌ای (یا به اختصار مرابطه) متغیری است که مقدار مجاز برای آن رابطه است. دقیقا مرابطه‌ها (و نه رابطه‌ها) هدف عملگرهای INSERT، DELETE و UPDATE هستند. قبلا دیدید که این عملگرها خلاصه شده انتساب‌های رابطه‌ای‌اند. همچنین یادآوری کردم که اولا اگر R یک مرابطه و r رابطه‌ای باشد که به R منتسب شده است آنگاه R و r بایستی از یک نوع باشند. دوما اصطلاحات عنوان، بدنه، ویژگی، تاپل، کاردینالیتی و درجه که در فصل سوم برای رابطه‌ها معرفی شدند، برای مرابطه‌ها هم قابل استفاده هستند. اکنون وقت آن است که این مباحث را دقیق‌تر بررسی کنیم. در مثال‌ها من از توتوریال‌دی برای تعریف متغیرهای رابطه‌ای پایه در پایگاه داده توزیع کنندگان و قطعات استفاده خواهم کرد:

```
VAR S BASE RELATION
    { SNO SNO, SNAME NAME, STATUS INTEGER, CITY CHAR }
     KEY { SNO } ;

  VAR P BASE RELATION
    { PNO PNO, PNAME NAME, COLOR COLOR, WEIGHT WEIGHT,
CITY CHAR }
     KEY { PNO } ;

  VAR SP BASE RELATION
    { SNO SNO, PNO PNO, QTY QTY }
     KEY { SNO, PNO }

     FOREIGN KEY { SNO } REFERENCES S
     FOREIGN KEY { PNO } REFERENCES P ;
```

به‌روزرسانی یعنی کار روی یک مجموعه در یک لحظه

اولین نکته‌ای که می‌خواهم بر آن تاکید کنم این است که انتساب رابطه‌ای (صرف‌نظر از نحوه نوشتن آن) عملگری در سطح مجموعه‌هاست (در حقیقت تمامی عملگرهای مدل رابطه‌ای در سطح مجموعه هستند. این موضوع را در فصل پنجم خواهید دید). بنابراین INSERT یک

مجموعه از تاپل‌ها را درون مرابطه مقصد درج و DELETE مجموعه‌ای از تاپل‌ها را از مرابطه مقصد حذف می‌کند و UPDATE تعدادی از تاپل‌های مرابطه مقصد را تغییر می‌دهد. ما معمولا بدین‌گونه صحبت می‌کنیم که (مثلا) تاپل‌های مشخص شده به‌روزرسانی شدند. ولی بایستی بدانید که:

- ظاهرا چنین به نظر می‌رسد که تاپل‌ها یکی‌یکی به‌روزرسانی گردیده‌اند.
- ولی به‌روزرسانی مجموعه‌ای از تاپل‌ها به صورت یک به یک در مواردی غیر ممکن است.

مثلا فرض کنید که مرابطه S تحت یک قید جامعیتی (فصل ششم) است که می‌گوید S1 و S4 همواره از یک شهر هستند. اگر یک «به‌روزرسانی تک‌تاپلی» بخواهد فقط شهر یکی از این دو را تغییر دهد حتما شکست می‌خورد. بنابراین بایستی هر دو آنها را در یک زمان تغییر داد، احتمالا با عبارتی شبیه این (در SQL):

```
UPDATE S
SET    CITY = 'New York'
WHERE  S.SNO = SNO('S1') OR S.SNO = SNO('S4') ;
```

بدیهی است در اینجا آنچه به‌روزرسانی می‌شود، مجموعه‌ای از تاپل‌ها است.

توجه

در اینجا همان عبارت به‌روزرسانی به توتوریال‌دی نوشته شده است (همانطور که مشاهده می‌کنید خیلی شبیه عبارت قبلی است):

```
UPDATE S WHERE SNO = SNO('S1') OR SNO = SNO('S4')
    ( CITY := 'New York' ) ;
```

از این بحث نتیجه می‌گیریم که به‌روزرسانی‌های مکان‌دار در SQL (UPDATE و DELETE همراه با WHERE CURRENT OF cursor) متعلق به مدل رابطه‌ای نیستند. چرا که این عملگرها در سطح تاپل (و نه در سطح مجموعه) می‌باشند. بیشتر محصولات امروزی این دستورات را به کار می‌گیرند و به همین دلیل هم نمی‌توانند پشتیبانی مناسبی از قیدهای جامعیت به عمل آورند. اگر محصولات روزی از این جهت اصلاح شوند،

«به‌روزرسانی مکان‌دار» دیگر کار نخواهد کرد و برنامه‌های کاربردی که از آن استفاده کرده‌اند، از کار خواهند افتاد. اتفاقی که به نظر من چندان مطلوب نیست.

اکنون بایستی به چیزی اعتراف کنم. واقعیت این است که صحبت‌هایم در مورد «به‌روزرسانی یک تاپل» یا مجموعه‌ای از تاپل‌ها، بسیار اشتباه (نمی‌گویم مغشوش) است. اگر *V* مفعول عمل به‌روزرسانی باشد آنگاه *V* بایستی یک *متغیر* باشد و نه یک *مقدار*. تاپل‌ها همانند رابطه‌ها مقدار هستند و قابل به‌روزرسانی نیستند. واقعا چه منظوری داریم زمانی که (مثلا) می‌گوییم درون مرابطه *R* تاپل *t1* به *t2* به‌روزرسانی شد. آیا یعنی درون *R* تاپل *t1* را با تاپل دیگری بنام *t2* عوض می‌کنیم؟ این صحبت هنوز هم مبهم است. آیا واقعا معنای این کار این است که مقدار *r2* جانشین *r1* که مقدار اصلی رابطه بود، شده است؟ رابطه *r2* در اینجا دقیقا چه نقشی دارد؟ فرض کنید دو رابطه *s1* و *s2* را داریم که هر کدام فقط یک تاپل به ترتیب به نام‌های *r1* و *r2* را در خود دارند. در این صورت *r2* معادل (*r1* MINUS *s1*) UNION *s2* خواهد بود. به زبان دیگر منظور از «به‌روزرسانی *t1* به *t2* در مرابطه *R*» می‌تواند حذف *t1* و سپس درج *t2* باشد. با وجود تمام این‌ها من تساهل در صحبت درباره حذف و درج یک تاپل خاص را مجاز می‌شمارم.

همین بحث در مورد غیر قابل فهم بودن «به‌روزرسانی ویژگی *A* در تاپل *t*» درون رابطه *r* و یا مرابطه *R* نیز وجود دارد. البته به هر حال ما این کار را انجام می‌دهیم و دلیل آن هم راحتی (و طولانی نکردن صحبت) است. با این وجود همانند موضوع رفیق بودن با کاربر در فصل اول، در اینجا هم این کار فقط در صورتی مجاز است که فهمیده باشیم چنین گفتاری فقط به واقعیت شباهت دارد وگرنه ممکن است ما را گمراه نماید.

در مورد کلید کاندید بیشتر بدانیم

من تعریف کلید کاندید را در فصل اول بیان کردم ولی اینجا می‌خواهم با دقت بیشتری به آن بپردازم. یک تعریف:

تعریف: اگر *K* زیرمجموعه‌ای از عنوان مرابطه *R* باشد آنگاه *K* درصورتی کلید کاندید (یا بطور مختصر کلید) R است اگر و تنها اگر دارای هر دو خاصیت زیر باشد:

یکتایی

غیر ممکن است در R دو تاپل مجزا دارای مقدار یکسان برای K باشند.

کاهش ناپذیری

هیچ زیرمجموعه‌ای از K خاصیت یکتایی ندارد.

توجه

مطابق معمول این کتاب، هر کجا گفته‌ام «B زیرمجموعه A است» و یا «A مجموعه مادر B است»، این احتمال وجود دارد که A و B مساوی باشند. اگر چنین احتمالی وجود نداشته باشد از اصطلاحات *مجموعه مادر محض* و *زیرمجموعه محض* استفاده خواهم کرد.

خاصیت یکتایی بی‌نیاز از هر توضیحی است ولی کاهش‌ناپذیری نیاز به کمی توضیح دارد. رابطه S را در نظر بگیرید و فرض کنید که زیرمجموعه‌ای از ویژگی‌های عنوان که شامل {SNO,CITY} است را SK نامیده‌ایم. مسلما این زیرمجموعه دارای خاصیت یکتایی است (هیچ دو تاپل مجزایی در S وجود ندارند که در SK دارای یک مقدار باشند). ولی این مجموعه خاصیت کاهش‌ناپذیری را ندارد چرا که می‌توانیم ویژگی CITY را از آن حذف کنیم و مشاهده نماییم که آنچه باقی مانده است (یعنی {SNO}) هنوز خاصیت یکتایی دارد. پس ما نمی‌توانیم SK را کلید به حساب آوریم چرا که «زیادی بزرگ» است. در مقابل {SNO} کاهش‌ناپذیر و همچنین کلید است.

چرا اصرار داریم که کلید کاهش‌ناپذیر باشد؟ یک دلیل این است که اگر یک «کلید» کاهش‌پذیر داشته باشیم آنگاه DBMS نخواهد توانست اجبار به یکتایی صفت را به درستی انجام دهد. مثلا فرض کنید که ما به دروغ {SNO,CITY} را بعنوان کلید به DBMS معرفی نموده‌ایم. در این صورت DBMS یکتایی «جهانی» شماره توزیع‌کننده را تضمین نمی‌کند و فقط مجبور به حفظ یکتایی «محلی» آن خواهیم بود بدین معنا که شماره توزیع‌کننده فقط در یک شهر خاص یکتاست. به همین یک دلیل (و همچنین دلایل دیگر) می‌گوییم کلید نبایستی دارای ویژگی‌های زاید (در تعیین یکتایی) باشد.

تمامی مرابطه‌هایی که تاکنون دیده‌ایم یک کلید داشته‌اند و برخی از آنها دارای دو کلید یا بیشتر بوده‌اند (خودتان می‌توانید آنها را پیدا کنید). به اتفاق طبیعی تداخل کلیدها در مثال دوم و سوم توجه کنید. TAX_BRACKET به معنای جدول مالیاتی، PERCENTAGE به معنای درصد، ROSTER به معنای جدول ثبت فرودگاه، GATE به معنای خروجی فرودگاه به هواپیما، PILOT به معنای خلبان، MARRIAGE به معنای عقد، SPOUSE_A و SPOUSE_B به معنای طرف اول و دوم ازدواج (زن و شوهر) می‌باشند.

```
VAR TAX_BRACKET BASE RELATION
  { LOW MONEY, HIGH MONEY, PERCENTAGE INTEGER }
    KEY { LOW }
    KEY { HIGH }
    KEY { PERCENTAGE } ;

VAR ROSTER BASE RELATION
  { DAY DAY_OF_WEEK, TIME TIME_OF_DAY, GATE GATE,
PILOT NAME }
    KEY { DAY, TIME, GATE }
    KEY { DAY, TIME, PILOT } ;

VAR MARRIAGE BASE RELATION
  { SPOUSE_A NAME, SPOUSE_B NAME, DATE_OF_MARRIAGE
DATE }
    /* فرض کنید هیچ کس در یک روز چند زن نگرفته است و هر زوج فقط یک بار ازدواج
    کرده‌اند (رجوع نکرده‌اند) */
    KEY { SPOUSE_A, DATE_OF_MARRIAGE }
    KEY { DATE_OF_MARRIAGE, SPOUSE_B }
    KEY { SPOUSE_B, SPOUSE_A } ;
```

این بحث را با چند نکته مختلف به پایان می‌برم. اول اینکه موضوع کلید مربوط به مرابطه‌ها (و نه رابطه‌ها) است. چرا؟ تعیین چیزی به عنوان کلید برای این است که بگوییم قید جامعیت در حال اجرا است و قیدهای جامعیت بر متغیرها (و نه مقدارها) اعمال می‌شوند. (قیدهای جامعیت، به‌روزرسانی را مقید و محدود می‌کنند و به‌روزرسانی عملی است که بر روی متغیرها انجام پذیر است. در فصل ششم این بحث ادامه خواهد یافت.)

دوم اینکه اگر R یک مرابطه باشد آنگاه R حداقل یک کلید کاندید خواهد داشت. به این دلیل که تمام مقدارهای ممکن برای R رابطه‌هایی هستند که طبیعتاً تاپل تکراری در آنها وجود ندارد. پس دست‌کم ترکیب تمامی ویژگی‌های R مطمئنا شرط یکتایی را برآورده خواهند نمود.* حال اگر این مجموعه شرط کاهش‌ناپذیری را نداشته باشد مسلما یک یا چند زیرمجموعه محض از آن دارای این شرط خواهند بود. در هر صورت *چیزی* وجود دارد که هر دو خاصیت یکتایی و کاهش‌ناپذیری را داشته باشد.

سوم اینکه توجه داشته باشید که مقدارهای متناظر با کلید *تاپل* هستند. در مورد مرابطه S با تنها کلید {SNO}، مقدار آن برای یک تاپل خاص مثلا توزیع‌کننده S1، خود یک تاپل است.

TUPLE { SNO SNO('S1') }

در عمل و بصورت غیر رسمی گفته می‌شود که مقدار مربوط به کلید در این مثال تنها S1 (و یا SNO('S1')) است. در حالی که در واقع چنین نیست.

در ادامه نکته قبل؛ بایستی هم اکنون روشن شود که مفهوم کلید مانند بسیاری دیگر از مفاهیم مدل رابطه‌ای به موضوع بنیادی تساوی تاپل‌ها، وابسته است. برای به اجرا در آوردن قید یکتایی بایستی مشخص شود چه وقت دو مقدار مربوط به کلید با هم مساوی‌اند و این مساله به موضوع تساوی تاپل‌ها بازمی‌گردد. حتی زمانی که مانند مرابطه S تاپل‌ها از درجه یک و «شبیه» مقدارهای ساده اسکالر باشند.

آخرین نکته در مورد بحث وابستگی تابعی است. من نمی‌خواهم در اینجا وارد جزئیات آن شوم –این کار را در فصل هفتم خواهم کرد– ولی در هر صورت بایستی با این موضوع آشنا شوید. فقط می‌خواهم در اینجا توجه شما را به موضوع جلب کنم. فرض کنید که K کلید و A یک ویژگی دلخواه از مرابطه R باشند. در این صورت حتما این وابستگی تابعی در آن وجود دارد:

$$K \rightarrow A$$

موشکافی: بطور کلی وابستگی تابعی $A \rightarrow K$ بدین معناست که هرگاه دو تاپل از R دارای مقداری یکسان در K باشند آنگاه دارای مقداری یکسان در A خواهند بود. اما اگر دو تاپل دارای مقداری یکسان در K باشند و K کلید باشد آنگاه طبق تعریف این دو تاپل کاملا یکسان

* البته چنین چیزی را نمی‌توان در مورد جدول‌های SQL گفت. در این جدول‌ها تکرار مجاز است بنابراین ممکن است کلیدی وجود نداشته باشد.

هستند و بایستی در A هم دارای مقدار مساوی باشند. به زبان ساده؛ همیشه «فلش‌های وابستگی تابعی» از کلید خارج می‌شوند و به هر چیز دیگری در مرابطه اشاره می‌نمایند. در فصل هفتم به این موضوع باز خواهیم گشت.

در مورد کلید خارجی بیشتر بدانیم

من تعریف اصلی کلید خارجی را در فصل اول شرح دادم ولی در اینجا تعریفی دقیق را ملاحظه می‌کنید (توجه کنید که در اینجا هم مفهوم تساوی تاپل‌ها وجود دارد):

فرض کنید که $R1$ و $R2$ دو مرابطه (نه لزوما) مجزا و K کلید کاندید $R1$ باشند. همچنین فرض کنید FK زیرمجموعه‌ای از عنوان $R2$ است که (احتمالا بعد از تغییر نام برخی ویژگی‌ها) دقیقا مساوی همان ویژگی‌های K است. در این صورت FK کلید خارجی خواهد بود اگر و تنها اگر در هر لحظه از زمان هر تاپل موجود در $R2$ در ویژگی FK دارای مقداری باشد که در همان لحظه از زمان معادل آن در مقدار K یک تاپل (لزوما یکتا) موجود در $R1$، وجود داشته باشد.

همانطور که می‌دانیم در پایگاه توزیع‌کنندگان و قطعات، {SNO} و {PNO} کلیدهای خارجی رابطه SP هستند که به تنها کلید کاندید (و در حقیقت کلید اصلی) مرابطه‌های S و P اشاره دارند. در اینجا مثال دیگری را مشاهده می‌کنید:

```
VAR EMP BASE RELATION
  { ENO ENO, ..., MNO ENO, ... }
   KEY { ENO }
   FOREIGN KEY { RENAME ( MNO AS ENO ) } REFERENCES
EMP ;
```

این پایگاه مربوط به کارمندان یک اداره است و ENO نشان‌دهنده شماره پرسنلی شخص و MNO نشان‌دهنده شماره پرسنلی مدیر بالادست وی می‌باشد. مرابطه رجوع کننده (R2 طبق تعریف قبل) و مرابطه مورد مراجعه (R1 طبق تعریف قبل) در این مثال یکی هستند. مثلا در تاپل مربوط به کارمند E3 ممکن است در MNO دارای مقدار E2 باشد که موجب اتصال وی به تاپل E2 می‌شود. مقدارهای کلید خارجی همانند مقدارهای کلید اصلی تاپل هستند. همچنین مجبوریم MNO در نقش کلید خارجی را تغییر نام دهیم تا تطبیق مساوی بودن تاپل‌ها قابل انجام باشد. (چرا تطبیق مساوی بودن تاپل‌ها؟ برای اینکه بدون شک این عمل یکی از مراحل کنترل قید کلید

خارجی است. به یاد آورید که پیش نیاز تطبیق تاپل‌ها «همنوع بودن» آنها است و همنوع بودن به این معناست که باید نوع و نام تک‌تک ویژگی‌ها با هم یکی باشد.)

به عنوان یک حاشیه، در اینجا بایستی تذکر دهم که قواعد مدل اصلی در سیستم رابطه‌ای به تطبیق کلید خارجی نه با کلید کاندید، بلکه بطور خاص‌تر با *کلید اصلی* تاکید دارد. در فصل اول دلایلی آوردم در مورد اینکه نباید اصرار داشته باشیم که حتما یک کلید کاندید بعنوان کلید اصلی انتخاب شود، از همین رو اصراری بر مطابقت کلید خارجی با کلید اصلی ندارم. (در این مورد با SQL موافقم.)

SQL نه تنها از کلید خارجی حمایت می‌کند بلکه از تاثیرات ارجاعی مانند CASCADE (که می‌تواند از انواع ON UPDATE و ON DELETE باشد.) نیز پشتیبانی می‌نماید. مثلا دستور ساخت جدول برای سفارش‌ها می‌تواند شامل عبارت زیر باشد:

FOREIGN KEY (SNO) REFERENCES S (SNO) ON DELETE CASCADE

با تعیین این ویژگی، تلاش برای حذف یک توزیع‌کننده خاص منجر به حذف تمامی سفارش‌های مربوط به وی خواهد شد. من این نکته را شرح دادم به این دلیل که:

- اولا این خصوصیت می‌تواند در عمل مفید باشد اما با این حال بخشی از مدل رابطه‌ای نیست.
- ولی این امر لزوما مشکل تلقی نمی‌شود! مدل رابطه‌ای زیربنای پایگاه داده است، ولی *فقط* یک زیربناست. دلیلی ندارد که امکانات اضافی بر رو یا در کنار این زیربنا ایجاد نشوند. فقط این امکانات نبایستی اصول مدل را زیر پا بگذارند. (و اگر در فضای مدل بگنجند و مفید باشند من توصیه می‌کنم که اضافه شوند). جهت موشکافی: الف. نظریه نوع‌ها مثال خوبی است. در فصل دوم دیدیم «نوع‌ها از جدول‌ها مستقل هستند.» ولی این را هم دیدیم که پشتیبانی کامل از نوع‌ها در مدل رابطه‌ای امری بسیار مطلوب و پسندیده است. ب. دومین مثال اینکه، مدل رابطه‌ای چیزی برای نظارت بر ترمیم و همزمانی ندارد. ولی این بدین معنا نیست که سیستم‌های رابطه‌ای نبایستی چنین امکاناتی داشته باشند. (در حقیقت مدل رابطه‌ای در این موارد مطالبی را بصورت تلویحی بیان می‌کند، چرا که این موضوعات مربوط به چگونگی پیاده‌سازی عمل به‌روزرسانی توسط DBMS هستند، بدون از بین رفتن اطلاعات.)

و آخرین تذکر جهت پایان این بخش: من کلیدهای خارجی را به این دلیل شرح دادم که از نظر کاربردی اهمیت بسیار زیادی دارند و همچنین بخشی از مدل اصلی هستند ولی تصور می‌کنم که بایستی بر این نکته تاکید کنم که آنها زیاد بنیادی نیستند. آنها فقط نشانه وجود قید جامعیت مشخصی هستند که در عمل زیاد به کار می‌آیند همانطور که در فصل ششم خواهیم دید.* (به همین اندازه هم صحبت در مورد کلید کاندید هم وجود داشت، ولیکن به دلیل اهمیت کم آن از دیدگاه کاربردی، مطرح کردن آن خارج از حوصله بود.)

در مورد ویوها بیشتر بدانیم

یک ویو که مرابطه مجازی نیز نامیده می‌شود، مرابطه‌ای است که وجود خارجی ندارد ولی از دید کاربر شبیه یک مرابطه واقعی به نظر می‌رسد.

تعریف: ویو یا *مرابطه مجازی* V، مرابطه‌ای است که مقدار آن در زمان t از محاسبه یک عبارت رابطه‌ای مشخص در زمان t بدست می‌آید. این عبارت در زمان تعریف V، تعیین می‌شود و به یک مرابطه یا بیشتر مراجعه دارد. در اینجا دو مثال وجود مشاهده می‌کنید، «توزیع‌کنندگان لندنی» و «توزیع‌کنندگان غیر لندنی» (سمت چپ با توتوریال‌دی و سمت راست با SQL):

```
VAR LS VIRTUAL                    | CREATE VIEW LS AS
( S WHERE CITY = 'London' ) ;     |  ( SELECT S.*
WHERE S.SNO IN                    |    FROM   S
                                  |    WHERE  S.CITY = 'London' ) ;

VAR NLS VIRTUAL                   |  CREATE VIEW NLS AS
( S WHERE CITY ≠ 'London' ) ;     |  ( SELECT S.*
                                  |    FROM   S
                                  |    WHERE  S.CITY <> 'London' )
```

استفاده از پرانتز در این مثال‌ها غیر ضروری و در عین حال مجاز است. من از آنها را برای خوانایی بیشتر استفاده کرده‌ام.

* دقیقا به همین دلیل توتوریال‌دی از آنها پشتیبانی نمی‌کند. با این حال مطمئنم که نگارش تجاری این زبان این حمایت را خواهد داشت و من حق دارم در این کتاب فرض کنم که این پشتیبانی در حال حاضر هم وجود دارد.

ویوهای استخراج‌کننده

تکرار می‌کنم، ویوها بایستی از نظر کاربر موجودیت‌های مستقلی باشند. به بیان دیگر کاربر بایستی آنها را به صورت مرابطه‌های پایه «ببیند و احساس کند». بطور خاص کاربر بایستی بتواند همان کارهایی که با مرابطه‌های پایه می‌کند را با ویوها نیز انجام دهد و DBMS هم بایستی این توانایی را داشته باشد که این عملیات کاربر را به نحوی مناسب و با توجه به ویویی که «در نهایت» تعریف شده است، به مرابطه‌های پایه منتسب کند. گفتم «در نهایت» زیرا اگر ویوها واقعا شبیه مرابطه‌های پایه باشند، تعریف ویوهای بعدی بر روی آنها امکان‌پذیر است. مانند این مثال SQL:

```
CREATE VIEW LS_STATUS
  AS ( SELECT LS.SNO, LS.STATUS
       FROM   LS ) ;
```

انتساب عملیات فقط خواندنی آسان است. فرض کنید این کوئری SQL را برای ویو LS نوشته‌ایم.

```
SELECT LS.SNO
FROM   LS
WHERE  LS.STATUS > 10
```

ابتدا DBMS منبع را در قسمت FROM با عبارتی که در تعریف ویو آمده است، جایگزین می‌کند. به این صورت:

```
SELECT LS.SNO
  FROM ( SELECT S.*
       FROM   S
       WHERE  S.CITY = 'London' ) AS LS
  WHERE  LS.STATUS > 10
```

اکنون عبارت می‌تواند ساده شود:

```
SELECT S.SNO
FROM   S
WHERE  S.CITY = 'London'
AND    S.STATUS > 10
```

این عملیات همیشه با موفقیت انجام می‌شود و دلیل آن خاصیت بسته بودن جبر رابطه‌ای است. بسته بودن مزایای دیگری نیز دارد از جمله اینکه می‌توان برخی چیزها را *نام‌گذاری* کرد. مثلا در یک کوئری همواره می‌توانیم عبارتی عمومی‌تر داشته باشیم تا چیزی را از یک نوع مشخص محاسبه نماید.* برای نمونه در قسمت FROM می‌توان نام جدول‌های SQL را نوشت. همچنین می‌توانیم عبارتی عمومی‌تر در مورد جدول‌ها بیاوریم چرا که مجاز هستیم عبارت تعریف ویو LS را بجای نام این ویو بنویسیم.

و یک تذکر مهم، چنین عبارات‌هایی در نگارش‌های قدیمی SQL کار نمی‌کنند.(خصوصا SQL استاندارد قبل از سال 1992) و دلیل آن این است که این نگارش‌ها از خاصیت بسته بودن پشتیبانی نمی‌کنند. در نتیجه نگاه ساده به جدول‌ها امکان‌پذیر است ولی نگاه ساده به کوئری‌ها (دیدها) با شکست مواجه می‌شود. همچنین مشکلات دیگری به وجود می‌آید که شرح دادن آنها مشکل است. به این مثال ساده توجه کنید:

```
CREATE VIEW V
    AS ( SELECT S.CITY, SUM ( S.STATUS ) AS ST
         FROM   S
         GROUP  BY S.CITY ) ;

SELECT V.CITY
  FROM   V
  WHERE  V.ST > 25
```

اجرای این مثال قبل از 1992، در SQL شکست می‌خورد. گرچه نگارش استاندارد اصلاح شده است ولی هنوز تمامی محصولات از آن پیروی نمی‌کنند و حداقل یکی از محصولات بزرگ در زمان نگارش کتاب این امکان را ندارد (اوایل 2005).

به‌روزرسانی ویوها

حال سراغ عملگرهای به‌روزرسانی می‌رویم. قبل از پرداختن به جزئیات دوباره به توزیع‌کنندگان لندنی و غیر لندنی می‌پردازم (و این بار از توتوریال‌دی استفاده می‌کنم).

* من ترجیح می‌دادم به جای کلمه مبهم «چیز» از لغت رسمی‌تر شیء برای این منظور استفاده کنم ولی شیء در علم کامپیوتر دارای بار معنایی دیگری است.

```
VAR LS  VIRTUAL ( S WHERE CITY = 'London' ) ;

VAR NLS VIRTUAL ( S WHERE CITY ≠ 'London' ) ;
```

نکته مهم اینجاست که: بجای اینکه S رابطه پایه و LS و NLS ویو باشند، LS و NLS می‌توانند رابطه پایه و S ویو باشد. مانند این:

```
VAR LS BASE RELATION
  { SNO SNO, SNAME NAME, STATUS INTEGER, CITY CHAR }
    KEY { SNO } ;

VAR NLS BASE RELATION
  { SNO SNO, SNAME NAME, STATUS INTEGER, CITY CHAR }
    KEY { SNO } ;

VAR S VIRTUAL ( LS UNION NLS ) ;
```

توجه

برای رسیدن تعادل بایستی قیدهای مشخصی تعیین کنیم. قیدهایی که در نتیجه برقراری آنها هر CITY در LS شهر لندن و هر CITY در NLS شهری غیر از لندن خواهد بود. برخی جزئیات را در اینجا از قلم انداخته‌ام. برای بحث بیشتر در این مورد به فصل ششم مراجعه نمایید.

پیام بزرگی در این مثال وجود دارد، *اینکه کدامیک از مرابطه‌ها پایه و کدامیک مجازی باشند، اختیاری است*. بنابراین نبایستی هیچ برتری دلخواه و غیر ضروری میان مرابطه‌های پایه و مجازی وجود داشته باشد. این امر *اصل تعویض‌پذیری* (مرابطه‌های پایه و مجازی)، نامیده می‌شود.

چند مفهوم:

- همانند مرابطه‌های پایه، ویوها هم زیر پوشش قیدهای جامعیت هستند. (معمولا تصور می‌کنیم قیدهای جامعیت فقط بر مرابطه‌های پایه اعمال می‌شوند ولی اصل تعویض پذیری نشان داد که نمی‌توان موقعیت پایه و مجازی را حفظ کرد.)
- دیدها دارای کلید کاندید هستند (و احتمالا بایستی پیش از این برای آنها کلید کاندید تعیین می‌کردم. توتوریال‌دی اجازه این کار را می‌دهد ولی SQL نه).

آنها همچنین می‌توانند کلید خارجی داشته باشند و یا کلید خارجی به آنها مراجعه نماید.

- من به این نکته در فصل اول اشاره نکردم، ولی قاعده «جامعیت وجودی» فقط بر مرابطه‌های پایه (و نه ویوها) قابل اعمال است. در نتیجه اصل تعویض پذیری را نقض می‌کند (البته در هر صورت من این قاعده را رد کرده‌ام چرا که با تهی سروکار دارد).
- بسیاری از محصولات و استانداردهای SQL امکان نوعی «شناسه سطر» را دارند. اگر این امکان فقط قابل اعمال بر جدول‌های پایه (و نه ویوها) باشد، که در عمل هم به همین صورت است، *اصل تعویض‌پذیری* را زیر پا می‌گذارد.* البته شناسه سطر جزئی از مدل رابطه‌ای نیست ولی این بدین معنا نیست که نبایستی از آن پشتیبانی شود. بعنوان یک حاشیه مهم عنوان می‌کنم که اگر این شناسه سطر بعنوان نوعی شناسه شی در نگاه شی‌گرا تلقی شود (که با تاسف بسیار در استاندارد و اکثر محصولات اصلی SQL چنین است) آنگاه استفاده از آنها ممنوع خواهد بود! شناسه شی در عمل یک اشاره‌گر است و مدل رابطه‌ای بطور صریح استفاده از اشاره‌گرها را منع کرده است.

حال به موضوع اصلی بحث باز می‌گردیم. ما *باید* بتوانیم ویوها را به‌روزرسانی کنیم. چرا که اگر نتوانیم، آنگاه بطور روشن *اصل تعویض‌پذیری* را زیر پا گذاشته‌ایم.

همانطور که احتمالا می‌دانید SQL –هم در استاندارد و هم در محصولات عمده تجاری– از این امکان به گونه‌ای بسیار ضعیف پشتیبانی می‌کند. دست‌کم SQL بطور معمول از به‌روزرسانی ویوهایی که عملیات گزینش یا پرتو را روی یک مرابطه پایه انجام می‌دهند، پشتیبانی می‌کند (با وجود مشکلاتی که دارد). برای مثال مطابق ویو زیر (شبیه همان چیزی است که در فصل اول دیدیم):

```
CREATE VIEW SST_PARIS
 AS ( SELECT S.SNO, S.STATUS
    FROM   S
    WHERE  S.CITY = 'Paris' ) ;
```

* همچنین ممکن است *اصل اطلاعات* را هم زیر پا بگذارد (فصل 8 را ببینید).

این ویو پرتوی از گزینش جدول پایه S است و ما می‌توانیم برای مثال DELETE زیر را بر روی آن اعمال کنیم:

```
DELETE
FROM   SST_PARIS
WHERE  SST_PARIS.STATUS > 15 ;
```

این DELETE همانند عبارت زیر است:

```
DELETE
FROM   S
WHERE  S.CITY = 'Paris'
AND    S.STATUS > 15 ;
```

ولی تعداد کمی از محصولات از به‌روزرسانی ویوهای پیچیده‌تر از این پشتیبانی می‌نمایند.

متاسفانه من اکنون در این وادی – که هنوز در آن مناقشات زیادی وجود دارد- سرگردان شده‌ام. نظر شخصی من این است که مساله به‌روزرسانی ویوها امروز تا حد زیادی حل شده است (حداقل به صورت تئوری) ولی دیگران لزوما در این مورد با من موافق نیستند. در هر صورت بحث بیشتر در این مورد نیازمند پیش زمینه‌ای است که در این کتاب وجود ندارد. تنها می‌توانم که شما را به کتاب دیگری تحت عنوان *پایگاه داده‌ها، نوع‌ها و مدل رابطه‌ای بیانیه سوم* نوشته سی‌جی‌دیت و هیو داروین، ارجاع دهم. اگر خواهان جزئیات بیشتری در این مورد باشید، این موضوع با عمق بیشتری در آن جا بررسی شده است.

نکات متفرقه

قبل از پایان این بخش بایستی چند نکته دیگر را بگویم. اول، چیزی است که همه می‌دانند ولی باز هم ارزش گفتن را دارد. ویوها به دو منظور مورد استفاده قرار می‌گیرند:

- کاربری که ویو V را تعریف می‌کند مسلما آگاه است که V جهت عبارت X تعریف شده است. این کاربر می‌تواند در هر کجا به‌جای عبارت X از V استفاده نماید. منظور این کاربر فقط خلاصه‌نویسی است.
- در مقابل کاربری هم هست که فقط در این حد آگاهی دارد که V وجود دارد و برای استفاده در اختیار وی قرار گرفته است ولی (حداقل در حالت ایده‌آل)

نمی‌داند که عبارت X چیست. برای چنین کاربری V می‌تواند یک مرابطه پایه فرض شود، همانطور که واقعا چنین به نظر می‌رسد. این استفاده دوم از ویوها اهمیت بسیار دارد و در این بخش تا حدودی بر آن تمرکز داشته‌ام.

دوم، وقتی ویوها را در ابتدای این فصل توضیح دادم گفتم عبارت رابطه‌ای که ویو را تعریف می‌کند حداقل شامل یک مرابطه است. چرا؟ برای اینکه اگر «مرابطه مجازی» در کار نباشد اصلا مرابطه‌ای نخواهیم داشت. چنین چیزی اساسا متغیر نیست و طبیعتاً قابل به‌روزرسانی هم نخواهد بود. در عوض یک ثابت رابطه‌ای داریم که میتوان آن را «ثرابطه» نامید. برای مثال (با یک ساختار من درآوردی):

```
CONST PERIODIC_TABLE ( RELATION       {
    TUPLE { ELEMENT 'Hydrogen', SYMBOL 'H' , ATOMICNO  1 },
    TUPLE { ELEMENT 'Helium'  , SYMBOL 'He', ATOMICNO  2 },
    ...
    TUPLE { ELEMENT 'Uranium' , SYMBOL 'U' , ATOMICNO 92 }
                                                  } ) ;
```

من تصور نمی‌کنم در شرایطی که عبارت فوق نوعی «ثرابطه» را تعریف می‌کند، بایستی آن را مرابطه در نظر گرفت. به نظر من جا زدن ثابت‌ها بجای متغیرها، هیچ کمکی به یادگیری نمی‌کند.

سوم، یک اصطلاح فنی نامناسب در مجامع دانشگاهی وجود دارد که به بازار تجاری هم سرایت کرده است. از فصل اول یادآوری می‌کنم که یک ویو می‌تواند از یک مرابطه مشتق شود. اما به گونه‌ای دیگر می‌توان «مرابطه مشتق شده» را به چیزی اطلاق کرد که به آن *تصویرلحظه‌ای* (*اسنپ‌شات*) گفته می‌شود. با این نام احتمالا شما را به یاد یک عکس، چیزی بدست آمده از چیز دیگر، واقعی و نه مجازی می‌اندازد. این مفهوم فقط به معنای تعریف یک عبارت روی مرابطه نیست، بلکه به معنای گرفتن یک رونوشت مجزا از داده‌ها هم هست. برای مثال (با یک ساختار دیگر به همان صورت):

```
VAR LSS SNAPSHOT ( S WHERE CITY = 'London' )
   REFRESH EVERY DAY ;
```

تعریف یک تصویر لحظه‌ای تقریبا مشابه اجرای یک کوئری است با این تفاوت که:

- یک تصویر لحظه‌ای با یک نام مشخص (مثل LLS) و بصورت یک مرابطه فقط-خواندنی در پایگاه ذخیره می‌شود (فقط-خواندنی بودن ربطی به تازه‌سازی دوره‌ای ندارد. پاراگراف بعد را ببینید).
- تصویرهای لحظه‌ای بصورت دوره‌ای (در مثال هر روز) تازه‌سازی می‌شوند. این عمل مقدار فعلی را دور می‌اندازد. کوئری را مجددا اجرا می‌کند و نتیجه را بعنوان مقدار جدید تصویر لحظه‌ای ذخیره می‌نماید.

در این مثال تصویر لحظه‌ای LLS داده‌های 24 ساعت گذشته را نمایندگی می‌کند.

تصویرهای لحظه‌ای در انبارکردن داده‌ها، سیستم‌های توزیع شده، و بسیاری موارد دیگر اهمیت دارند. برنامه‌های حسابداری نمونه خوبی در این مورد هستند. در این برنامه‌ها داده‌ها بایستی در لحظه مناسب فریز شوند (مثلا در آخر سال مالی)، و تصویر لحظه‌ای امکان این فریز کردن را بدون قفل کردن سایر برنامه‌ها امکان‌پذیر می‌کند.

مشکل آن است که تصویرهای لحظه‌ای (حداقل در برخی حوزه‌ها) نه بصورت تصویر لحظه‌ای، بلکه بصورت ویوهای «*جامه عمل پوشیده*» شناخته شده‌اند. ولی تصویرهای لحظه‌ای، ویو نیستند! مدل رابطه‌ای تا حدودی این امر را تایید می‌کند که ویوها، جامه عمل نپوشیده‌اند. ویوها با انتساب عملگرهای مناسب به مرابطه‌های پایه ایجاد می‌شوند. بنابراین «دید جامه عمل پوشیده» عبارتی است که در معنا دچار تناقض است. از آن بدتر اینکه کلمه ویو را امروزه بجای «دید جامه عمل پوشیده» به کار برده‌اند (بازهم، حداقل در برخی حوزه‌ها) و این خطر وجود دارد که معنای اصلی این کلمه لوث شود. در این کتاب من همیشه کلمه ویو را در معنای اصلی آن به کار می‌برم ولی هشدار می‌دهم که این کلمه همه جا چنین معنایی ندارد.

مرابطه‌ها و گزاره‌نماها

اکنون به مهمترین قسمت این فصل رسیده‌ایم. بطور خلاصه؛ می‌توانیم جور دیگری به مرابطه‌ها نگاه کنیم. تصور بیشتر افراد از مرابطه‌ها شبیه فایل‌های کامپیوتری معمولی است. فایل‌هایی با تجرید (شاید بهتر است بگوییم با انضباط) بالاتر، ولی در هر صورت فایل. اما می‌توان آنها را از دیدگاهی دیگر نگاه کرد و به عقیده من این روش ما را به درک عمیق‌تری خواهد رسانید. این راه چنین است.

توزیع‌کنندگان مرابطه S را در نظر بگیرید. مانند بقیه مرابطه‌ها، فرض بر آن است که این مرابطه هم بخشی از جهان واقعی را نمایندگی می‌کند. بطور دقیق‌تر عنوان مرابطه نماینده یک گزاره‌نمای مشخص است. بدین معنا که یک عبارت عمومی در مورد دنیای واقعی است (عمومی بدین جهت که پارامتری است؛ توضیح خواهم داد). گزاره‌نمای مرابطه مذکور چنین است:

توزیع‌کننده SNO طرف قرارداد است و نامش SNAME است و رتبه آن STATUS می‌باشد و محل استقرار آن شهر CITY است.

این گزاره‌نما برای مرابطه S *تفسیر موردنظر* است. به این گزاره‌نما *معنا* و یا *بسط داخلی* مرابطه S می‌گویند. می‌توانید فرض را بر این بگذارید که این گزاره‌نما، تابعی است که مقدار برگشتی آن همیشه درست یا غلط است که این مقدار برگشتی بستگی به پارامترهای آن دارد. در مورد این مثال پارامترهایSNO, SNAME, STATUS وCITY (متناظر با ویژگی‌های مرابطه) هستند. این پارامترها پذیرای مقادیری از نوع مناسب‌اند (به ترتیب از نوع INTERGER, NAME, SNO و CHAR). وقتی می‌خواهیم تابع را فراخوانی کنیم بایستی پارامترها را مقدار دهی نماییم. مثلا به ترتیب به آنها این مقدارها را می‌دهیم: S1، اسمیت، 20 و لندن. پس از آن به گزاره زیر خواهیم رسید:

توزیع‌کننده S1 طرف قرارداد است و نامش اسمیت است و رتبه آن 20 و محل استقرار وی شهر لندن است.

بطور کلی یک گزاره در منطق عبارتی است که یا صحیح است و یا غلط دو مثال:

ادوارد اَبی نویسنده کتاب گروه تبهکار *آچار شلاقی* است.

ویلیام شکسپیر نویسنده کتاب گروه تبهکار *آچار شلاقی* است.

اولی صحیح است و دومی غلط. دچار اشتباه نشوید و تصور نکنید گزاره‌ها همیشه صحیح هستند! اما من به هنگام بحث در مورد آنها و مرابطه‌ها فرض را بر این می‌گذارم که همه آنها صحیح‌اند.

- هر مرابطه یک گزاره‌نمای متناظر دارد که *گزاره‌نمای مرابطه* نامیده می‌شود.
- فرض کنید که مرابطه R دارای گزاره‌نمای P است. آنگاه هر تاپل مانند t که در R وجود دارد می‌تواند به منزله یک گزاره خاص p به شمار رود. p از جای‌گذاری پارامترهایی که در t وجود دارد بدست می‌آید.

- و (بسیار مهم!) قبول داریم که هر گزاره مانند p که به این طریق ایجاد شود، گزاره‌ای صحیح (و نه غلط) است.

از این‌رو در مثال خودمان برای مرابطه S می‌پذیریم که تمام گزاره‌ها صحیح‌اند:

توزیع‌کننده S1 طرف قرارداد است و نامش اسمیت است و رتبه آن 20 و محل استقرار آن شهر لندن است.

توزیع‌کننده S2 طرف قرارداد است و نامش جونز است و رتبه آن 10 و محل استقرار آن شهر پاریس است.

توزیع‌کننده S3 طرف قرارداد است و نامش بلَک است و رتبه آن 30 و محل استقرار آن شهر پاریس است.

و به همین ترتیب. بیشتر در این مورد صحبت کنیم. اگر یک تاپل فرضی ظاهرا قابل قبول باشد و بتواند در مرابطه حاضر شود ولی در واقع این اتفاق نیافتد (در مرابطه دیده نشود)، فرض را بر این خواهیم گذاشت که این گزاره در آن زمان نادرست است (به عبارت دیگر فرض *بسته بودن جهان* را می‌پذیریم). برای مثال تاپل:

```
TUPLE { SNO SNO('S6'), SNAME NAME('Lopez'),
        STATUS 30, CITY 'Madrid' }
```

قبول دارید که این یک تاپل ظاهرا معتبر است ولی در این زمان در مرابطه S حضور ندارد. در نتیجه می‌پذیریم که موضوع زیر در این لحظه از زمان واقعیت ندارد:

توزیع‌کننده S6 طرف قرارداد است و نامش لوپز است و رتبه آن 30 و محل استقرار آن شهر مادرید است.

به زبان دیگر، یک مرابطه در هر لحظه از زمان *«فقط شامل»* و *«شامل تمام»* تاپل‌هایی است که گزاره‌های صحیح در آن زمان را نمایندگی می‌کنند.

یک اصطلاح دیگر: مجددا فرض کنید P گزاره‌نمای مرابطه و مربوط به مرابطه R است و همچنین R در یک زمان خاص دارای مقدار r می‌باشد. آنگاه r (یا به زبان دقیقتر بدنه r) *بسط خارجی* P در یک زمان مشخص است. توجه داشته باشید که بسط خارجی در طول زمان تغییر می‌کند ولی بسط داخلی چنین نیست.

عبارت‌های رابطه‌ای

ایده‌ای مطرح شده در این قسمت را می‌توان در مورد عبارت‌های رابطه‌ای دلخواه تعمیم داد. مثلا در اینجا یک پرتو از توزیع‌کنندگان شامل تمام ویژگی‌ها بجز CITY، را مشاهده می‌کنید:

S { SNO, SNAME, STATUS }

نتیجه شامل تمامی تاپل‌های با این شکل خواهد بود:

TUPLE { SNO *s*, SNAME *n*, STATUS *t* }

که از تاپل‌هایی به این شکل بدست آمده‌اند:

TUPLE { SNO *s*, SNAME *n*, STATUS *t*, CITY *c* }

توجه کنید که در S هم یک شهر با مقدار *c* وجود داشته است. به زبان دیگر نتیجه گسترش عبارت گزاره‌نمایی به شکل زیر است:

وجود دارد شهری بنام CITY که توزیع‌کننده طرف قرارداد SNO به نام SNAME و رتبه STATUS در شهر CITY واقع است.

توجه کنید که گزاره‌نما در اصل فقط سه پارامتر دارد و رابطه متناظر (پرتو توزیع‌کنندگان بجز نام شهر) هم فقط سه ویژگی دارد. CITY در اینجا پارامتر نیست و منطق‌دانان به آن متغیر *مقید* می‌گویند به این دلیل که با عبارت *وجود دارد شهری*، «محدود» شده است (برای توضیح بیشتر در مورد متغیرهای مقید به ضمیمه کتاب مراجعه کنید). بهتر است فرض کنیم که این گزاره‌نما فقط سه (و نه چهار) پارامتر دارد. چنین گزاره‌نمایی به این صورت خواهد بود:

توزیع‌کننده SNO طرف قرارداد است و نامش SNAME بوده، رتبه‌اش STATUS و در شهری (که ما نمی‌دانیم کجاست) مستقر است.

تمامی ویوها نشان‌دهنده چنین گزاره‌نماهایی می‌باشند. مثلا اگر ویو SST به این صورت تعریف شده باشد:

VAR SST VIRTUAL (S { SNO, SNAME, STATUS }) ;

گزاره‌نمای این مرابطه دقیقا چنین است:

توزیع‌کننده SNO طرف قرارداد است و نامش SNAME بوده، رتبه‌اش STATUS و در شهری مستقر است.

و آخرین نکته که می‌خواهم در مورد گزاره‌ها و گزاره‌نماها بگویم: گفته شد که گزاره‌نما مجموعه‌ای از پارامترها را داراست. طبیعی است که این مجموعه می‌تواند خالی باشد و

اگر چنین باشد گزاره‌نما تبدیل به گزاره می‌شود! (چیزی که بدون قید و شرط صحیح یا غلط است). به زبان دیگر گزاره، یک گزاره‌نمای *تباه‌شده* است. تمام گزاره‌ها، گزاره‌نما هستند ولی بیشتر گزاره‌نماها، گزاره نیستند.

درباره تفاوت میان رابطه‌ها و نوع‌ها بیشتر بدانیم

فصل دوم تفاوت بین رابطه‌ها و نوع‌ها نام داشت ولی در آن زمان در موقعیتی نبودیم که بتوانیم مهمترین تفاوت این دو را مشاهده کنیم ولی در اینجا موقعیت آنرا داریم و این کار را انجام می‌دهم:

نشان دادم که پایگاه داده‌ها در هر لحظه از زمان بصورت مجموعه‌ای از گزاره‌های صحیح به نظر می‌رسد: برای مثال گزاره *توزیع‌کننده S1 طرف قرارداد است و نامش اسمیت است و رتبه آن 20 و محل استقرار آن شهر لندن است*. بطور دقیق‌تر نشان دادم که آرگومان‌هایی در این گزاره وجود دارند. (در اینجا S1 و اسمیت و 20 و لندن) که همان ویژگی‌های تاپل مربوطه هستند و هر کدام از این مقدارها از یک نوع متناسب می‌باشند. بنابراین:

نوع‌ها مجموعه‌ای از چیزهایی هستند که می‌توانیم در مورد آنها صحبت کنیم.

رابطه‌ها جملاتی (صحیح) در مورد این چیزها هستند.

به بیانی دیگر، نوع‌ها فرهنگ لغات (چیزهایی که می‌توانیم درمورد آنها صحبت کنیم) را به ما می‌دهند و رابطه‌ها ما را قادر می‌سازند در مورد چیزهایی که می‌توانیم در مورد آنها صحبت کنیم، چیزی بگوییم (این مقایسه جالب می‌تواند کمک کند؛ *نسبت نوع به رابطه مانند نسبت اسامی به جمله است*). اگر صحبت‌مان را محدود به توزیع‌کنندگان کنیم، خواهیم دید که:

- تنها می‌توانیم در مورد شماره توزیع‌کنندگان، نام‌ها، اعداد صحیح و رشته‌های کاراکتری صحبت کنیم.
- چیزی که می‌توانیم بگوییم چنین قالبی دارد: «یک توزیع‌کننده با یک شماره مشخص طرف قرارداد است و یک نام مشخص دارد و رتبه‌ای دارد که با یک عدد صحیح مشخص شده است و در شهری مستقر است که نام آن با یک رشته کاراکتری مشخص شده است.» همین و بس. (همین و بس به جز چیزهایی که بطور منطقی از آنچه می‌توانیم بگوییم استنتاج می‌شود. مثلا با توجه به چیزهایی

که می‌دانیم، می‌توانیم در مورد توزیع‌کننده S1 چنین بگوییم که *توزیع‌کننده* SNO *طرف قرارداد است و نامش* SNAME *بوده، رتبه‌اش* STATUS *و در شهری مستقر است*. ما شهر را نامعلوم گذاشتیم. اگر این جمله به نظرتان آشناست و فکر می‌کنید ارتباط تنگاتنگی با پرتو رابطه‌ای دارد.... درست حدس زده‌اید.)

این موضوع سه پی‌آمد مهم دارد. رابطه همانطور که دیدید «تکه‌ای از دنیای واقعی است» (در بخش قبل گفتم):

1. نوع‌ها و رابطه‌ها هر دو ضروری‌اند. بدون نوع چیزی برای آنکه در موردش صحبت کنیم، نداریم و بدون رابطه اصلا چیزی نمی‌توان گفت.
2. نوع‌ها و رابطه‌ها در کنار یکدیگر برای ما کافی‌اند و برای صحبت منطقی به هیچ چیز دیگری نیازمند نیستیم (البته برای نشان دادن تغییرات دنیای واقعی به مرابطه‌ها هم نیاز داریم ولی برای نشان دادن موقعیت‌های لحظه‌ای به آنها نیازی نداریم).
3. نوع‌ها و رابطه‌ها با هم یکی نیستند. از کسانی که سعی دارند چنین چیزی را به شما بقبولانند دوری کنید! در واقع این ادعا که نوع یک گونه خاص از رابطه است، چیزی است که محصولات تجاری سعی در جا انداختن آن دارند و امیدوارم روزی روشن شود محصولاتی که بر پایه خطاهای منطقی ساخته شده‌اند محکوم به فنا هستند. چنین محصولاتی از نظر من رابطه‌ای نیستند. بیشتر این محصولات آنهایی هستند که از شی‌گرایی پشتیبانی می‌کنند و سعی دارند شی‌ها و جدول‌هایSQL را به زور وارد حجله کنند (دست‌کم یکی از این محصولات تاکنون شکست خورده است). جزئیات بیشتر از بحث این کتاب خارج است.

بخش را با ارائه دیدگاه رسمی در مورد آنچه گفته شد به پایان می‌برم. گفتم که پایگاه می‌تواند بصورت مجموعه‌ای از گزاره‌های صحیح تصور شود. در حقیقت پایگاه داده‌ها به همراه عملگرهایی که می‌توانند بر روی گزاره‌ها (یا مجموعه‌ای از گزاره‌ها) عمل کنند، تشکیل یک *دستگاه منطقی* می‌دهند. وقتی می‌گویم «یک دستگاه منطقی» منظورم یک دستگاه رسمی (مانند

دستگاه علم هندسه) است که در آن با داشتن *اصول‌موضوعه* (حقایق داده‌شده) *و اصول استنتاج* می‌توان *قضیه‌ها* (حقایق نتیجه‌گیری شده) را اثبات نمود. در حقیقت کاد با بینشی فوق‌العاده مدل رابطه‌ای را در سال 1969 اختراع کرد و گفت پایگاه داده‌ها (بر خلاف نامی که دارد) فقط مجموعه‌ای از *داده‌ها* نیست بلکه مجموعه‌ای از *واقعیت‌ها*ست و یا به بیان دیگر گزاره‌های درست. این گزاره‌ها (که با مرابطه‌های پایه نشان داده می‌شوند) اصول موضوعه دستگاه منطقی مورد بحث هستند و حقایق نتیجه‌گیری‌شده (استنتاجی) گزاره‌های جدیدی هستند که از گزاره‌های موجود بدست می‌آیند. به زبان دیگر این قوانین به ما می‌گویند که چگونه عملگرهای جبر رابطه‌ای را به‌کارگیریم. بنابراین زمانی که سیستم یک عبارت رابطه‌ای را مورد ارزیابی قرار می‌دهد (خصوصا زمانی که به یک کوئری پاسخ می‌دهد)، یک واقعیت را از واقعیت‌های دیگر بدست می‌آورد. در عمل مانند ثابت کردن یک قضیه!

حال که این مطالب را فهمیدیم می‌توانیم ابزاری که منطق برای حل «مساله پایگاه داده‌ها» در اختیارمان قرار می‌دهد را در دست گیریم. به بیانی دیگر سوالاتی مانند:

- پایگاه داده‌ها بایستی در نگاه کاربر چگونه باشد؟
- قیدهای جامعیت چه شکلی دارند و چگونه به نظر می‌رسند؟
- زبان کوئری‌ها چگونه است؟
- چگونه می‌توان به بهترین پیاده‌سازی کوئری‌ها رسید؟
- کلی‌تر، چگونه می‌توان عبارت‌های پایگاه را به بهترین صورت ارزیابی نمود؟
- نتایج چگونه بایستی برای کاربر نمایش داده شوند؟
- اساسا طراحی پایگاه داده‌ها چگونه انجام می‌شود؟

(و سوالات دیگر مانند این‌ها) پیش خواهند آمد. پرسش‌های منطقی نیازمند پاسخ‌های منطقی هستند. بدیهی است که مدل رابطه‌ای مستقیما از این مفاهیم پشتیبانی می‌کند. به عقیده من این مدل مانند دژ مستحکمی، «برحق» و شکست‌ناپذیر است. بر همین اساس معتقدم که سایر «مدل‌های داده‌ای» اصلا در باغ نیستند. من جدا شک دارم که «مدل‌های» دیگر را اساسا بتوان مدل نامید و فقط مدل رابطه‌ای است که می‌تواند مدل نامیده شود. اکثر آنها بجای اینکه مستحکم باشند، تک‌منظوره هستند و در مقابل فقط مدل رابطه‌ای است که بر اساس نظریه مجموعه‌ها و گزاره‌های منطقی بنا شده است. این موضوع را در فصل هشتم کاملا باز خواهم کرد.

خلاصه

مهمترین بخش از این فصل آنجاست که به گزاره‌نماها می‌پردازد (همراه بخش بعد از آن که در مورد رابطه‌ها و نوع‌ها بود). اساسا هر مرابطه R دارای یک گزاره‌نمای متناظر P می‌باشد (گزاره مرابطه R). P *آماده به تفسیر* و یا *بسط داخلی* R شمرده می‌شود و در طی زمان هرگز تغییر نمی‌کند. اگر مقدار فعلی R برابر با r باشد آنگاه r *بسط خارجی* P در حال حاضر است که شامل مجموعه‌ای از تاپل‌هاست و هر تاپل نماینده یک گزاره صحیح است. بسط خارجی در طول زمان تغییر می‌کند. از این رو پایگاه داده‌ها به همراه عملگرهایش می‌توانند بصورت یک دستگاه منطقی در نظر گرفته شوند. در اینجا چند نکته مهم از مطالب این فصل آورده می‌شود:

- فقط مرابطه‌ها قابل به‌روزرسانی هستند. گفتن کلماتی مانند «به‌روزرسانی یک تاپل» و «به‌روزرسانی یک ویژگی» راحت و در عین حال نادرست‌اند.
- هر مرابطه حداقل یک کلید(کاندید) دارد. کلیدها خاصیت یکتایی و کاهش‌ناپذیری دارند. مقدارهای متناظر با کلید تاپل هستند.
- برخی مرابطه‌ها کلید خارجی دارند. SQL از عملیات ارجاعی (مانند CASCADE) پشتیبانی می‌کند. این عملیات ممکن است مفید و کاربردی باشند ولی جزء مدل رابطه‌ای نیستند و دیگر اینکه کلیدهای خارجی هم موضوعی بنیادی نیستند.
- عملگرهای درون ویوها با برقراری تناظر به مرابطه‌های پایه پیاده‌سازی می‌شوند. (این نگاشت به دلیل خاصیت بسته بودن عملی است، برای ویوهای فقط خواندنی این کار آسان است ولی برای ویوهای قابل به‌روزرسانی ساده نیست.) اصل تعویض پذیری می‌گوید نبایستی هیچ تبعیض غیر ضروری و دلخواهی بین مرابطه‌های پایه و مجازی وجود داشته باشد.
- نسبت نوع به رابطه مانند نسبت اسم به جمله است.
- نوع‌ها و رابطه‌ها هر دو برای نشان دادن داده مورد نیازند. (این مطلب فقط مربوط به سطح منطقی است، همانطور که می‌دانید در سطح فیزیکی ساختارهای دیگری سودمند هستند ولیکن به دلیل مجزا بودن این دو سطح، سطح فیزیکی عمدا خارج از قلمرو مدل رابطه‌ای قرار داده شده است.)

تمرین‌ها

1- به زبان خودتان شرح دهید که چرا جملاتی از قبیل «این UPDATE رتبه توزیع‌کنندگانی که در لندن مستقرند را به‌روزرسانی می‌کند» چندان دقیق نیستند. در این مورد جمله‌ای بگویید که تا جایی که می‌تواند دقیق باشد.

2- چرا استفاده از عملگرهای به‌روزرسانی‌های مکان‌دار در SQL، ایده بدی است.

3- تعریف SQL مرابطه‌های TAX_BRACKET، ROSTER و MARRIAGE در بخش «مطالبی بیشتر در مورد کلید کاندید» را بنویسید.

4- چرا جمله «رابطه یک کلید دارد» چندان ملموس و صحیح نیست؟

5- در متن من دلیلی برای فایده کاهش‌ناپذیری آوردم. آیا شما هم می‌توانید دلایل دیگری بیاورید؟

6- «مقدارهای متناظر با کلید اسکالر نیستند هم مانند تاپل‌ها اسکالر هستند.» این جمله را شرح دهید.

7- فرض کنید مرابطه R از درجه n باشد. R حداکثر چند کلید می‌تواند داشته باشد؟

8- مرابطه EMP در بخش «مطالبی بیشتر در مورد کلید کاندید» نمونه‌ای از *خود ارجاعی* مرابطه‌هاست. یک سری داده ساده برای این مرابطه بسازید. آیا این مرابطه ما را ملزم به پشتیبانی از تهی می‌کند؟ (پاسخ: خیر، ولی نشان می‌دهد که تهی می‌تواند چقدر اغوا کننده باشد.) چگونه می‌توان این تمرین را در حالتی که استفاده از تهی غیر مجاز است، حل کرد؟

9- SQL چیزی مانند امکان تغییر نام توتوریال‌دی برای تشخیص کلید خارجی ندارد. چرا؟

10- آیا می‌توان حالتی را تصور کرد که در آن دو مرابطه R1 و R2 با کلید خارجی به یکدیگر مراجعه نمایند؟

11- محصول SQLی که در دسترس دارید را بررسی کنید. کدام عملیات ارجاعی توسط آن پشتیبانی می‌شود؟ تصور می‌کنید کدامیک از آنها سودمند هستند؟ آیا می‌توانید محصولی را تصور کنید که از آنها پشتیبانی نمی‌کند ولی سودمند است؟

12- مدل رابطه‌ای چیزی درباره «پروسجرهای تریگردار» ندارد. آیا این مشکلی است که بر اثر غفلت بوجود آمده است؟ چرا؟ آیا تصور می‌کنید این امکان ضروری است؟ مطلوب چطور؟

13- فرض کنید LSSP ویویی است که به این صورت تعریف شده است (SQL):

```
CREATE VIEW LSSP
    AS ( SELECT S.SNO, S.SNAME, S.STATUS, SP.PNO, SP.QTY
         FROM   S, SP
         WHERE  S.SNO = SP.SNO
         AND    S.CITY = 'London' ) ;
```

حال یک کوئری بر روی این ویو نوشته می‌شود:

```
SELECT DISTINCT LSSP.STATUS, LSSP.QTY
FROM   LSSP
WHERE  LSSP.PNO IN
    ( SELECT P.PNO
      FROM   P
      WHERE  P.CITY <> 'London' )
```

اگر این کوئری بر روی مرابطه‌های پایه نوشته شود چه شکلی خواهد داشت؟

14- در تمرین قبل کلید(های) ویو LSSP کدامند؟

15- محصول SQLی که در دسترس دارید را بررسی کنید. آیا هیچ ویوی قانونی‌ای وجود دارد که اجرای آن با شکست مواجه شود؟ اگر بلی، بسیار مختصر شرح دهید که در چه شرایطی؟ تولید کننده برای عدم توفیق در پشتیبانی مناسب چه توجیهی دارد؟ (این سوال فقط در مورد کوئری‌هاست نه به‌روزرسانی‌ها)

16- محصول SQLای که در دسترس دارید را بررسی کنید. در آن به‌روزرسانی کدام ویوها پشتیبانی می‌شوند؟ بسیار مختصر پاسخ دهید.

17- با استفاده از پایگاه توزیع‌کنندگان و قطعات و یا هر پایگاه مشابه دیگری، مثالی بیاورید که نشان دهد اینکه کدامیک از مرابطه‌ها پایه و کدام مجازی باشد، اختیاری است.

18- محصول SQLای که در دسترس دارید را بررسی کنید (از این نظر همه محصولات مثل هم هستند!). آیا محصول اصل تعویض‌پذیری را زیرپا می‌گذارد؟

19- تفاوت ویوها و تصویرهای لحظه‌ای را شرح دهید. آیا SQL از تصویرهای لحظه‌ای پشتیبانی می‌کند؟ آیا محصولی می‌شناسید که این کار را انجام دهد؟

20- «دید جامه عمل پوشیده» چیست؟ چرا به کار بردن این کلمه بد است؟

21- کلمات گزاره و گزاره‌نما را با مثال شرح دهید.

22- گزاره‌نماهای مربوط به مرابطه‌های P و SP را بنویسید.

23- از اصطلاحات بسط داخلی و بسط خارجی چه می‌فهمید؟

24- فرض کنید DB یکی از پایگاه‌هایی باشد که می‌شناسید و R یکی از مرابطه‌های آن باشد. گزاره‌نمای R چیست؟ توجه: نکته این تمرین آن است که نظرات بنیادی شرح داده شده در این فصل را، بر روی داده‌های خودتان اعمال کنید با این هدف که از این دید به داده‌ها نگاه کنید. این سوال یک جواب مشخص ندارد.

25- ویوهای LS و NLS بخش «مطالبی بیشتر در مورد ویوها» را در نظر بگیرید. گزاره‌نمای متناظر با هر مرابطه چیست؟ آیا اگر بجای اینکه آنها ویو باشند، مرابطه پایه می‌بودند تفاوتی می‌کرد؟

26- گزاره‌نمای مربوط به ویو LSSP تمرین 13 چیست؟

27- *فرض بسته بودن جهان* را شرح دهید.

28- یک کلید مجموعه‌ای از ویژگی‌هاست و یک مجموعه خالی، مجموعه‌ای مجاز است. پس ما می‌توانیم یک کلید خالی تعریف کنیم که مجموعه‌ای خالی از ویژگی‌هاست. آیا فکر می‌کنید چنین کلید به هیچ دردی می‌خورد؟

29- گزاره‌نمای مرابطه‌ای با درجه صفر چیست؟ (آیا پاسخ به این سوال آسان است؟ چرا؟)

30- هر مرابطه دارای یک رابطه بعنوان مقدار است. آیا عکس این صحبت هم درست است؟ اینکه هر مقدار رابطه‌ای دارای یک مرابطه است.

فصل پنجم

جبر رابطه‌ای

این فصل را با یادآوری چند نکته مهم از فصل اول آغاز می‌کنم. اول، یادآوری می‌کنم که هر عملگر جبری دست کم یک رابطه را بعنوان ورودی می‌گیرد و یک رابطه دیگر را به عنوان خروجی تولید می‌کند. دوم، یادآوری می‌کنم این واقعیت را که ورودی‌ها و خروجی و هم‌نوع هستند (و در اینجا همگی رابطه)، خاصیت بسته بودن جبر می‌گویند. همین ویژگی است که به ما اجازه نوشتن عبارت‌های جبری تودرتو را می‌دهد. سوم، در فصل اول بصورت اجمالی «هشت عملگر اصلی» (گزینش، پرتو، ضرب، اشتراک، اجتماع، تفاضل، پیوند و تقسیم) را توضیح دادم. در اینجا می‌خواهم این عملگرها را – مثل بقیه مطالب- با دقت بیشتری شرح دهم ولی قبل از آن بایستی چند نکته را بدانید:

- اول، عملگرها عمومی هستند. آنها می‌توانند بر تمامی رابطه‌های ممکن اعمال شوند. مثلا به یک پیوند خاص برای پیوند رابطه کارمندها و ادارات و یک پیوند دیگر برای پیوند توزیع‌کنندگان و سفارش‌ها، نیازی نداریم. (به نظر شما این قضیه در مورد سیستم‌های شی‌گرا هم صادق است؟)
- دوم، عملگرها فقط خواندنی هستند. آنها عملوندها را می‌خوانند و نتیجه را برمی‌گردانند ولی چیزی را به‌روزرسانی نمی‌کنند. به بیان دیگر آنها بر روی *رابطه‌ها* کار می‌کنند نه روی *مرابطه‌ها*.
- البته، این بدان معنا نیست که عملگرها نمی‌توانند به مرابطه‌ها مراجعه کنند. مثلا R UNION S در حالی که R و S نام دو مرابطه هستند در توتوریال‌دی عبارتی صحیح به شمار می‌رود. در مورد چنین عباراتی می‌توان گفت نام مرابطه در اینجا متناظر با خود مرابطه نیست بلکه اشاره آن به مقدار فعلی آن در این لحظه از زمان است. به بیان دیگر می‌توان از نام یک مرابطه بعنوان یک عملوند استفاده کرد و عبارت *مراجعه‌کننده به مرابطه*، یک عبارت رابطه‌ای

صحیح است* ولی در اصل بایستی ثابت رابطه‌ای مربوطه مورد استفاده قرار گیرد. (یک مقایسه ممکن است به درک این مطلب کمک کند. فرض کنید N یک متغیر از نوع عدد صحیح است و در زمان t دارای مقدار 3 می‌باشد. در نتیجه N+2 عبارتی صحیح است ولی فقط در لحظه t این عبارت معادل 2+3 خواهد بود. نه کمتر و نه بیشتر.)

- آخری، بدیهی است عملگرهای جبر رابطه‌ای همگی فقط خواندنی‌اند. عملگرهای INSERT و DELETE و UPDATE (و همچنین انتساب رابطه‌ای) عملگرهای رابطه‌ای *هستند* ولی جزء جبر رابطه‌ای *نیستند*.

اینجا لازم است توضیحی در مورد ساختار توتوریال‌دی بدهم چرا که نحوه پشتیبانی آن از جبر رابطه‌ای تفاوت معناداری با SQL دارد (بهتر است بگویم که عمدا از جبر پشتیبانی مستقیم‌تری به عمل می‌آورد). نکته مهم این است که در مورد عملگرهایی چون UNION و JOIN که نیازمند وجود تناظر بین ویژگی‌های عملوندها هستند، در توتوریال‌دی بایستی این ویژگی‌ها هم‌نام (و در عین حال هم‌نوع) باشند. مثلا عبارت زیر در توتوریال‌دی، قطعات و توزیع‌کنندگان را بر روی شهر پیوند می‌دهد:

```
P JOIN S
```

طبق تعریف این پیوند بر پایه شهر توزیع‌کنندگان و قطعات انجام می‌شود، P و S تنها در ویژگی CITY با هم اشتراک دارند. حال به همین عمل در SQL توجه کنید (خصوصا به خط آخر):

```
SELECT P.PNO, P.PNAME, P.COLOR, P.WEIGHT, P.CITY,
   S.SNO, S.SNAME, S.STATUS
FROM   P,S
WHERE  P.CITY = S.CITY
```

توجه: این مثال می‌تواند به صورت‌های مختلفی عبارت‌نویسی شود که در اینجا سه حالت را مشاهده می‌کنید. می‌بینید که دومین و سومین حالت به فضای توتوریال‌دی کمی نزدیک‌ترند. (توجه کنید که این دو عبارت ستونی بنام CITY دارند و نه P.CITY و یا S.CITY).

---

* ولی نه در SQL! مثلا اگر R و S نام دو جدول SQL باشند نمی‌توانیم عبارت *R* UNION *S* را بنویسیم بلکه بایستی بجای آن از چنین عبارتی استفاده کنیم: SELECT *R*.* FROM *R* UNION SELECT *S*.* FROM *S*

```
SELECT P.PNO, P.PNAME, P.COLOR, P.WEIGHT, P.CITY,
   S.SNO, S.SNAME, S.STATUS
FROM   P JOIN S
ON   P.CITY = S.CITY

SELECT P.PNO, P.PNAME, P.COLOR, P.WEIGHT, CITY,
   S.SNO, S.SNAME, S.STATUS
FROM   P JOIN S
USING  ( CITY )

SELECT P.PNO, P.PNAME, P.COLOR, P.WEIGHT, CITY,
   S.SNO, S.SNAME, S.STATUS
FROM   P NATURAL JOIN S
```

ولی از میان حالت‌های مختلف من این عبارت خاص را انتخاب کردم چرا که تنها عبارتی است که با SQL اولیه مطابقت دارد و از آن مهم‌تر اینکه مرا قادر می‌سازد چند نکته دیگر در مورد تفاوت‌های SQL و جبر را توضیح دهم (حداقل در محدوده توتوریال‌دی).

- در SQL نام‌های نقطه دار مجاز (و در برخی موارد ضروری) است ولی توتوریال‌دی چنین نیست.
- در برخی موارد در توتوریال‌دی تغییر نام ویژگی‌ها اجتناب ناپذیر است چرا که در نام‌ها دچار تکرار یا عدم تطابق می‌شویم ولی در SQL معمولا چنین اتفاقی نمی‌افتد (با این وجود SQL دارای روشی مانند RENAME در توتوریال‌دی است که برای اهدافی مانند آنچه در بخش بعد خواهیم دید، مورد استفاده قرار می‌گیرد).
- در نتیجه توتوریال‌دی نیازی به «نام‌های مرتبط شده» SQL ندارد (در عمل روش تغییر نام ویژگی‌های مورد نظر جهت «رفع ابهام»، به جای نام‌های مرتبط شده به کار رفته است و در صورت عدم انجام این کار با خطای دستوری روبرو خواهیم شد).
- علاوه بر پشتیبانی ضمنی از برخی ویژگی‌های جبر رابطه‌ای، SQL از حساب رابطه‌ای نیز پشتیبانی می‌کند (بطور خاص به EXISTS. ضمیمه مراجعه کنید). توتوریال‌دی چنین امکانی ندارد. در نتیجه این تفاوت، SQL به زبانی پر زائده

تبدیل می‌شود که معمولا در آن برای نوشتن یک کوئری راه‌های بی‌شماری وجود دارد (واقعیتی که برای بهنیه‌ساز مانع جدی به حساب می‌آید).*

- کوئری‌های SQL بایستی با الگوی SELECT-FROM-WHERE مطابقت داشته باشند. توتوریال‌دی چنین پیش‌فرضی ندارد. در این فصل و در بخش گسترش دادن و خلاصه‌کردن در این مورد بیشتر خواهم گفت.

در ادامه مثال‌هایی از توتوریال‌دی و SQL به شما نشان خواهم داد.

در مورد بسته بودن بیشتر بدانیم

امیدوارم فهمیده باشید که وقتی گفتم خروجی هر عملیات جبری یک رابطه جدید است، از یک دیدگاه خیالی به قضیه نگاه می‌کردم. منظور این نبود که خروجی کاملا جامه عمل پوشیده است. مثلا مطابق با مثال زیر (گزینش از پیوند، سمت چپ توتوریال‌دی و سمت راست SQL):

```
( P JOIN S )                | SELECT P.*,
WHERE PNAME > SNAME         |   S.SNO, S.SNAME, S.STATUS
                            | FROM   P, S
                            | WHERE  P.CITY = S.CITY
                            | AND  P.PNAME > S.SNAME
```

واضح است به همان سرعتی که تاپل پیوند شکل می‌گیرد، سیستم گزینش شرط PNAME>SNAME (در SQL بصورت P.PNAME>S.SNAME) را بررسی می‌کند و می‌گوید که تاپل بایستی در نتیجه نهایی ظاهر شود یا نه. بنابراین نتیجه فوری که از پیوند حاصل می‌شود ممکن است هرگز بطور کامل جامه عمل نپوشد و در خروجی نهایی ظاهر نشود. (واقعیت این است که سیستم به سختی تلاش می‌کند تا نتایج فوری بطور کامل جامه عمل *نپوشند*، دلیل این کار بالا بردن کارایی است.)

مثال قبل نکته مهم دیگری را نیز روشن می‌کند. به عبارت بولین PNAME>SNAME در عبارت توتوریال‌دی دقت کنید. این عبارت حاصل P JOIN S را درخواست می‌کند و

* در گذشته مقاله‌ای نوشته‌ام با عنوان «پنجاه راه برای نوشت کوئری‌شما» (www.dbpd.com جولای 1998)، که در آن نشان دادم کوئری ساده‌ای چون «نام توزیع‌کنندگانی که قطعه P2 را توزیع می‌کنند» را می‌توان به پنجاه فرم مختلف در SQL نوشت.

SNAME و PNAME در واقع به ویژگی‌های نتیجه اشاره می‌کنند *نه* ویژگی‌های مرابطه‌های S و P. ولی از کجا می‌فهمیم که نتیجه دارای چنین ویژگی‌هایی است؟ عنوان رابطه بدست آمده چیست؟ و بطور کلی چگونه عنوان حاصل یک عبارت جبری را پیدا می‌کنیم؟ معلوم است که به مجموعه‌ای از *قواعد پیداکردن* نیازمندیم و بطور دقیق قواعد پیداکردن نوع رابطه. بدین صورت اگر نوع رابطه(های) ورودی را بدانیم می‌توانیم نوع رابطه خروجی را پیدا کنیم. در مورد پیوند فوق این قواعد می‌گویند که خروجی P JOIN S از این نوع خواهد بود:

```
RELATION { PNO PNO, PNAME NAME, COLOR COLOR, WEIGHT
WEIGHT,
        CITY CHAR, SNO SNO, SNAME NAME, STATUS INTEGER }
```

در حقیقت عنوان خروجی برابر با *اجتماع* عنوان ورودی‌ها خواهد بود. اجتماعی که از آن صحبت می‌کنم اجتماع معمولی مجموعه‌هاست و نه اجتماع رابطه‌ها که در این فصل در مورد آن توضیح می‌دهم. به بیان دیگر عنوان خروجی محتوی تمام ویژگی‌های ورودی‌هاست به غیر از ویژگی‌های مشترک – در این مثال فقط CITY- که فقط یک‌بار ظاهر می‌شود نه دوبار. این ویژگی‌ها فاقد ترتیب – راست به چپ- هستند و من می‌توانم نوع نتیجه P JOIN S را به این صورت هم بنویسم:

```
RELATION { SNO SNO, PNO PNO, SNAME NAME, PNAME NAME,
CITY CHAR, STATUS INTEGER, WEIGHT WEIGHT, COLOR COLOR }
```

توجه داشته باشید که بسته بودن پیش نیاز قواعد پیداکردن نوع است. بسته بودن می‌گوید نتیجه یک رابطه است و رابطه‌ها فقط بدنه ندارند بلکه عنوان هم دارند، پس هر نتیجه بایستی دارای یک عنوان مناسب – و همینطور یک بدنه مناسب- باشد.

عملگر RENAME که در این فصل معرفی شده، به پشتیبانی وسیع مدل رابطه‌ای از قواعد پیداکردن نوع نیازمند است. RENAME یک رابطه را بعنوان ورودی دریافت می‌کند و یک رابطه دیگر را به عنوان خروجی بازمی‌گرداند. رابطه خروجی با ورودی یکسان است با این تفاوت که نام یکی از ویژگی‌ها در آن عوض شده است. برای مثال (سمت چپ توتوریال‌دی و سمت راست SQL):

```
S RENAME ( CITY AS SCITY ) | SELECT S.SNO, S.SNAME, S.STATUS,
                           |        S.CITY AS SCITY
                           | FROM   S
```

با مقدار همیشگی مثال‌مان چنین نتیجه‌ای تولید می‌شود:

SNO	SNAME	STATUS	SCITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

(در این فصل معمولا نتایج را به تصویر در نخواهم آورد، مگر زمانی که حس کنم شما با عملگر مورد بحث ناآشنا هستید، مثل همین مورد.)

توجه

مهم: مثال قبل مرابطه S را در پایگاه تغییر *نمی‌دهد!* RENAME شبیه دستور ALTER TABLE در SQL نیست. RENAME فقط یک فراخوانی عبارت است (مثل P JOIN S و یا N+2) و مانند همه عبارات فقط یک مقدار مشخص برمی‌گرداند (البته تا وقتی که فقط یک عبارت باشد نه یک دستور) این عبارت می‌تواند داخل عبارات دیگر قرار گیرد و در آینده مثال‌های بسیاری از عبارت‌ها خواهیم دید.

حال ببینیم SQL با نام ویژگی‌های نتیجه (یا نام ستون‌ها) چه کار می‌کند؟ پاسخ این است که خیلی خوب عمل نمی‌کند. اولا در فصل سوم دیدیم که در SQL «نوع رابطه» جایی ندارد (بجای آن نوع سطر وجود دارد). ثانیا ممکن است در نتیجه ستون‌هایی به وجود آید که اصلا نام ندارند. مثلا:

```
SELECT DISTINCT P.WEIGHT * 2 FROM P
```

ثالثا ممکن است ستون‌هایی با دو نام ایجاد شوند مثلا:

```
SELECT DISTINCT P.CITY, S.CITY FROM P, S
```

نهایتا بیایید دوباره به مثال اول فصل نگاهی بیاندازیم:

```
( P JOIN S )                | SELECT P.*,
WHERE PNAME > SNAME         | S.SNO, S.SNAME, S.STATUS
                            | FROM   P, S
                            | WHERE  P.CITY = S.CITY
                            | AND  P.PNAME > S.SNAME
```

به عبارت مقایسه که در توتوریال‌دی، PNAME>SNAME و در SQL، P.PNAME>S.SNAME است دقت کنید. کدامیک از آنها غیر منطقی است؟ می‌دانید که این عبارت شرطی به نتیجه FROM اشاره می‌نماید و S یا P مسلما در نتیجه FROM وجود ندارند! توضیح اینکه چگونه چیزی مانند P.PNAME در بخشی مانند SELECT یا WHERE (یا هر جای دیگر عبارت) ظاهر شده است، واقعا مشکل است. استاندارد SQL این امر را توضیح می‌دهد ولی طرح آن *خیلی* پیچیده‌تر از قواعد توتوریال‌دی است. آنقدر پیچیده که من آن را در اینجا توضیح نمی‌دهم ولی فرض را بر آن می‌گذارم که توجیه SQL قابل قبول است و این قواعد در صورت نیاز قابل اثبات هستند. برای توجیه این کوتاهی یادآوری می‌کنم که این کتاب در مورد مدل رابطه‌ای است نه در مورد SQL.

حال می‌خواهم تعدادی از عملگرهای جبری را شرح دهم. توجه داشته باشید که من قصد ندارم آنها را با همه جزئیات توضیح دهم و نیز نمی‌خواهم که «تمام عملگرهای شناخته شده» را پوشش دهم. در اکثر موارد من تعریفی درست اما غیر رسمی از این عملگرها را به همراه چند مثال ارائه می‌کنم.

عملگرهای اصلی

در این فصل عملگرهای اصلی جبر رابطه‌ای که توسط کاد تعریف شدند، را شرح می‌دهم. این عملگرها عبارتند از: گزینش، پرتو، پیوند، اشتراک، اجتماع، تفاضل، ضرب و تقسیم.

گزینش

اگر *bx* عبارتی بولی باشد که دارای صفر یا بیشتر نام ویژگی است و تمام این ویژگی‌ها متعلق به رابطه *r* باشند.* آنگاه گزینش *r* بر پایه *bx*:

r WHERE *bx*

رابطه‌ای است با عنوانی همانند *r* و بدنه‌ای محتوی تمامی تاپل‌های *r* که *bx* بر اساس آنها صحیح ارزیابی می‌شود. برای مثال:

```
S WHERE CITY = 'Paris'     | SELECT S.*
                           | FROM   S
                           | WHERE  S.CITY = 'Paris'
```

بعنوان حاشیه، گزینش برخی اوقات *select* خوانده می‌شود. من استفاده از این واژه را توصیه نمی‌کنم زیرا ممکن است با SELECT زبان SQL اشتباه شود.†

پرتو

فرض کنید رابطه *r* دارای ویژگی‌های $X,Y,...,Z$ (و احتمالا ویژگی‌های دیگر) است. آنگاه پرتو *r* بر روی $X,Y,...Z$ که عبارت است از:

r { *X*, *Y*,..., *Z* }

رابطه‌ای است با الف) یک عنوان که از عنوان *r* و با حذف تمامی ویژگی‌هایی که در $\{X,Y,...,Z\}$ حضور ندارند، بدست آمده است. و ب) یک بدنه محتوی تمام تاپل‌های به شکل $\{X_x,Y_y,...,Z_z\}$ به گونه‌ای که هر تاپل با مقدار *x* برای *X*، *y* برای *Y* و... *z* برای *Z* از *r* در آن وجود دارد. مثلا:

---

* عبارت بولی محدودیت‌های دیگری نیز دارد. تمرین 6 همین فصل را ببینید.

† از آنجا که امکان چنین اشتباهی پس از ترجمه وجود ندارد به هنگام ترجمه به فارسی بجای کلمه محدودیت *restrict* از گزینش استفاده نموده‌ام. مترجم

```
S { SNAME, CITY, STATUS }          | SELECT DISTINCT S.SNAME,
                                   |        S.CITY, S.STATUS
                                   | FROM   S
```

دوباره می‌گویم، نتیجه یک رابطه است بنابراین «تکرارها رفع می‌شوند». از این رو وجود DISTINCT در عبارات SQL واقعا ضروری است.

و اما پرتو در توتوریال‌دی می‌تواند با تمام ویژگی‌ها به جز یک ویژگی نیز انجام شود. برای مثال عبارت‌های:

```
S { SNAME, CITY, STATUS }
```

و همچنین

```
S { ALL BUT SNO }
```

معادل یکدیگرند. این امکان موجب صرفه‌جویی بسیار در نوشتن می‌شود. (تصور کنید که چگونه باید 99 ویژگی از یک رابطه با درجه 100 را پرتو کرد). چنین امکانی در مورد SUMMARIZE و GROUP هم وجود دارد (در قسمت‌های بعد «گسترش‌دادن و خلاصه‌کردن» و همچنین «گروه‌بندی و از گروه درآوردن» را ملاحظه کنید).

اولویت پرتو بالاست. بنابراین در توتوریال‌دی عبارت زیر:

```
S JOIN P { PNO, CITY }
```

به معنای این است:

```
S JOIN ( P { PNO, CITY } )
```

نه به معنای این:

```
( S JOIN P ) { PNO, CITY }
```

برای تمرین و با استفاده از داده‌های معمولی مثال خودمان، تفاوت بین این دو تفسیر را نشان دهید.

پیوند

فرض کنید r رابطه‌ای با ویژگی‌های $X_1, X_2,..., X_m, Y_1, Y_2,..., Y_n$ و رابطه s با ویژگی‌های $Y1, Y2,..., Yn, Z1, Z2,..., Zp$ وجود دارند. به بیان دیگر Yها (n تا) ویژگی‌های مشترک‌اند و Xها (m تا) ویژگی‌های اختصاصی r و Zها (p تا) ویژگی‌های اختصاصی s هستند. بدون استثنا می‌توان فرض کرد که هیچیک از Xها و Zها با یکدیگر همنام نیستند، عملی شدن

این فرض مدیون RENAME است. حال تصور کنید که همه Xها را به اختصار X می‌نامیم و همین کار را در مورد Yها و Zها انجام می‌دهیم. آنگاه *پیوند طبیعی*[*] (یا به اختصار *پیوند*) r و s:

r JOIN s

الف) دارای یک عنوان که اجتماع (مجموعه‌ای) عنوان‌های r و s است. ب) دارای یک بدنه که محتوی تمام تاپل‌هایی است مانند t که t اجتماع (مجموعه‌ای) یک تاپل از r و یک تاپل از s است. به زبان دیگر عنوان به شکل $\{X,Y,Z\}$ و بدنه محتوی تمام تاپل‌های به صورت $\{X_x,Y_y,Z_z\}$ است، به گونه‌ای که تاپل‌های موجود در r برای X دارای مقدار x و برای Y دارای مقدار y و تاپل‌های موجود در s برای Y دارای مقدار y و برای Z دارای مقدار z هستند. باز هم یک مثال از فصل مقدمه:

```
P JOIN S | SELECT P.PNO, P.PNAME, P.COLOR, P.WEIGHT, P.CITY,
         |      S.SNO, S.SNAME, S.STATUS
         | FROM   P, S
         | WHERE  P.CITY = S.CITY
```

کاملا مطمئنم که شما با مطالب گفته شده در مورد پیوند آشنا بودید ولی ممکن است این نکات را تاکنون نشنیده باشید. اول، یادآوری می‌کنم که پیوند توزیع‌کنندگان و قطعات بر روی شهرها می‌تواند با استاندارد SQL به گونه‌ای نوشته شود که تا حدودی شبیه *توتوریال‌دی* باشد (و این بار بجای فهرست طولانی نام ستون‌ها مقابل SELECT، از * استفاده می‌کنم):

```
SELECT *
 FROM   P NATURAL JOIN S
```

هر چند تمامی محصولات SQL از این روش پشتیبانی نمی‌کنند.

دوم و مهمتر، *اشتراک* نوع خاصی از پیوند است. بطور دقیق اگر $m=p=0$ (بدین معنا که X و Zای وجود نداشته باشد و در رابطه هم نوع باشند)، r JOIN s به r INTERSECT s تنزل مقام پیدا می‌کند (در قسمت بعد اشتراک را ببینید).

سوم و باز هم مهم، ضرب دکارتی هم نوع خاصی از پیوند است: اگر $n=0$ بدین معنا که هیچ Yای موجود نباشد و r و s هیچ ویژگی مشترکی نداشته باشند، r JOIN s به r TIMES s

[*] انواع دیگر پیوند را در قسمت بعد بررسی می‌کنم.

تنزل می‌یابد (در قسمت‌های بعد ضرب دکارتی را ببینید). چرا؟ به این دلایل: الف)مجموعه ویژگی‌های مشترک r و s تهی است. ب) تمامی تاپل‌های ممکن برای مجموعه‌ای تهی از ویژگی‌ها، دارای مقدار یکسان هستند (تاپل صفر). ج) هر تاپل از r به تمامی تاپل‌های s پیوند می‌خورد و به این ترتیب ضرب دکارتی ایجاد می‌شود.

چهارم، در عمل مشاهده می‌شود که بسیاری از کوئری‌ها نیازمند نوع توسعه داده شده پیوند هستند، که *نیم‌پیوند* نامیده می‌شود. (ممکن است تا بحال نام نیم‌پیوند را نشنیده باشید ولی این موضوع بسیار مهم است.) تعریف *نیم پیوند* r با s چنین است: r و s پیوند یافته و سپس نتیجه بر روی ویژگی‌های r پرتو می‌شود. مثلا کوئری «توزیع‌کنندگانی را بده که حداقل یک قطعه را توزیع می‌کننده را بده»:

```
S SEMIJOIN SP          | SELECT DISTINCT S.*
                       | FROM   S, SP
                       | WHERE  S.SNO = SP.SNO
```

همانطور که می‌بینید این کوئری می‌خواهد لیست توزیع‌کنندگانی که حداقل یک قطعه توزیع نموده‌اند را درخواست کند. *توتوریال‌دی* راه دوستانه‌تری برای این کار دارد:

```
S MATCHING SP
```

حال تصور کنید که اگر $p=0$ باشد چه اتفاقی برای r JOIN s می‌افتد (منظور این است که عنوان s زیرمجموعه عنوان r است و یا هر ویژگی s در r هم وجود دارد). امیدوارم فهمیده باشید که در این حالت r JOIN s تنزل یافته s MATCHING r است (توجه داشته باشید که s MATCHING r و r MATCHING s با هم تفاوت دارند).

پنجم، پیوند اساسا عملگری دوتایی است، بدین معنا که دو عملوند دارد. با این وجود نوع nتایی و پیشوندی آن ممکن و مفید است (و *توتوریال‌دی* از آن پشتیبانی می‌کند)، بنابراین می‌توانیم عباراتی را به این شکل بنویسیم:

```
JOIN { r, s,..., w }
```

تا هر تعداد رابطه را به هم پیوند دهد. برای مثال پیوند بین قطعات و توزیع‌کنندگان می‌تواند به این شکل نوشته شود:

```
JOIN { P, S }
```

از این شکل دستور می‌توانیم برای ضرب یک رابطه و یا حتی هیچ رابطه! استفاده کنیم. حاصل پیوند یک رابطه مثل JOIN{r}، خود r است. چنین به نظر می‌رسد که این مورد استفاده

چندانی ندارد در عوض پیوند هیچ رابطه یا JOIN{} مهم است و نتیجه آن DEE می‌باشد. (یادآوری می‌کنم که DEE یک رابطه خاص بدون ویژگی و با یک تاپل است.) چرا نتیجه DEE می‌شود؟ با توجه به این دلایل:

- در ریاضیات 0 عضو خنثی «+» است. بدین معنا که هر چه باشد x، آنگاه $x+0$ و $0+x$ برابر x خواهد بود. در نتیجه *حاصل جمع هیچ عدد صفر است*. (برای اینکه ببینید این امر معقول است، قطعه برنامه‌ای را در نظر بگیرید که حاصل جمع n عدد را محاسبه می‌کند به این صورت که ابتدا حاصل جمع را صفر قرار می‌دهد سپس اعداد را یکی یکی با آن جمع می‌کند. اگر $n=0$ باشد چه اتفاقی می‌افتد؟)
- به همین صورت، 1 عضو خنثی «×» است. بدین معنا که هر چه باشد x، آنگاه $x\times1$ و $1\times x$ برابر x خواهد بود. در نتیجه حاصل ضرب هیچ عدد صفر است.
- در جبر رابطه‌ای DEE *عضو خنثی پیوند است*. بدین معنا که حاصل پیوند هر رابطه مانند r با DEE خود r خواهد بود. در نتیجه حاصل پیوند هیچ رابطه DEE خواهد بود.

اگر درک این نظریه برایتان سخت است، فعلا زیاد نگران آن نباشید ولی بعدا برای دوباره خواندن آن بازگردید. توصیه می‌کنم خودتان را متقاعد کنید که هر دو r JOIN DEE و DEE JOIN r برابر r می‌شوند. این امر موجب می‌شود این واقعیت که پیوند مورد بحث در حقیقت همان ضرب دکارتی است، قابل قبول باشد.

اشتراک

فرض کنید r و s دو رابطه هم‌نوع باشند آنگاه اشتراک این دو رابطه، رابطه‌ای از همان نوع خواهد بود با بدنه‌ای که حاوی تمام تاپل‌هایی مانند t است به گونه‌ای که t هم در r و هم در s وجود دارد. برای مثال:

```
S { CITY }          | SELECT DISTINCT S.CITY
  INTERSECT         | FROM   S
  P { CITY }        | INTERSECT
                    | SELECT DISTINCT P.CITY
                    | FROM   P
```

(در اینجا واقعا به DISTINCTها نیازی نیست ولی من به دلیل وضوح آنها را توصیه می‌کنم. در این مورد در بخش اجتماع بحث می‌شود.)

همانطور که ملاحظه کردید اشتراک نوع خاصی از پیوند است با این حال برای راحتی بیشتر کاربران، توتوریال‌دی و SQL هر دو از آن پشتیبانی می‌کند. توتوریال‌دی همچنین از نوع پیشوندی و nتایی آن پشتیبانی می‌کند که در اینجا از توضیح آن می‌گذرم.

اجتماع

دوباره، فرض کنید r و s هم‌نوع باشند آنگاه اجتماع این دو رابطه، رابطه‌ای از همان نوع خواهد بود با بدنه‌ای که حاوی تمام تاپل‌هایی مانند t است و به گونه‌ای که t در r و یا در s و یا در هر دو وجود دارد. برای مثال:

```
S { CITY } UNION P { CITY }
```

```
| SELECT DISTINCT S.CITY
| FROM   S
| UNION  DISTINCT
| SELECT DISTINCT P.CITY
| FROM   P
```

در مورد اجتماع نکته ارزشمندی وجود دارد: «تکرار برطرف می‌شود». در SQL وجود DISTINCT بعد از کلمه UNOIN ضروری نیست با این حال این کلمه در اینجا همان نقشی را دارد که در SELECT نیز ایفا می‌کرد (DISTINCT در مقابلALL)، در اجتماع DISTINCT پیش‌فرض است نه ALL (درست برعکس SELECT، یادآوری از فصل سوم). در نتیجه هیچکدام از DISTINCTهای موجود در دو SELECT لازم نیستند و البته غلط هم نیستند. این امر در مورد اشتراک هم صدق می‌کند (همچنین تفاضل که بزودی خواهیم دید).

توتوریال‌دی از اجتماع مجزا و یا D_UNION پشتیبانی می‌کند. نوعی اجتماع که در آن عملوندها مجزا هستند بدین معنا که هیچ تاپل مشترکی بین آنها وجود ندارد. مثلا:

```
S { CITY } D_UNION P { CITY }
```

با مقدارهای همیشگی مثال خودمان، این عبارت منجر به صدور پیام خطا می‌شود چرا که شهرهای قطعه‌ها و شهرهای توزیع‌کنندگان مجزا نیستند. مشابه این عبارت در SQL می‌تواند به این شکل باشد:

```
SELECT *
  FROM ( SELECT S.CITY FROM S
         UNION
         SELECT P.CITY FROM P ) AS POINTLESS
  WHERE  NOT EXISTS
     ( SELECT S.CITY FROM S
       INTERSECT
       SELECT P.CITY FROM P )
```

این دو تا حدودی با هم متفاوت‌اند. اگر شهرهای قطعه‌ها و شهرهای توزیع‌کنندگان مجزا نباشند، SQL اعلام خطا نمی‌کند بلکه یک نتیجه خالی بازمی‌گرداند. همچنین بایستی بگویم که همه نگارش‌های SQL در مقابل FROM از زیرکوئری‌های تودرتو (به صورتی که اینجا به‌کار رفته) پشتیبانی نمی‌کنند با این وجود SQL استاندارد این پشتیبانی را دارد.

توجه

نام POINTLESS در اینجا واقعا بی‌خاصیت است ولی به دلیل رعایت قواعد دستوری، نوشتن آن ضروری است.

توتوریال‌دی همچنین از نوع پیشوندی و nتایی اجتماع و اجتماع مجزا پشتیبانی می‌کند که اینجا از آنها می‌گذرم.

تفاضل

باز هم فرض کنید r و s هم‌نوع باشند آنگاه تفاضل این دو رابطه (r منهای s)، رابطه‌ای از همان نوع خواهد بود با بدنه‌ای که حاوی تمام تاپل‌هایی مانند t است به گونه‌ای که t در r هست و در s نیست. برای مثال:

```
S { CITY } MINUS P { CITY }
```

```
| SELECT S.CITY
| FROM   S
| EXCEPT
| SELECT P.CITY
| FROM   P
```

عملگری مربوط به پیوند که نیم‌پیوند نام داشت را به یاد آورید. حال نوبت نیم‌تفاضل است ولی در این مورد عملگرها چندان «بجا» نیستند، خود تفاضل نوع خاصی از نیم‌تفاضل است (می‌توان گفت تمام تفاضل‌های عادی، نیم تفاضل‌اند). و اگر نیم‌پیوند به نوعی از پیوند مهم‌تر است در مورد تفاضل هم چنین است ولی با شدت بیشتر. اکثر کوئری‌هایی که نیازمند تفاضل به نظر می‌رسند در حقیقت به نیم‌تفاضل نیاز دارند. اینجا یک تعریف می‌بینید. نیم‌تفاضل r و s عبارت است از تفاضل r از نیم‌پیوند r و s. یعنی:

r SEMIMINUS s = r MINUS (r MATCHING s)

برای مثال برای کوئری «نام توزیع‌کنندگانی که هیچ قطعه‌ای را توزیع نمی‌کنند را بده»:

```
S SEMIMINUS SP          | SELECT S.*
                        | FROM   S
                        | EXCEPT
                        | SELECT S.*
                        | FROM   S, SP
                        | WHERE  S.SNO = SP.SNO
```

در توتوریال‌دی می‌توان این کوئری را به شکل دوستانه‌تری نوشت:

```
S NOT MATCHING SP
```

برای اینکه بدانید تفاضل حالت خاصی از نیم‌تفاضل است، وضعیتی را در نظر بگیرید که r و s هم‌نوع باشند (جزئیات را به شکل یک تمرین خواهم گفت).

ضرب دکارتی

من این عملگر را فقط به دلیل کامل بودن مطلب توضیح می‌دهم؛ همانطور که دیدیم ضرب فقط نوع خاصی از پیوند است و به حضورتان عرض می‌کنم که توتوریال‌دی هم مستقیما از آن پشتیبانی نمی‌کند ولی برای گسسته نشدن بحث اجازه دهید که فرض کنیم این پشتیبانی وجود دارد. در این شرایط عملوندها نبایستی دارای هیچ ویژگی‌ای با نام مشترک باشند (وگرنه چه اتفاقی می‌افتد؟)، و عنوان نتیجه اجتماع (مجموعه‌ای) عنوان عملوندهای ورودی خواهد بود. برای مثال:

```
( S RENAME ( CITY AS SCITY ) )
TIMES
( P RENAME ( CITY AS PCITY ) )
```

و در SQL:

```
SELECT S.SNO, S.SNAME, S.STATUS, S.CITY AS SCITY
    P.PNO, P.PNAME, P.COLOR, P.WEIGHT, P.CITY AS PCITY
FROM   S, P
```

تقسیم

طبق قولی که در مقدمه این فصل دادم، تعریف کامل این عملگر را در اینجا ارائه می‌کنم ولی به دلایلی، زیاد وارد جزئیات نمی‌شوم. مهمترین دلیل این کار آن است که کوئری‌هایی که با تقسیم نوشته می‌شود می‌توانند به روش دیگر (و به نظر من ساده‌تری) نوشته شوند که همان استفاده از *انتساب رابطه‌ای* است (در مورد انتساب رابطه‌ای در همین فصل صحبت خواهم کرد). از این رو ممکن است شما از این مبحث (حداقل در دور اول مطالعه کتاب) بگذرید.

دلیل دیگر وارد نشدن به جزئیات این است که چند نوع «تقسیم» مختلف وجود دارد. اینها در حقیقت عملگرهای متفاوتی هستند که متاسفانه همگی «تقسیم» نامیده شده‌اند و مسلما من قصد ندارم همه آنها را اینجا توضیح دهم. در عوض من توجه‌ام را به تقسیم اصلی و در عین حال ساده‌ترین تقسیم محدود می‌کنم. رابطه‌های r و s را در نظر بگیرید به گونه‌ای که عنوان s زیرمجموعه عنوان r باشد. آنگاه حاصل تقسیم r بر s، خلاصه شده عبارت زیر است[*]:

r { X } MINUS ((r { X } TIMES s) MINUS r) { X }

X در اینجا تفاضل (مجموعه‌ای) عنوان r و s است. بنابراین برای مثال این عبارت:

SP { SNO, PNO } DIVIDEBY P { PNO }

با توجه به مقدارهای مثال خودمان چنین نتیجه‌ای می‌دهد:

SNO
S1

(تقریبا شماره توزیع‌کنندگانی که تمام قطعات را توزیع می‌کنند. دلیل اینکه چرا «تقریبا» را در بخش «مقایسه‌های رابطه‌ای» این فصل توضیح خواهم داد). مشابه این عبارت در SQL:

[*] توتوریال‌دی به طور مستقیم از « تقسیم اصلی و در عین حال ساده‌ترین تقسیم» پشتیبانی نمی‌کند و چنین عبارتی در این زبان مجاز نیست.

```
SELECT DISTINCT SPX.SNO
FROM   SP AS SPX
WHERE  NOT EXISTS
  ( SELECT P.PNO
    FROM   P
    WHERE  NOT EXISTS
     ( SELECT SPY.SNO
       FROM   SP AS SPY
       WHERE  SPY.SNO = SPX.SNO
       AND  SPY.PNO = P.PNO ) )
```

تا بحال شنیده‌اید که چرا اسم تقسیم را تقسیم گذاشته‌اند؟ به این دلیل که اگر r و s رابطه‌هایی بدون ویژگی مشترک باشند و ما حاصل ضرب دکارتی s TIMES r را محاسبه کنیم و سپس حاصل را بر s تقسیم کنیم، مجددا به r خواهیم رسید.* به عبارت دیگر ضرب دکارتی و تقسیم، عکس یکدیگر هستند.

کدام عملگرها اولیه‌اند؟

همانطور که قبلا هم اشاره کردم، تمام عملگرهای مورد بحث اولیه نیستند و بسیاری از آنها می‌توانند از عملگرهای دیگر بدست‌آیند. یک (و نه تنها) حالت ممکن عبارت است از: گزینش، پرتو، پیوند، اجتماع و نیم تفاضل. توجه: ممکن است تعجب کنید که چطور تغییر نام در این فهرست وجود ندارد. در واقع تغییر نام هم اولیه نیست ولی من هنوز پیش زمینه تشریح دلیل آنرا برای شما آماده نساخته‌ام. این نمونه به ما نشان می‌دهد که بین اولیه‌بودن و مفید بودن تفاوت هست! مسلما من نمی‌توانم بدون عملگر تغییر نام خودمان زندگی کنم، حتی اگر بدانم این عملگر اولیه نیست.

ارزیابی عبارات SQL

کاد بجز پیوند طبیعی، عملگر دیگری بنام پیوند تتا را تعریف کرد که در آن بجای تساوی می‌توان از هر یک از عملگرهای مقایسه‌ای (مانند «=»، «≠»، «<» و غیره) استفاده کرد. این عملگر

* تا وقتی که S خالی نباشد. اگر باشد چه می‌شود؟

اولیه نیست و در حقیقت معادل گزینش از حاصل‌ضرب دکارتی است. در اینجا مثالی در توتوریال‌دی را می‌بینید که در آن پیوند «نامساوی» توزیع‌کنندگان و قطعات بر روی شهر برقرار شده است (پس تتا در اینجا «≠» است).

```
( ( S RENAME ( CITY AS SCITY )
  TIMES
  ( P RENAME ( CITY AS PCITY ) )
WHERE SCITY ≠ PCITY
```

توجه کنید که نتیجه دارای دو «شهر» است. PCITY و SCITY. و اما در SQL:

```
SELECT S.SNO, S.SNAME, S.STATUS, S.CITY AS SCITY,
    P.PNO, P.PNAME, P.COLOR, P.WEIGHT, P.CITY AS PCITY
FROM   S, P
WHERE  S.CITY <> P.CITY
```

می‌توانید تصور کنید که این عبارت SQL در سه مرحله بصورت زیر اجرا می‌شود:

1. بند FROM اجرا شده و حاصل‌ضرب جدول‌های S و P را تولید می‌کند. (اگر بخواهیم رابطه‌ای کار کنیم بایستی قبل از ضرب، ویژگی شهر را تغییر نام دهیم. SQL از زیر این کار شانه خالی می‌کند چرا که در هر جدول ستون‌ها از چپ به راست ترتیب دارند و دو ستون بنام CITY با توجه به محل استقرارشان از همدیگر باز شناخته می‌شوند. برای سادگی از جزئیات می‌گذرم.)
2. بعد، بند WHERE به اجرا در می‌آید و گزینشی از حاصل ضرب را برمی‌گرداند به گونه‌ای که سطرهایی که دو مقدار شهر در آنها مساویند، حذف می‌شوند.*
3. در نهایت، بند SELECT اجرا شده و پرتوی از حاصل گزینش، ستون‌هایی خاص از آن را نمایش می‌دهد. (البته SELECT عمل تغییر نام را هم انجام می‌دهد همانطور که در این مثال مشاهده می‌شود و در آینده خواهیم دید عملی همانندگسترش رابطه‌ای را هم انجام می‌دهد. دیگر اینکه SELECT تکرارها

* اگر تتا بجای =، ≠ می‌بود، مرحله 2 به این صورت در می‌آمد: گزینش حاصل‌ضرب دکارتی به گونه‌ای که سطرهایی که شهر آنها برابر است باقی بمانند. این حالت پیوند مساوی توزیع‌کنندگان و قطعات بر روی شهر خوانده می‌شود. به عبارت دیگر پیوند مساوی، پیوند تتایی است که در آن تتا برابر «=» باشد. تمرین: پیوند مساوی و پیوند طبیعی چه تفاوتی دارند؟

را برطرف نمی‌کند مگر اینکه با DISTINCT بر این امر تصریح شده باشد.
ولی من برای سادگی از این جزئیات می‌گذرم.)

حداقل در اولین رویارویی چنین به نظر می‌رسد که FROM با ضرب دکارتی، WHERE با گزینش و SELECT با پرتو در ارتباط‌اند و کل عبارت نشان‌دهنده پرتوی از گزینش یک ضرب دکارتی است. این فقط یک شرح تقریبی – ولی مستدل و رسمی- برای معنای سمنتیک SELECT_FROM_WHERE در SQL است. به همین صورت یک الگوریتم ادراکی برای ارزیابی این عبارت ارائه می‌دهم. البته هیچ پیاده‌سازی دقیقا این الگوریتم را برای اجرای چنین عبارتی بکار نمی‌بندد و هر الگوریتمی که اجرای آن بدست آمدن چنین نتیجه‌ای را تضمین کند، می‌تواند مورد استفاده قرار گیرد. غالبا دلیل خوبی – معمولا در جهت افزایش کارایی- برای انتخاب یک الگوریتم وجود دارد، بنابراین (مثلا) اجرای بندها با ترتیبی متفاوت و بازنویسی کوئری به شکلی دیگر می‌تواند مورد توجه قرار گیرد. در هرحال پیاده‌سازی آزاد است تا هر کاری که صلاح می‌داند را انجام دهد، البته *فقط تا زمانی که بتوان ثابت کرد الگوریتم مورد استفاده از نظر منطقی معادل مفهوم مورد نظر است*. وظیفه بهینه‌ساز را می‌توان چنین دانست: پیداکردن الگوریتم‌هایی که به صورت تضمینی معادل مفهوم مورد نظر هستند ولی کارایی بهتری دارند.

گسترش و خلاصه‌سازی

همانطور که احتمالا متوجه شده‌اید، جبر رابطه‌ای که توضیح داده شد تا حدود زیادی فاقد امکانات محاسباتی است در حالی که SQL چنین نیست. می‌توانیم کوئری‌هایی به این شکل در SQL داشته باشیم:

```
SELECT A+B AS ...
SELECT SUM(C) AS ...
```

همانطور که بزودی خواهید دید، SUM و یا «+» از محدوده جبر رابطه‌ای (اصلی) خارج است. ولی ما به افزودن عملگرهای جدید به جبر رابطه‌ای -به منظور افزایش قدرت آن- نیاز داریم. این علت بوجود آمدن خلاصه‌سازی است. به بیان تقریبا صحیح می‌توان گفت: گسترش امکان محاسبه «در سطح تاپل» و خلاصه‌کردن امکان محاسبه «ویژگی‌های زیر هم» را فراهم می‌کند. بیایید نگاهی نزدیکتر داشته باشیم.

گسترش‌دادن

چون این عملگر برایتان تازگی دارد، با یک مثال شروع می‌کنم. فرض کنید وزن قطعه‌ها به پوند داده شده است و ما به وزن آنها بر حسب گرم نیاز داریم. هر 454 گرم یک پوند است، پس چنین می‌نویسیم:

```
EXTEND P ADD                          | SELECT P.*,
( WEIGHT * 454 AS GMWT )              |   ( P.WEIGHT * 454 ) AS
GMWT                                  | FROM   P
```

با مقدارهای مثال خودمان چنین نتیجه‌ای حاصل می‌شود:

PNO	PNAME	COLOR	WEIGHT	CITY	GMWT
P1	Nut	Red	12.0	London	5448.0
P2	Bolt	Green	17.0	Paris	7718.0
P3	Screw	Blue	17.0	Oslo	7718.0
P4	Screw	Red	14.0	London	6356.0
P5	Cam	Blue	12.0	Paris	5448.0
P6	Cog	Red	19.0	London	8626.0

توجه

مهم: مرابطه P در پایگاه تغییری *نمی‌کند*! گسترش را با ALTER TABLE در SQL اشتباه نگیرید؛ عبارت گسترش، فقط یک عبارت است و مثل هر عبارت دیگر نماینده یک مقدار است.

با مثال ادامه می‌دهیم، حال این کوئری را در نظر بگیرید «شماره قطعه و وزن بر حسب گرم قطعاتی را بده که از 7000 گرم سنگین‌تر باشند.» عبارت توتوریال‌دی چنین است:

```
( ( EXTEND P
    ADD ( WEIGHT * 454 AS GMWT ) )
     WHERE GMWT > 7000.0 ) { PNO, GMWT }
```

و همچنین SQL:

```
SELECT P.*, ( P.WEIGHT * 454 ) AS GMWT
FROM   P
WHERE  ( P.WEIGHT * 454 ) > 7000.0
```

همانطور که می‌بینید عبارت P.WEIGHT * 454 دو بار در SQL ظاهر شده است و باید امیدوار باشیم که پیاده‌سازی آن‌قدر هوشمند باشد که آن‌را برای هر تاپل دوبار محاسبه نکند. در توتوریال‌دی این عبارت فقط یک بار آمده است.

مشکلی که در این مثال خود را نشان می‌دهد این است که ساختار SELECT-FROM-WHERE در SQL بیش از حد انعطاف ناپذیر است. چه باید کرد تا این عبارت‌ها مانند توتوریال‌دی ساده شوند؟ در عبارات SQL بند WHERE بایستی نتیجه حاصل از بند SELECT را محدود کند(و می‌کند)، ولی مشکل از آنجا ناشی می‌شود که در قالب SELECT-FROM-WHERE بخش WHERE به حاصل بند FROM نگاه می‌کند، نه بند SELECT. نکته اساسی جبر رابطه‌ای این است که عبارت‌ها می‌توانند به شکل دلخواه با هم ترکیب شوند (با استفاده از خاصیت بسته بودن) ولی قالب SELECT-FROM-WHERE در SQL عملا به این معناست که کوئری‌ها *باید* به شکل ضرب دکارتی نوشته شوند.* بر اثر این محدودیت بایستی ترکیب، تغییر نام و گسترش عبارت به شکل خاصی انجام شود که بسیاری از کوئری‌ها نمی‌توانند از این الگو پیروی کنند.

ممکن است بگویید چرا عبارت SQL را به این صورت ننوشتم:

```
SELECT P.*, ( P.WEIGHT * 454 ) AS GMWT
FROM   P
WHERE  GMWT > 7000.0
```

(خط آخر تغییر کرده است.) به این دلیل که GMWT نام یک ستون در *نتیجه نهایی* است، جدول P چنین ستونی ندارد، بند WHERE سرگردان می‌شود و اجرا عبارت با خطا روبرو می‌شود.

در SQL استاندارد می‌توان این کوئری را به گونه‌ای نوشت که اندکی به توتوریال‌دی شبیه است:

```
SELECT TEMP.PNO, TEMP.GMWT
FROM ( SELECT P.PNO, ( P.WEIGHT * 454 ) AS GMWT
     FROM P ) AS TEMP
WHERE  TEMP.GMWT > 7000.0
```

* این درست است که عملوندهای این ضرب دکارتی محدود به جدول‌های نامدار نیستند(جدول‌های پایه یا دیدها) که این ویژگی مدیون خیلی چیزها از جمله قابلیت زیر کوئری‌های تو در تو در بند FROM است، ولی با این وجود بند FROM هنوز هم به معنای ضرب دکارتی است.

همانطور که قبلا هم متوجه شدید، تمام محصولات SQL از زیر کوئری‌های تو در تو – به این شکل–پشتیبانی نمی‌کنند.

یک تعریف. فرض کنید r یک رابطه باشد، آنگاه *گسترش*:

EXTEND r ADD (*exp* AS X)

رابطه‌ای است با الف) عنوانی مشابه عنوان r که با ویژگی X گسترش داده شده است و ب) بدنه‌ای محتوی تمام تاپل‌هایی مانند t به گونه‌ای که t تاپلی از r باشد که بوسیله مقدار X گسترش یافته‌اند و X مقداری است که با استفاده از عبارت *exp* بر روی تاپل r محاسبه می‌شود. رابطه r نبایستی ویژگی‌ای بنام X داشته باشد و *exp* نبایستی به X ارجاع داشته باشد. توجه داشته باشید که کاردینالیتی نتیجه مساوی کاردینالیتی r و درجه آن مساوی درجه r بعلاوه یک است و نوع X با نوع *exp* یکی است.

خلاصه کردن

باز هم با مثال شروع می‌کنم (کوئری چنین است «برای هر یک از توزیع‌کنندگان، شماره توزیع‌کننده و تعداد قطعاتی که توزیع می‌کند را بده»):

SUMMARIZE SP PER (S { SNO }) ADD (COUNT ( ) AS P_COUNT)

با مقدارهای مثال همیشگی‌مان چنین نتیجه‌ای بدست می‌آید:

SNO	P_COUNT
S1	6
S2	2
S3	1
S4	3
S5	0

توجه داشته باشید که –خصوصا در مورد توزیع‌کننده S5– کلمه PER نشانه این است که خلاصه کردن از «پرتو S بر روی SNO» حاصل می‌شود. یعنی نتیجه دارای پنج تاپل است (همه آنها در پرتو هستند). در مقابل این عبارت SQL (که احتمالا فکر می‌کنید معادل عبارت توتوریال‌دی است):

```
SELECT SP.SNO, COUNT(*) AS P_COUNT
  FROM   SP
  GROUP  BY SP.SNO
```

حاصل فقط شامل توزیع‌کنندگان S1، S2، S3 و S4 خواهد بود. دلیل این امر آن است که نتیجه فقط از SP استخراج می‌شود و S5 اساسا در SP وجود ندارد. عبارت SQL که واقعا معادل عبارت توتوریال‌دی است:

```
SELECT S.SNO, TEMP.P_COUNT
  FROM   S, LATERAL ( SELECT COUNT(*) AS P_COUNT
         FROM   SP
         WHERE  SP.SNO = S.SNO ) AS TEMP
```

همانطور که گفتم تمام محصولات SQL از این نوع عبارات پشتیبانی نمی‌کنند.

توجه

استاندارد به کلمه کلیدی LATERAL نیاز دارد چرا که زیرکوئری بصورت «جنبی laterally» به عنصر موجود در FROM اشاره می‌کند.

و حالا تعریف: فرض کنید r و s دو رابطه هستند به گونه‌ای که s با پرتوی از r هم‌نوع است و فرض کنید ویژگی‌های s عبارتند از :$A_1, A_2,...A_n$ آنگاه *خلاصه*:

SUMMARIZE r PER (s) ADD (*summary* AS X)

رابطه‌ای است با الف) عنوانی مانند s که با X گسترش داده شده است و ب) بدنه‌ای محتوی تمام تاپل‌هایی مانند t که t تاپلی است از s که با مقدار X گسترش یافته است. مقدار X با خلاصه کردن تمام تاپل‌هایی از r که در $A_1, A_2,...A_n$ دارای یک مقدار باشند محاسبه می‌شود. رابطه s نبایستی دارای ویژگی به نام X باشد و خلاصه نبایستی به X مراجعه نماید. توجه داشته باشید که کاردینالیتی نتیجه مساوی کاردینالیتی s و درجه آن مساوی درجه s بعلاوه یک است و نوع X با نوع *خلاصه* یکی است.

چه «خلاصه‌هایی» انجام پذیرند؟ این فهرست باز (قابل افزایش) است ولی مسلما توابع خلاصه‌سازی معمول مانند COUNT، SUM، AVG، MAX و MIN در آن وجود دارند. یک مثال برای پیدا کردن بزرگترین و کوچکترین:

```
SUMMARIZE SP PER ( SP { SNO } ) ADD ( MAX ( QTY ) AS MAXQ ,
MIN ( QTY ) AS MINQ )
```

این مثال دو نکته را آشکار می‌کند:

- می‌توان «چند خلاصه‌سازی» را با یک عبارت انجام داد. (قبلا این را نگفته بودم ولی در مورد RENAME و EXTEND هم صادق است.)
- آنچه در اینجا مقابل PER آمده «هم‌نوع با» پرتوی از رابطه که خلاصه شده است، نیست. بلکه در عمل فقط نشان دهنده یک پرتو است و در چنین مواردی می‌تواند به آسانی ساده شود:

```
SUMMARIZE SP BY { SNO } ADD ( MAX ( QTY ) AS MAXQ ,
MIN ( QTY ) AS MINQ )
```

خلاصه‌کننده‌های مجاز دیگر عبارتند از: CONNTD، SUMD و AVGD (که D در اینجا نماینده متمایز distinct است و به معنای «حذف مقدارهای تکراری قبل از خلاصه‌کردن» است) AND، OR، XOR (برای ویژگی‌های بولی)، INTERSECT، UNION و UNION_D (برای ویژگی‌هایی که خود رابطه‌اند) و غیره.

به هر حال COUNT و وابستگان به آن جزء عملگرهای جمعی به حساب *نمی‌آیند*. با این وجود اکثر آنها هم‌نام توابع جمعی‌اند (در SQL مخلوط کردن این دو موضوع نتایج بدی به بار آورده است). برای اجرای یک عملگر جمعی از یک عبارت اسکالر استفاده می‌شود و یک نتیجه اسکالر بازمی‌گرداند.* مثلا:

```
VAR N INTEGER
N := COUNT ( S WHERE CITY = 'London' ) ;
```

در مقابل خلاصه کننده‌ها فقط عملوندهایی برای اجرای خلاصه‌سازی هستند و خارج از این محدوده معنایی ندارند و نبایستی مورد استفاده قرار گیرند.

* توابع جمعی غیراسکالر هم می‌توانند وجود داشته باشند ولی این موضوع از دامنه بحث این کتاب خارج است.

یک مثال دیگر در رابطه با خلاصه‌کردن (چند توزیع‌کننده در لندن وجود دارد؟):

```
SUMMARIZE ( S WHERE CITY = 'London' ) ADD ( COUNT ( ) AS N )
```

این مثال هم دو نکته را روشن می‌کند:

اول، خلاصه کردن بصورت پیش‌فرض دارای ویژگی PER نیست و خلاصه سازی *per* *TABLE_DEE* انجام می‌شود. بطور خلاصه:

```
SUMMARIZE ( S WHERE CITY = 'London' )
    PER ( TABLE_DEE ) ADD ( COUNT ( ) AS N )
```

یادآوری می‌کنم DEE رابطه‌ای است بدون ویژگی و با یک تاپل (و در نتیجه با پرتوی از تمام رابطه‌ها هم‌نوع است! البته پرتو مورد بحث هیچ ویژگی ندارد). خروجی این خلاصه‌سازی دارای یک ویژگی و یک تاپل است و با مقدارهای همیشگی مثال خودمان چنین نتیجه می‌دهد:*

N
2

دوم، چند لحظه پیش گفتم که توابع جمعی و عملگرهای خلاصه‌کننده دو چیز متفاوت هستند و ممکن است متعجب شده باشید که تفاوت مثال مورد بحث و عملگر COUNT چند پاراگراف قبل در چیست؟

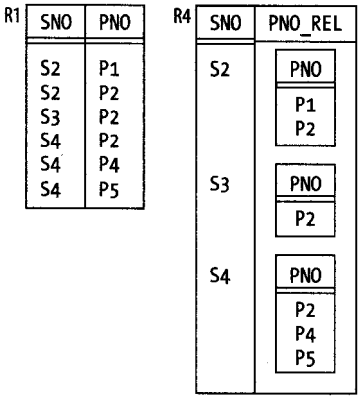
```
COUNT ( S WHERE CITY = 'London' )
```

تفاوت اصلی این است که خلاصه‌سازی یک رابطه برمی‌گرداند و عملگر (تابع) جمعی یک مقدار اسکالر. برای بحث بیشتر به تمرین 11 مراجعه کنید.

گروه‌بندی و از گروه درآوردن

یادآوری از فصل دوم، رابطه‌ای که ویژگی‌های آن خودشان هم رابطه باشند، قابل قبول است (RVA). شکل 5-1 رابطه‌های R1 و R4 را به تصویر درمی‌آورد که از شکل 2-1 و شکل 2-2 گرفته شده‌اند. R4 یک RVA است و R1 چنین نیست ولی هر دو رابطه یک مطلب را می‌رسانند.

* اینکه در این شکل هیچ ستونی دوخطه نیست، مشکلی بوجود نمی‌آورد. تمرین 28 از فصل چهارم را ببینید.



شکل 5–1. دو رابطه R1 و R4 برگرفته از شکل‌های 2–1 و 2–2

البته ما به راهی برای حرکت بین استفاده و عدم استفاده از RVA نیاز داریم و هدف از گروه‌بندی همین است. من قصد ندارم در اینجا وارد جزئیات این عملگرها شوم و فقط می‌خواهم این را در مورد رابطه‌های R1 و R4 همانطور که در شکل 5–1 نشان داده شده‌اند، بگویم. R4 با این عبارت ساخته می‌شود:

```
R1 GROUP ( { PNO } AS PNO_REL )
```

و R1 با این عبارت:

```
R4 UNGROUP ( PNO_REL )
```

و SQL هیچ همانندی برای این عملگرها ندارد.

اگر R4 برای توزیع‌کننده S_x دقیقا یک تاپل داشته باشد، و مقدار PNO_REL در آن تاپل خالی باشد، آنگاه نتیجه UNGROUP فوق دارای شماره توزیع‌کننده S_x نخواهد بود.* برای اطلاعات بیشتر به کتاب *مقدمه‌ای بر پایگاه داده‌ها* ویرایش هشتم 2004 و یا *پایگاه داده‌ها، نوع‌ها و مدل رابطه‌ای سومین بیانیه* 2006 نوشته هیو داروین و من، مراجعه نمایید.

ممکن است دیدن ظاهر عملگرهایی که با RVA سر و کار دارند، تعجب کنید. در هر رابطه زمانی که یک عملگر به یک ویژگی مانند A که از نوع T است مراجعه می‌کند، «زیرعملگرهایی» را برای کار با مقدار A به کار می‌گیرد که از نوع T هستند. پس اگر T «نوع

* این اشکالی است که بر اثر حذف تهی از مدل رابطه‌ای بوجود آمده است. مترجم

رابطه» باشد، زیر عملگرهای مورد بحث نیز بایستی برای رابطه‌ها تعریف شده باشند و چنین «زیرعملگرهایی» عملگرهایی ذاتاً رابطه‌ای هستند (مانند پیوند و...)! یعنی همان عملگرهایی که در این فصل آنها را شناختیم. تمرین‌های 27 و 30 را ببینید.

تبدیل عبارت‌ها

تا اینجا در مورد تمام عملگرهای جبری کم و بیش با جزئیات صحبت کرده‌ام ولی موضوعات دیگری نیز هستند که جا دارد در این فصل ذکری از آنها شود. یکی از این موضوعات تبدیل یک عبارت رابطه‌ای به یک عبارت دیگر است به گونه‌ای که هر دو عبارت معادل یکدیگر باشند. من امکان انجام چنین کاری را برای نخستین بار در فصل 3 و قسمت «چرا تکرار غیر مجاز است»، مطرح کردم و شرح دادم که تبدیل یکی از کارهایی است که بهینه‌ساز انجام می‌دهد. در حقیقت انجام این تبدیل‌ها یکی از دو کاری است که ایده بهینه‌سازی رابطه‌ای بر پایه آنها شکل گرفته است (دیگری که از محدوده این کتاب خارج است استفاده از آمار در بهبود کارایی پایگاه داده‌هاست که *بهینه‌سازی بر مبنای هزینه* نامیده می‌شود). در این قسمت قصد دارم مطالبی دیگر در مورد عملیات تبدیل عبارت (و یا بازنویسی کوئری) بگویم. با یک مثال ساده شروع می‌کنم. عبارت جبری معادل این کوئری را در نظر بگیرید «مشخصات توزیع‌کنندگانی که P2 را توزیع می‌کنند به همراه تعداد توزیعی هر یک را بده»:

```
( ( S JOIN SP ) WHERE PNO = PNO('P2') ) { ALL BUT PNO }
```

تصور کنید که 100 توزیع‌کننده و 1،000،000 سفارش موجودند که 500 سفارش مربوط به P2 است. اگر ارزیابی این عبارت به همین صورت که نوشته شده و بدون بهینه‌سازی هوشمندانه صورت گیرد، چنین اتفاقاتی پیش می‌آید:

1. *پیوند S و SP*: در این مرحله بایستی 100 تاپل توزیع‌کنندگان خوانده شوند و هر یک از 1،000،000 تاپل سفارش‌ها بایستی برای هر توزیع‌کننده یک مرتبه –و در مجموع هر تاپل 100 مرتبه– خوانده شود. سپس جهت ایجاد نتیجه میانی بایستی 1،000،000 تاپل در دیسک نوشته شوند (برای سادگی فرض کرده‌ام تمامی تاپل‌ها بصورت فیزیکی ذخیره سازی می‌شوند، همچنین «تعداد دفعات ذخیره و بازیابی تاپل» را معیار سنجش کارایی گرفته‌ام. این

فرضیات واقعی نیستند ولی این امر به روش استدلال من لطمه‌ای وارد نمی‌کند.)

2. *گزینش حاصل مرحله 1:* در این مرحله تمامی تاپل‌های حاصل از مرحله قبل یعنی 1،000،000 تاپل خوانده می‌شوند و از میان آنها فقط 500 تاپل انتخاب می‌شود که فرض می‌کنم در حافظه قابل نگهداری هستند (ولی نه یک میلیون تاپل!).
3. *پرتو حاصل مرحله 2:* در این مرحله هیچ تاپلی ذخیره و بازیابی نمی‌شود، پس می‌توان از این مرحله صرفنظر کرد.

روند زیر از نظر نتیجه نهایی با روش قبل تفاوتی ندارد ولی بطور واضح کارایی آن بهتر است:

1. *گزینش تاپل‌هایی که مربوط به P2 هستند از رابطه SP:* در این مرحله 1،000،000 تاپل خوانده می‌شوند و از میان آنها فقط 500 تاپل انتخاب می‌شود و در حافظه باقی می‌مانند.
2. *پیوند S و نتیجه مرحله1:* در این مرحله فقط بایستی 100 تاپل مربوط به توزیع کنندگان خوانده شوند (فقط یک بار نه یک‌بار برای هر سفارش، چرا که تمام سفارش‌های P2 در حافظه وجود دارند). حاصل شامل 500 تاپل است (که باز هم در حافظه باقی می‌ماند).
3. *پرتو حاصل مرحله 2:* باز هم می‌توان از این مرحله صرفنظر کرد.

راه اول دارای 102,000,100 و راه دوم فقط 1,000,100 ذخیره و بازیابی دارد. بنابراین واضح است که دومی بیش از صد برابر سریعتر از اولی است و همچنین واضح است که ما دوست داریم پیاده‌سازی راه دوم را بجای راه اول مورد استفاده قرار دهد! برای این کار بایستی عبارت اول:

```
( S JOIN SP ) WHERE PNO = PNO('P2')
```

تبدیل به عبارت دوم شود (پرتو را حذف کردم چرا که در استدلال ما تغییری ایجاد نمی‌کند):

```
S JOIN ( SP WHERE PNO = PNO('P2') )
```

این دو عبارت از نظر منطقی با هم مساوی‌اند ولی همانطور که دیدید کارایی آنها از زمین تا آسمان با هم فرق می‌کند. اگر عبارت اول به سیستم داده شود، توقع داریم که قبل از اجرا آن را به عبارت دوم تبدیل کند و البته سیستم هم چنین کاری را انجام می‌دهد. نکته اینجاست که جبر رابطه‌ای – که یک روش عبارت‌نویسی سطح بالاست– می‌تواند شامل قواعدی به نام *قوانین تبدیل* شود. مثلا قانونی هست که می‌گوید پیوندی که گزینش می‌شود می‌تواند به گزینشی که پیوند می‌شود، تبدیل گردد (من از همین قانون در مثال استفاده کردم). *یک بهینه‌ساز خوب این قوانین را بلد است* و آنها را به کار می‌بندد. چرا که کارایی یک کوئری نبایستی در به نحوه عبارت‌نویسی آن کوئری وابسته باشد. (این نتیجه اولیه نبودن همه عملگرهای جبری است که عبارت‌ها می‌توانند به عبارت‌های دیگری تبدیل شوند. مثلا به جای استفاده از اشتراک در نوشتن عبارت‌ها، می‌توان از پیوند استفاده کرد.)

قوانین تبدیل بسیار زیادند و اینجا جای بحث جامع در مورد آنها نیست و فقط قصد دارم بر نکات کلیدی و مهم تکیه کنم. اول، قانونی که در پاراگراف قبل شرح داده شد حالت خاصی است از یک قانون عمومی‌تر به‌نام *قانون توزیع‌پذیری*. بطور کلی زمانی که عملگر یکتایی مانند f بر عملگر دوتایی g توزیع می‌شود هر چه باشد a و b:

$$f(g(a,b)) = g(f(a),f(b))$$

مثلا در ریاضیات جذر SQRT بر روی ضرب توزیع می‌شود:

$$\mathrm{SQRT}(a * b) = \mathrm{SQRT}(a) * \mathrm{SQRT}(b)$$

برای هر چه باشد a و b (f را SQRT و g را * فرض کنید)؛ بنابراین یک بهینه‌ساز عبارت ریاضی می‌تواند همیشه در زمان تبدیل عبارت هر کدام از این دو عبارت را جانشین دیگری نماید. و به عنوان یک مثال مخالف این قاعده، جذر بر روی جمع توزیع نمی‌شود، چرا که جذر $a+b$ مجموع جذر a و b نیست.

در جبر رابطه‌ای گزینش بر روی اشتراک، اجتماع و تفاضل توزیع می‌شود. در مورد توزیع گزینش بر پیوند، این کار فقط در صورتی ممکن است که عبارت شرطی مرکب باشد و از دو قسمت مجزا که با AND از هم جدا شده‌اند، تشکیل شده باشد و هر یک از آنها مربوط به یکی از عملوندهای پیوند باشند. در مورد مثال قبل، این شرط برقرار است (شرط گزینش بسیار ساده و فقط به یکی از عملوندها مربوط است) و با استفاده از توزیع‌پذیری می‌توان عبارت را به منظور افزایش کارایی تعویض نمود. اصل این است که می‌توان «گزینش را زودتر انجام داد». زود

انجام دادن گزینش همیشه کار خوبی است چرا که تعداد تاپل‌هایی که بایستی در مرحله بعد بررسی شوند را کاهش می‌دهد و به احتمال زیاد تعداد تاپل‌هایی که در مرحله بعد تولید می‌شوند را نیز، کمتر می‌کند.

یک ویژگی دیگر از توزیع‌پذیری و این بار در ارتباط با پرتو. اولا، پرتو روی اجتماع توزیع می‌شود ولی تفاضل و اشتراک چنین خاصیتی ندارند. ثانیا، پرتو فقط در صورتی بر پیوند توزیع می‌شود که همه ویژگی‌هایی که پیوند بر آنها انجام می‌شود، در پرتو موجود باشند. با استفاده از این قوانین می‌توان «پرتو را زودتر انجام داد»، که این کار هم به دلیلی که قبلا گفته شد کار خوبی است.

دو قانون مهم دیگر عبارتند از *جابجایی‌پذیری* و *شرکت‌پذیری*:

یک عملگر دوتایی جابجایی‌پذیر است اگر هر چه باشد a و b:

$$g(a,b) = g(b,a)$$

در ریاضیات جمع و ضرب جابجایی‌پذیرند ولی تفریق و تقسیم این ویژگی را ندارند. در جبر رابطه‌ای هم اشتراک، اجتماع و پیوند جابجایی‌پذیر هستند* ولی تفاضل و تقسیم جابجایی‌پذیر نیستند. پس مثلا اگر در یک کوئری پیوند r و s را داشته باشیم، تفاوتی ندارد که کدامیک اول در دیگری پیوند می‌شود و سیستم مجاز است همیشه رابطه کوچکتر را به رابطه بزرگتر پیوند دهد.

یک عملگر دوتایی شرکت‌پذیر است اگر هر چه باشد a و b:

$$g(a,g(b,c)) = g(g(a,b),c)$$

در ریاضیات جمع و ضرب شرکت پذیرند ولی تفریق و تقسیم این ویژگی را ندارند. در جبر رابطه‌ای اشتراک، اجتماع و پیوند شرکت‌پذیرند و تفاضل و تقسیم شرکت‌پذیر نیستند. پس اگر در یک کوئری پیوند سه رابطه r، s، و u را داشته باشیم، قوانین شرکت‌پذیری و جابجایی‌پذیری به ما می‌گویند که می‌توانیم آنها را به هر ترتیبی در هم ضرب کنیم (همچنین می‌گویند که می‌توانیم یک عملگر nتایی داشته باشیم. همان کاری که توتوریال‌دی کرده است.) و سیستم است که تشخیص می‌دهد که از میان حالات مختلف، کدامیک کارایی بیشتری دارد.

* به بیان دقیق، پیوند در SQL جابجایی پذیر نیست. در SQL ستون‌ها ترتیب دارند و ترتیب ستونها در نتیجه r JOIN s و s JOIN r *با هم تفاوت دارند.*

توجه داشته باشید که تمام این تبدیل‌ها بدون توجه به مقدار واقعی داده‌ها و ساختار واقعی ذخیره و بازیابی در رسانه فیزیکی (مثل شاخص‌ها)، انجام‌پذیرند. به بیان دیگر چنین تبدیل‌هایی به صورت مجازی و بدون توجه به شکل فیزیکی پایگاه، بهبود کارایی را تضمین می‌کنند.

مقایسه‌گرهای رابطه‌ای

در فصل 2 در مورد این واقعیت صحبت کردم که عملگر مقایسه‌ای «=» بر روی تمامی نوع‌ها قابل استفاده است و این قاعده طبیعتاً شامل نوع *رابطه* نیز می‌شود. دو رابطه به نام‌های r و s و هر دو از نوع T داریم و بایستی بتوانیم بفهمیم که آنها با هم برابرند یا نه. مقایسه‌گرهای دیگر نیز مفیدند. مثلا ممکن است بخواهیم بدانیم که آیا r زیر مجموعه s است (یعنی هر تاپل r در s هم هست)؟ و یا آیا r زیر مجموعه محض s است (یعنی هر تاپل r در s هم هست ولی s حداقل یک تاپل دارد که در r نیست)؟

در اینجا بایستی تاکید کنم که چنین عملگرهایی رابطه‌ای نیستند –و طبعا به جبر رابطه‌ای هم تعلق ندارند– چرا که مقدار بازگشتی آنها صحیح و غلط است، نه رابطه. با این وجود ترجیح می‌دهم این بحث را در همین فصل جای دهم. یک مثال:

```
S { CITY } = P { CITY }
```

طرف سمت چپ پرتو توزیع‌کنندگان بر روی شهر است و سمت راست پرتو قطعات بر روی شهر، و اگر این دو با هم مساوی باشند، مقایسه صحیح و در غیر این صورت غلط را بر می‌گرداند. به بیان دیگر این مقایسه (که یک عبارت بولی است) بدین معناست که: «آیا مجموعه شهرهای توزیع‌کنندگان و مجموعه شهرهای قطعات مساویست؟»

یک مثال دیگر:

```
S { SNO } ⊃ SP { SNO }
```

توضیح: علامت «⊃» در اینجا به معنای «زیر مجموعه محض» است. معنای این مقایسه‌گر (برداشت آزاد) این است که : «آیا توزیع‌کننده‌ای هست که هیچ قطعه‌ای را توزیع نکند؟» مقایسه‌گرهای رابطه‌ای دیگر عبارتند از «⊇»، «⊆» (زیرمجموعه) و «⊂» (زیر مجموعه محض).

بدیهی است مقایسه‌گرهای رابطه‌ای برای انجام گزینش کاربرد فراوان دارند.* بیایید چند مثال را بررسی کنیم. اولین کوئری را در نظر بگیرید «توزیع‌کنندگانی را بده که همه قطعات را توزیع می‌کنند.» یک عبارت ممکن چنین است:

```
WITH ( SP RENAME ( SNO AS X ) ) AS R :
    S WHERE ( R WHERE X = SNO ) { PNO } = P { PNO }
```

توضیح: عبارت WITH ... AS R عملا مانند انتسابی است که مقدار عبارت

```
SP RENAME (SNO AS X)
```

را به یک مرابطه موقت و ساخته شده بوسیله سیستم به نام R منتسب می‌کند. پس از این انتساب مقدار رابطه R مشابه مرابطه SP خواهد بود با این تفاوت که ویژگی SNO به X تغییر نام یافته است† (هدف از استفاده از اسامی X و R جلوگیری از تداخل نام‌ها است). سپس برای یافتن توزیع‌کننده S_X از مرابطه S، این عبارت:

```
( R WHERE X = SNO ) { PNO }
```

رابطه‌ای با یک ویژگی PNO را برمی‌گرداند که شامل شماره تمامی قطعاتی است که توزیع‌کننده S_X آنها را توزیع می‌کند. توجه داشته باشید که اگر S_X قطعه‌ای را توزیع نکند، این رابطه هیچ تاپلی نخواهد داشت. در نهایت، تساوی این رابطه درجه یک با پرتو PNO بر روی P مورد بررسی قرار می‌گیرد. بدیهی است حاصل این بررسی «صحیح» خواهد بود اگر و تنها اگر شماره قطعاتی که توسط S_X توزیع می‌شوند مساوی شماره قطعاتی باشد که در مرابطه P وجود دارد.

البته در صورت تمایل می‌توانیم این کوئری را بصورت یک عبارت بنویسیم:

```
S WHERE ( ( SP RENAME ( SNO AS X ) ) WHERE X = SNO ) { PNO }
= P { PNO }
```

ولی در بیشتر موارد استفاده از WITH برای معرفی نام زیر عبارت‌ها، نوشتن عبارت‌های پیچیده را ساده می‌کند. یک تعریف: اگر *rx* یک عبارت رابطه‌ای باشد که به مرابطه R ارجاع دارد، آنگاه عبارت

```
WITH ry AS R: rx
```

---

* تمرین 26 را هم ببینید.

† SQL هم از WITH پشتیبانی می‌کند، ولی عملوندها طور دیگری نوشته می‌شوند: (عبارت AS نام WITH)

خود یک عبارت رابطه‌ای است اگر *ry* هم یک عبارت رابطه‌ای باشد. البته *rx* هم می‌تواند عبارتی با WITH باشد. مانند این عبارت:

WITH *ry* AS *R1* : WITH *rz* AS *R2* : *rx*

که می‌تواند به این صورت خلاصه شود:

WITH *ry* AS *R1*, *rz* AS *R2* : *rx*

در آینده مثال‌های زیادی از خلاصه‌سازی خواهید دید.

در هر حال، مثال قبل «توزیع‌کنندگانی که تمام قطعات را توزیع می‌کنند را بده» شباهت زیادی به عملگر رابطه‌ای تقسیم دارد. بطور دقیق عبارت زیر:

SP { SNO, PNO } DIVIDEBY P { PNO }

(که برای مثال در بخش عملگرهای اصلی ارائه شد) معمولا به عنوان عبارتی برای کوئری «شماره توزیع‌کنندگانی که تمام قطعات را توزیع می‌کنند را بده»، شناخته می‌شود در حالی که واقعا چنین نیست. بلکه این عبارت درخور این کوئری است: «شماره توزیع‌کنندگانی را بده که حداقل یک قطعه توزیع می‌کنند و همچنین تمام قطعات را توزیع می‌کنند.» (اگر نمی‌دانید این دو از نظر منطقی چه تفاوتی با هم دارند تمرین 25 را ببینید.) به نظر من مزیت عبارتی که دارای مقایسه‌گر رابطه‌ای است فقط فهم ساده‌ترنیست، بلکه *درست‌تر* نیز هست.

یک مثال دیگر، کوئری مورد نظر چنین است: «زوج شماره توزیع‌کنندگانی مانند Sx و Sy را بده، به گونه‌ای که Sx و Sy دقیقا قطعات مشابهی را توزیع می‌نمایند.» این کوئری بدون مقایسه‌گرهای رابطه‌ای بسیار دشوار است! در اینجا می‌بینید:

```
WITH ( SP RENAME ( SNO AS SX ) ) { SX, PNO } AS R1 ,
  ( SP RENAME ( SNO AS SY ) ) { SY, PNO } AS R2 ,
  R1 { SX } AS R3 ,
  R2 { SY } AS R4 ,
  ( R1 JOIN R4 ) AS R5 ,
  ( R2 JOIN R3 ) AS R6 ,
  ( R1 JOIN R2 ) AS R7 ,
  ( R3 JOIN R4 ) AS R8 ,
  SP { PNO } AS R9 ,
  ( R8 JOIN R9 ) AS R10 ,
  ( R10 MINUS R7 ) AS R11 ,
  ( R6 JOIN R11 ) AS R12 ,
  ( R5 JOIN R11 ) AS R13 ,
  R12 { SX, SY } AS R14 ,
  R13 { SX, SY } AS R15 ,
```

```
  ( R14 UNION R15 ) AS R16 ,
  R7 { SX, SY } AS R17 :
R17 MINUS R16
```

ولی با استفاده از مقایسه‌گرها بسیار راحت انجام می‌شود:

```
WITH ( S RENAME ( SNO AS SX ) ) { SX } AS RX ,
  ( S RENAME ( SNO AS SY ) ) { SY } AS RY :
( RX JOIN RY ) WHERE ( SP WHERE SNO = SX ) { PNO } =
      ( SP WHERE SNO = SY ) { PNO }
```

به عنوان یک حاشیه تذکر می‌دهم، افزودن AND SX < SY به بند WHERE نتیجه را تا حدودی منظم می‌کند. این کار به طور دقیق الف) زوج‌های (S*x*,S*x*) را حذف می‌کند. ب) مطمئن می‌شود که هر دو حالت (S*x*,S*y*) و (S*y*,S*x*) با هم ظاهر نمی‌شوند.

یک مقایسه خاص که در عمل معمولا پیش می‌آید، مقایسه‌گر = بین رابطه *r* و یک رابطه خالی از همان نوع است. به بیان دیگر آزمایشی برای اینکه بدانیم آیا *r* خالی است یا نه. بیایید با خلاصه شده این مقایسه‌گر آشنا شویم:

```
IS_EMPTY ( rx )
```

عبارت فوق، در صورتی که عبارت رابطه‌ای rx خالی باشد صحیح و در غیر این صورت غلط را باز خواهد گرداند. در فصل بعد بر این مفهوم تکیه خواهم کرد.

یک نیاز دیگر بررسی این موضوع است که آیا تاپل *t* در رابطه *r* موجود است:

```
t ∈ rx
```

عملوند سمت چپ در اینجا، اجرای یک انتخابگر رابطه است و رابطه‌ای با فقط یک تاپل به نام *t* را بازمی‌گرداند. این مقایسه‌گر در صورتی که *t* در عبارت رابطه‌ای *rx* وجود داشته باشد صحیح و در غیر این صورت غلط را بازمی‌گرداند. (∈ یک عملگر مجموعه‌ای است. عبارت *t* ∈*r* می‌تواند به صورت *t* در *r* [دیده می‌شود]، خوانده شود.) در واقع همانطور که احتمالا تا به حال فهمیده‌اید، ∈ مشابه IN در SQL است با این تفاوت که عملوند سمت چپ IN معمولا اسکالر است، نه یک سطر که تا حدودی به تاپل نزدیک است. با این حال SQL از هیچ مقایسه‌گر دیگری بجز این یک مورد پشتیبانی نمی‌کند. (البته از NOT_IN هم پشتیبانی می‌کند، توتوریال‌دی هم این پشتیبانی را دارد و در هر دو مورد به معنای ∉ است.)

در مورد انتساب رابطه‌ای بیشتر بدانیم

عملگر انتساب =: از این رو به عملگر مقایسه‌ای = شباهت دارد که در مورد تمام نوع‌ها، بدون استثنا قابل اعمال است. انتساب رابطه‌ای از یک نظر دیگر هم به = و سایر مقایسه‌گرها شبیه است: انتساب جزء مدل رابطه‌ای نیست. چرا؟ چون مقصد بایستی دقیقا یک مرابطه باشد نه یک رابطه (مرابطه‌ها به جبر رابطه‌ای تعلق ندارند و نمی‌توان در جبر رابطه‌ای به‌روزرسانی را تصور کرد، و «متغیر» یعنی چیز «قابل به‌روزرسانی»). با این حال قصد دارم در اینجا قدری در مورد انتساب رابطه‌ای صحبت کنم و به عملگرهای INSERT، DELETE و UPDATE نگاهی نزدیک‌تر داشته باشم. همانطور که می‌دانید این عملگرها فقط خلاصه‌نویسی شده عملگرهای رابطه‌ای هستند ولی من اکنون در موقعیتی هستم که می‌توانم فقط خلاصه نشده آنها را به صورت عملگرهای جبری نشان دهم. این کار را با چند مثال انجام می‌دهم.

اول، انتساب رابطه‌ای در حالت عمومی خود به شکل زیر است:

$R := rx$

در اینجا R یک مرابطه و rx یک عبارت رابطه‌ای هم‌نوع با R است. نتیجه اجرای این عبارت، انتساب مقدار رابطه r که با عبارت rx تعیین شده، به مرابطه R است (و از فصل 2 یادآوری می‌کنم که بعد از انتساب، عبارت بولی $R = r$ صحیح ارزیابی می‌شود؛ این یعنی *اصل انتساب*). یک مثال ساده:

```
S := RELATION { TUPLE { SNO SNO('S1'), SNAME NAME('Smith'),
                        STATUS 20, CITY 'London' } ,
                TUPLE { SNO SNO('S2'), SNAME NAME('Jones'),
                        STATUS 10, CITY 'Paris'  } ,
                TUPLE { SNO SNO('S3'), SNAME NAME('Blake'),
                        STATUS 30, CITY 'Paris'  } ,
                TUPLE { SNO SNO('S4'), SNAME NAME('Clark'),
                        STATUS 20, CITY 'London' } ,
                TUPLE { SNO SNO('S5'), SNAME NAME('Adams'),
                        STATUS 30, CITY 'Athens' } } ;
```

همان گونه که در فصل 3 دیدیم طرف راست، حاصل اجرای یک انتخابگر رابطه –و در واقع یک ثابت رابطه‌ای– است.

حال به سراغ INSERT، DELETE و UPDATE می‌روم. فرض کنید مرابطه PQ به این شکل تعریف شده است:

```
VAR PQ BASE RELATION { PNO PNO, QTY QTY } KEY { PNO } ;
```

و عمل INSERT بر روی آن:

```
INSERT PQ ( SUMMARIZE SP PER ( P { PNO } )
        ADD ( SUM ( QTY ) AS QTY ) ) ;
```

و معادل مختصر نشده همین عمل:

```
PQ := PQ UNION ( SUMMARIZE SP PER ( P { PNO } )
          ADD ( SUM ( QTY ) AS QTY ) ) ;
```

به بیان دیگر کاری که INSERT انجام می‌دهد این است که اجتماع مقدار قبلی PQ – آنرا *pq* می‌نامیم– و رابطه‌ای که حاصل یک «خلاصه کردن» است را جایگزین مقدار *pq* در مرابطه PQ می‌نماید. (در اینجا فرض را بر آن گذاشته‌ام که درج تاپلی که موجود است، ایجاد خطا نخواهد کرد. اگر چنین فرض نکنیم بایستی بجای UNION از D_UNION استفاده کنیم)

حال یک مثال از DELETE:

```
DELETE S WHERE CITY = 'Athens' ;
```

معادل مختصر نشده:

```
S := S WHERE NOT ( CITY = 'Athens' ) ;
```

در نهایت مثالی از UPDATE:

```
UPDATE P WHERE CITY = 'London'
   ( WEIGHT := 2 * WEIGHT , CITY := 'Oslo' ) ;
```

معادل مختصر نشده:

```
P := WITH ( P WHERE CITY = 'London' ) AS R1 ,
       ( EXTEND R1 ADD ( 2 * WEIGHT AS NEW_WEIGHT,
                  'Oslo' AS NEW_CITY ) ) AS R2 ,
R2 { ALL BUT WEIGHT, CITY } AS R3,
R3 RENAME { NEW_WEIGHT AS WEIGHT, NEW_CITY AS CITY }
AS R4,
P MINUS R1 AS R5 :
R5 UNION R4;
```

این یکی به کمی توضیح نیاز دارد. اولا R1 مجموعه‌ای از تاپل‌هاست که بایستی به‌روزرسانی شود. (صحبت تقریبی، فصل 4 را ببینید) ثانیا ما ابتدا تمامی تاپل‌های R1 را با ستون‌های جدید WEIGHT و CITY توسعه می‌دهیم تا به R2 برسیم. سپس مقدارهای قدیمی WEIGHT و CITY دور انداخته می‌شوند (R3) و در نهایت ستون‌های NEW_WEIGHT و NEW_CITY به WEIGHT و CITY تغییر نام می‌یابند(R4). سپس مجموعه‌ای از تاپل‌ها که به‌روزرسانی نشده‌اند شناسایی شده(R5)، اجتماع R4 و R5 به مرابطه P منتسب می‌شود.

عملگر مرتب‌سازی

آخرین موضوعی که در این فصل بیان می‌کنم مرتب سازی است. مرتب‌سازی نیز به جبر رابطه‌ای تعلق ندارد و همانطور که در فصل 1 اشاره کردم اساسا رابطه‌ای نیست چرا که چیزی تولید می‌کند که رابطه نیست (یک رابطه را به عنوان ورودی می‌گیرد ولی چیز دیگری تولید می‌کند که *رشته‌ای از تاپل‌ها* نام دارد). البته من نمی‌گویم که مرتب‌سازی به درد نمی‌خورد، بلکه می‌گویم وجود آن در عبارت‌های رابطه‌ای قانونی و عاقلانه نیست.* طبق تعریف عبارت زیر – گرچه مجاز است– عبارت رابطه‌ای به حساب نمی‌آیند:

```
S MATCHING SP                | SELECT DISTINCT S.*
  ORDER ( ASC SNO )          | FROM   S, SP
                             | WHERE  S.SNO = SP.SNO
                             | ORDER  BY S.SNO ASC
```

با این وجود من به دو دلیل به این عملگر نامتجانس پرداخته‌ام. اول اینکه این عملگر با چینش تاپل‌ها به یک ترتیب خاص، می‌تواند کارکرد موثری داشته باشد. (تمرین 11 فصل 3 را ببینید). دوم اینکه مرتب سازی یک تابع نیست. این در حالی است که تمامی عملگرهای جبر رابطه‌ای – و در واقع تمام عملگرهای فقط‌خواندنی- تابع *هستند* بدین معنا که با یک ورودی مشخص فقط می‌توانند یک خروجی تولید کنند. برای مثال اثر ORDER BY CITY را بر

* خصوصا نمی‌توانند در تعریف دیدها وجود داشته باشند. برخلاف اینکه حداقل یکی از محصولات معروف این کار را مجاز می‌شمارد.

رابطه توزیع‌کنندگان در نظر بگیرید. بدیهی است این عملگر می‌تواند هر یک از چهار نتیجه زیر را ایجاد کند (برای سادگی فقط شماره توزیع‌کنندگان را آورده‌ام):

S5, S1, S4, S2, S3

S5, S4, S1, S2, S3

S5, S1, S4, S3, S2

S5, S4, S1, S3, S2

تذکری در مورد SQL: اکثر عملگرهای SQL تابع نیستند و اگر تاکنون به این موضوع اشاره نشده، به دلیل قصور من بوده است. دلیل این امر همان‌طور که در تمرین‌های فصل 2 به آن اشاره شد این است که در برخی موارد SQL عبارت *v1=v2* حتی اگر که *v1* و *v2* متفاوت باشند را درست ارزیابی می‌کند. مانند دو رشته کاراکتری 'پاریس' و 'پاریس ' (به فضای خالی آخر کلمه‌ها توجه کنید). این دو مقدار بطور واضح متفاوت‌اند ولی SQL گاهی اوقات آنها را مساوی تشخیص می‌دهد. در نتیجه کوئری‌های مشخص، نتایج «غیر قطعی» تولید می‌کنند. بعنوان مثال:

```
SELECT DISTINCT S.CITY FROM S
```

اگر شهر یکی از توزیع‌کنندگان 'پاریس' و شهر توزیع‌کننده دیگر 'پاریس ' باشد، آنگاه نتیجه ممکن است دارای هر دو 'پاریس' و 'پاریس ' و یا یکی از آنها باشد. ولی این که چه اتفاقی می‌افتد ممکن است به درستی معلوم نباشد و امروز یک نتیجه و فردا یک نتیجه دیگر تولید شود، حتی اگر در این مدت پایگاه هیچ تغییری نکرده باشد. در برخی پیاده‌سازی‌ها ممکن است راه حلی برای این مشکل وجود داشته باشد.

خلاصه

این فصل به ناچار طولانی‌ترین فصل کتاب بود و این در حالی است که من به تمامی عملگرهای جبری و همچنین تمامی جزئیات در مورد عملگرهای شرح داده شده نپرداختم، ولی امیدوارم دراین‌باره که جبر رابطه‌ای چه می‌گوید، به شما *دید خوبی داده باشم*. عملگرهایی که در مورد آنها بحث کردم عبارتند از: تغییرنام، گزینش، پرتو (شامل حالت ALL BUT)، پیوند، نیم‌پیوند (تطابق)، اشتراک، اجتماع، اجتماع مجزا (D_UNION)، تفاضل، نیم‌تفاضل (عدم

تطابق)، ضرب، تقسیم، پیوند تتا (شامل پیوند مساوی)، گسترش، خلاصه‌سازی، گروه‌بندی و ازگروه‌درآوردن. همچنین چند عملگر مهم غیر جبری را شرح دادم؛ مقایسه‌گرهای رابطه‌ای، درج، حذف و به‌روزرسانی (و انتساب رابطه‌ای) و مرتب سازی. سایر موضوعاتی که در این فصل پوشش داده شده‌اند عبارتند از:

- *بسته‌بودن*: نیاز به وجود قواعدی را شرح دادم که بر اساس آنها همواره نوع نتیجه برگشتی از هر عبارت رابطه‌ای برای ما قابل پیش بینی است.
- *عملگرهای اولیه*: نشان دادم که بسیاری از عملگرها در حقیقت خلاصه‌نویسی شده ترکیب عملگرهای دیگر هستند. بطور مشخص اشتراک و ضرب حالت خاصی از پیوند و حالت خاصی از نیم تفاضل است.
- *پیوند*: همچنین (با استفاده از شرکت‌پذیر و توزیع‌پذیر بودن پیوند) نشان دادم، می‌توان نوع پیشوندی و nتایی پیوند را تعریف کرد (که n یک عدد صحیح بزرگتر یا مساوی صفر است). حاصل ضرب صفر رابطه در هم DEE است.
- *عبارت‌های SQL*: یک الگوریتم ادراکی برای ارزیابی SELECT-FROM-WHERE ارائه نمودم و اشاره کردم که این قالب در برخی موارد بسیار انعطاف ناپذیر است.
- تفاوت *خلاصه‌کننده‌ها* و *عملگرهای* جمعی: (بدون اینکه زیاد وارد جزئیات شوم) بر این نکته پای‌فشردم که این دو با هم متفاوتند.
- *تبدیل عبارت‌ها*: قانون‌های توزیع‌پذیری، جابجایی‌پذیری و شرکت‌پذیری و نقش آنها در بهینه‌سازی (بازنویسی کوئری‌ها) را شرح دادم.
- *WITH*: از کاربرد WITH در ساده کردن عبارات صحبت کردم و این‌که چگونه با نام‌گذاری زیرعبارت‌ها این کار را انجام می‌دهد.

تمرین‌ها

1- از نقطه نظر رابطه‌ای، هر یک از عبارات SQL زیر چه اشکالی دارند (اگر دارند)؟

a. SELECT * FROM S, SP
b. SELECT S.SNO, S.CITY FROM S
c. SELECT SP.SNO, SP.PNO, 2 * SP.QTY FROM SP
d. SELECT P.PNO FROM P
e. SELECT S.SNO FROM S, SP
f. SELECT S.SNO FROM S ORDER BY S.CITY DESC

2- به همان دلیل که بسته بودن در ریاضیات اهمیت دارد، در جبر رابطه‌ای نیز مهم است. به هر حال این ویژگی در ریاضیات در یک حالت نقض می‌شود. تقسیم بر صفر. آیا چنین اتفاقی در جبر رابطه‌ای هم پیش می‌آید؟

3- با مقدارهای معمول پایگاه توزیع‌کنندگان و قطعات، حاصل عبارت JOIN{S,SP,P} چه می‌شود؟ *توجه*: در اینجا یک نکته انحرافی وجود دارد. به دلیل همین نکته است که بعضی معتقدند پیوند عملگر خطرناکی است (و من در متن فصل در مورد آن صحبت کردم). نظر شما چیست؟

4- فکر می‌کنید چرا نام عملگر پرتو را پرتو گذاشته‌اند؟

5- با مقدارهای معمول پایگاه توزیع‌کنندگان و قطعات، مقدار برگشتی عبارات زیر چه خواهد بود؟ در مورد هر یک از کوئری‌ها تفسیری غیررسمی و با بیان عادی بگویید.

a. S SEMIJOIN (SP WHERE PNO = PNO('P2'))
b. P NOT MATCHING (SP WHERE SNO = SNO('S2'))
c. S { CITY } MINUS P { CITY }
d. (S { SNO, CITY } JOIN P { PNO, CITY }) { ALL BUT CITY }
e. JOIN { (S RENAME (CITY AS SC)) { SC } ,
(P RENAME (CITY AS PC)) { PC } }

6- اجتماع، اشتراک، ضرب و پیوند همگی جابجایی‌پذیر و شرکت‌پذیرند. این ادعا را بررسی کنید. همچنین بررسی کنید که نیم‌پیوند شرکت‌پذیر است ولی جابجایی پذیر نیست.

7- در چه شرایطی (اگر ممکن است)، r SEMIJOIN s و s SEMIJOIN r مساوی می‌شوند؟

8- تعریف کدامیک (اگر هست) از عملگرهای جبررابطه‌ای به تساوی تاپل‌ها ربطی ندارد؟

9- در بند FROM از SQL، FROM $t_1, t_2,...t_n$ (که هر t_i نماینده یک جدول است) ضرب دکارتی آرگومان‌ها را برمی‌گرداند. ولی اگر $n=1$ باشد چه می‌شود؟ ضرب دکارتی فقط یک جدول چه نتیجه‌ای دارد؟ و یک مساله دیگر؛ ضرب دکارتی $t1$ و $t2$ در شرایطی که هر دو $t1$ و $t2$ دارای سطرهای تکراری باشند، چه نتیجه‌ای دارد؟

10- نشان دهید که تغییر نام، اولیه نیست؟

11- تفاوت عملگرهای خلاصه‌سازی و توابع جمعی چیست؟ SQL با این مورد چطور برخورد می‌کند؟

12- عبارتی بنویسید که بجای خلاصه‌سازی از گسترش استفاده کند و از نظر منطقی معادل این عبارت باشد:

SUMMARIZE SP PER (S { SNO }) ADD (COUNT ( ) AS NP)

13- با مقدارهای معمول پایگاه توزیع‌کنندگان و قطعات، مقدار عبارات زیر چه خواهد بود؟ در هر مورد با بیان عادی یک تفسیر غیررسمی از کوئری بگویید.

a. EXTEND S ADD ('Supplier' AS TAG)
b. EXTEND (P JOIN SP) ADD (WEIGHT * QTY AS SHIPWT)
c. EXTEND P ADD (WEIGHT * 454 AS GMWT, WEIGHT * 16 AS OZWT)
d. EXTEND S
   ADD (COUNT ((SP RENAME (SNO AS X)) WHERE X = SNO) AS NP)
e. SUMMARIZE S BY { CITY } ADD (AVG (STATUS) AS AVG_STATUS)

14-کدامیک از این عبارت‌ها با هم معادل هستند؟

a. SUMMARIZE r PER (r { }) ADD (COUNT ( ) AS CT)
b. SUMMARIZE r ADD (COUNT ( ) AS CT)
c. SUMMARIZE r BY { } ADD (COUNT ( ) AS CT)
d. EXTEND TABLE_DEE ADD (COUNT (r) AS CT)

15- به تفاوت این دو عبارت توجه کنید. اولی جمع مقدار تمام سفارش‌های توزیع‌کننده S1 و دومی جمع مقدار تمام سفارش‌های توزیع‌کننده S1 به صورت *مجزا distinct*. SQL در این مورد چه راه‌کاری دارد؟

16- در توتوریال‌دی، اگر اجرای تابع جمعی با مجموعه‌ای تهی از آرگومان‌ها انجام شود، COUNT صفر برمی‌گرداند و SUM هم همینطور. MAX و MIN به ترتیب بزرگترین و کوچکترین مقدار آن نوع را برمی‌گردانند، AND و OR به ترتیب صحیح و غلط را برمی‌گردانند، AVG منجر به خطا می‌شود (از بقیه توابع جمعی توتوریال‌دی صرف‌نظر می‌کنم). SQL در این شرایط چه می‌کند و چرا؟

17- فرض کنید رابطه R4 شکل 5-1 مقدار فعلی مرابطه *R* را نشان می‌دهد. اگر R4 دارای همان تعریفی باشد که در فصل 2 گفته شد، گزاره‌نمای مرابطه *R* را بنویسید.

18- فرض کنید *r* رابطه‌ای است که با عبارت زیر مشخص می‌شود:

SP GROUP ({ } AS X)

با مقدارهای همیشگی مثال خودمان، *r* به چه شکلی در می‌آید؟ عبارت زیر چه نتیجه‌ای تولید می‌کند؟

r UNGROUP (X)

19- بدون استفاده از اختصار IS_EMPTY، یک عبارت توتوریال‌دی بنویسید که اگر مقدار فعلی مرابطه P خالی باشد، صحیح و در غیر این‌صورت غلط برگرداند. معادل SQL این عبارت را هم بنویسید.

20- به توتوریال‌دی و/ یا SQL برای کوئری‌های زیر در پایگاه توزیع‌کنندگان و قطعات عبارتی بنویسید.

الف) تمامی سفارش‌ها را بده.

ب) شماره توزیع‌کنندگانی را بده که قطعه P1 را توزیع می‌کنند.

ج) توزیع‌کنندگانی را بده که رتبه‌شان بین 15 و 25 باشد.

د) شماره قطعاتی را بده که توسط توزیع‌کنندگان لندنی توزیع می‌شوند.
ه) شماره قطعاتی را بده که توسط هیچ توزیع‌کننده لندنی توزیع نمی‌شوند.
و) نام شهرهایی را بده که حداقل دو توزیع‌کننده درآنها وجود دارد.
ز) تمام زوج قطعاتی را بده که هر دو توسط یک توزیع‌کننده توزیع می‌شوند.
ح) تعداد قطعاتی را بده که توسط S1 توزیع می‌شوند.
ط) شماره توزیع‌کنندگانی را بده که رتبه‌شان از رتبه S1 پایین‌تر(کمتر) باشد.
ی) شماره توزیع‌کنندگانی را بده که نام شهرشان در لیست شهرها (به ترتیب الفبا)، اول باشد.
ک) شماره قطعاتی را بده که توسط تمام توزیع‌کنندگان لندنی، توزیع می‌شوند.
ل) زوج (شماره قطعه/شماره توزیع‌کننده)هایی را بده که توزیع‌کننده، قطعه را توزیع نکند.
م) توزیع‌کنندگانی را بده که دست‌کم تمام قطعاتی که S2 توزیع می‌کند را، توزیع کنند.
ن) شماره توزیع‌کنندگانی را بده که دست‌کم تمام قطعاتی که حداقل توسط یک توزیع‌کننده که قطعه‌ای از لندن دارد، توزیع می‌شوند، را توزیع کند.

21- ادعاهای زیر را ثابت کنید (در صورت نیاز آنها را مختصر کنید):
الف) چند گزینش متوالی از یک رابطه، می‌توانند به یک گزینش تبدیل شوند.
ب) چند پرتو متوالی از یک رابطه، می‌توانند به یک پرتو تبدیل شوند.
ج) یک گزینش از یک پرتو می‌تواند به یک پرتو از گزینش تبدیل شود.

22- اجتماع یک عملگر خودتوان است. چرا که r UNION r برای تمام rها مساوی خود r است. (آیا در SQL هم همینطور است؟) همانطور که احتمالا حدس زده‌اید، خاصیت خودتوانی می‌تواند در تبدیل عبارت‌ها به‌کار رود. کدامیک از عملگرهای رابطه‌ای دیگر (اگر هست)، دارای این خاصیت هستند؟

23- فرض کنید r یک رابطه است. عبارت $r\{\ \}$ چه معنایی دارد؟ چه نتیجه‌ای برمی‌گرداند؟

24- این عبارت بولی:

$$x > y \text{ AND } y > 3$$

(که ممکن است بخشی از یک کوئری باشد) با عبارت زیر معادل است (و می‌تواند به آن تبدیل شود):

$x > y$ AND $y > 3$ AND $x > 3$

این تساوی بر این مبناست که عملگر مقایسه < دارای خاصیت تراگذری است. این تساوی زمانی مفید و ارزشمند است که x و y از دو رابطه باشند چرا که سیستم می‌تواند قبل از عمل پیوند اصلی، یک گزینش اضافی $x>3$ انجام دهد. همانطور که در متن این فصل دیدیم، گزینش زود هنگام ایده خوبی است و اینکه سیستم بتواند گزینش‌های زودهنگام را *استنتاج* کند هم خوب است. آیا فکر می‌کنید هیچیک از محصولات تجاری امروزی امکان چنین بهینه‌سازی را دارند؟

25- عبارت‌های زیر را در نظر بگیرید:

الف. WITH (P WHERE COLOR = 'Purple') AS PP ,
(SP RENAME (SNO AS X)) AS T :
S WHERE (T WHERE X = SNO) { PNO } ⊇ PP { PNO }

ب. WITH (P WHERE COLOR = 'Purple') AS PP :
S JOIN (SP { SNO, PNO } DIVIDEBY PP { PNO })

هر دو عبارت برای این کوئری نوشته شده‌اند: «توزیع‌کنندگانی را بده که تمام قطعات بنفش را توزیع می‌کنند.» با مقدارهای همیشگی مثال خودمان، هر یک از عبارت‌ها چه نتیجه‌ای را برمی‌گردانند. اگر تفاوتی هست، کدام نتیجه صحیح است؟ توضیح دهید.

26- گزینش r WHERE bx را در نظر بگیرید. در متن گفتم هر ویژگی موجود در شرط گزینش یعنی شرط bx بایستی از ویژگی‌های r باشد، ولی این را هم گفتم که bx محدودیت‌های دیگری نیز دارد. بطور دقیق یکی از این محدودیت‌ها* این است که bx فقط می‌تواند به شکل x Op y باشد که x و y یا یکی از ویژگی‌های bx و یا مقدار ثابت هستند و Op یک عملگر مقایسه‌ای متناسب با x و y است. ولی در دنیای واقعی بند WHERE می‌تواند شامل عبارت‌های بولی با میزان پیچیدگی دلخواه باشد (فقط با محدودیت «متغیرهای آزاد»، ضمیمه را ببینید) در ویژگی r است. نشان دهید این توسعه مفهوم گزینش مجاز است، بدین صورت که این توسعه نوعی اختصار برای چیزی است که ما آن را در حال حاضر مجاز می‌دانیم.

* محدودیت دیگر این است که نباید هیچ سوری (یا هر چیز معادل عبارت سوری) در شرط باشد.

27- در اینجا دو عبارت وجود دارد که درباره رابطه R4 شکل 5-1 هستند. این دو، نماینده چه کوئری‌هایی هستند؟

```
( R4 WHERE TUPLE { PNO PNO('P2') } ∈ PNO_REL ) { SNO }

( ( R4 WHERE SNO = SNO('S2') ) UNGROUP ( PNO_REL ) ) { PNO }
```

28- با مقدارهای همیشگی مثال خودمان در پایگاه توزیع‌کنندگان و قطعات، عبارت زیر چه معنایی دارد؟

```
EXTEND S
   ADD ( ( ( SP RENAME ( SNO AS X ) ) WHERE X = SNO ) { PNO }
AS PNO_REL )
```

29- فرض کنید رابطه‌ای که از عبارت تمرین قبل برگشته است را بصورت مرابطه‌ای به نام SSP نگهداری کرده‌ایم. این به‌روزرسانی‌ها چه کاری انجام می‌دهند؟

```
INSERT SSP RELATION
  { TUPLE { SNO SNO('S6'),
     PNO_REL RELATION { TUPLE { PNO PNO('P5') } } } } ;

UPDATE SSP WHERE SNO = SNO('S2')
  ( INSERT PNO_REL RELATION { TUPLE { PNO PNO('P5') } } ) ;
```

30- با استفاده از مرابطه SSP در مثال قبل، برای این کوئری‌ها عبارت بنویسید:

- زوج توزیع‌کنندگانی را بده که دقیقا یک مجموعه قطعات را توزیع می‌کنند.
- زوج قطعاتی را بده که دقیقا توسط یک مجموعه توزیع‌کنندگان، توزیع می‌شوند.

31- کوئری سهمیه‌ای، کوئری است که سعی دارد کاردینالیتی نتیجه را محدود -یا سهمیه بندی- کند. مثلا کوئری «سه قطعه سنگین‌تر را بده». در اینجا سهمیه سه عدد است. عبارت توتوریال‌دی و SQL این کوئری را بنویسید.

فصل ششم

قیدهای جامعیت

به فهرست

من در فصل‌های گذشته به موضوع قیدهای جامعیت اشاره‌ای کوتاه داشتم، ولی اکنون می‌خواهم بطور دقیق به آن بپردازم. تعریف اولیه‌ای که در فصل 1 ارائه دادم چنین بود. یک قید جامعیت (یا بطور خلاصه قید) یک عبارت بولی است که بایستی حتما مقدار صحیح برگرداند. قیدها* به این دلیل چنین نامیده شده‌اند که مقدارهای مُجاز یک محتوای خاص را، مقید و محدود می‌کنند. قیدهایی که مورد نظر ما هستند خود به دو دسته تقسیم می‌شود، قیدهای *نوع* و قیدهای *پایگاه*. بطور دقیق، یک قید نوع مقدارهایی که با یک نوع خاص مطابقت دارند را تعیین می‌کند و یک قید پایگاه مقدارهایی که می‌توانند در یک پایگاه داده خاص ظاهر شوند را معین می‌نماید.

می‌توان گفت که قیدها نگارش رسمی *قواعد تجاری* هستند. در فصل بعد دوباره به این موضوع اشاره خواهم کرد.

قیدهای نوع

همانطور که در فصل 2 دیدیم، یکی از کارهایی که به هنگام تعریف نوع انجام می‌شود، مشخص کردن مقدارهای مجاز در آن نوع است. یک مثال:

```
1  TYPE QTY
2     POSSREP QPR
3        { Q INTEGER
4              CONSTRAINT Q ≥ 0 AND Q ≤ 5000 } ;
```

توضیح:

- خط 1 فقط می‌گوید داریم نوعی به نام QTY تعریف می‌کنیم «شناسه».
- خط 2 دارای یک «نماممکن» (*نمایش امکان‌پذیر possible representation*) به نام QPR است. همانطور که در فصل 2 دیدیم نمایش

* قید اصطلاحی است به معنای به بندکشیدن و محدودکردن بخشی از آزادی کسی یا چیزی. مثل مقید کردن حیوانات خانگی هنگام معاینه توسط دامپزشک. مترجم

فیزیکی همواره از چشم کاربر پنهان است. توتوریال‌دی برای هر نوع به حداقل یک نماممکن نیاز دارد* که شامل مقدارهایی است که در آن نوع «نمایش آنها امکان‌پذیر» است. بر خلاف نمایش فیزیکی، مشخصات نماممکن به کاربر نشان داده می‌شود (در این مثال کاربر این شرایط را تحت عنوان QPR مشاهده می‌کند). –حتما به این نکته توجه داشته باشید– با این وجود که توصیه نمی‌شود در نماممکن به نحوه پیاده‌سازی فیزیکی اشاره شود، ولی در مواردی ممکن است این کار انجام شود اما در هر صورت تفاوتی برای کاربر ندارد.

- خط 3 می‌گوید که نماممکن فقط یک جز دارد که Q نامیده می‌شود و از نوع INTEGER است. به بیان دیگر «نمایش» مقدارهایی که از نوع QTY هستند، با عدد صحیح «امکان‌پذیر» است (و کاربران از این واقعیت آگاه می‌شوند).
- و در نهایت خط 4 تعیین می‌کند که این اعداد صحیح بایستی در بازه 0 تا 5000 باشند و این امر برای نوع QTY، قید نوع به شمار می‌آید. به بیان دیگر در اینجا مقدارهای مجاز اعداد صحیح درون بازه مشخص شده، هستند.

یک مثال کمی مشکل‌تر:

```
TYPE POINT
    POSSREP CARTESIAN { X NUMERIC, Y NUMERIC
        CONSTRAINT SQRT ( X ^ 2 + Y ^ 2 ) ≤ 100.0 } ;
```

نوع POINT برای نقاطی در فضای دو بعدی است و یک نماممکن به نام CARTESIAN دارد که دارای دو جز به نام‌های X و Y از نوع عددی است. قید نوع هم می‌گوید که عملا فقط نقاطی مورد نظر هستند که در محدوده و یا داخل دایره‌ای به شعاع 100 که مرکز آن مبدا مختصات است، قرار داشته باشند.

انتخابگرها و عملگرهای _THE

قبل از ادامه بحث قیدهای نوع می‌خواهم گریزی به یک موضوع دیگر بزنم. مثال‌های QTY و POINT دارای نکاتی هستند که باید آنها را روشن کنم و همین‌جا را برای این کار انتخاب کرده‌ام.

* استثناهای کوچکی وجود دارند که در اینجا نیازی به بحث در مورد آنها وجود ندارد.

از فصل 2 یادآوری می‌کنم که نوع‌های تعریف شده بوسیله کاربر مانند QTY و POINT دارای یک انتخابگر و یک عملگر _THE هستند. این عملگرها دارای ارتباطی معنادار با نامماکن هستند. در حقیقت انتخابگرها با نامماکن، و عملگرهای _THE با اجزا نامماکن دارای یک تناظر یک به یک هستند. چند مثال:

QPR(250)

این عبارت انتخابگر نوع QTY را اجرا می‌کند (انتخابگر با نوع هم‌نام است). این انتخابگر زیربنای نامماکن آن نوع است و یک آرگومان ورودی دارد که متناظر و هم‌نوع آرگومان با جزء اصلی نامماکن است و یک مقدار بازمی‌گرداند که از نوع QTY است. *توجه*: نامماکن‌ها معمولا در عمل با نوع‌ها هم‌نام هستند. من در QTY برای روشن کردن تفاوت منطقی این دو، از نام‌های متفاوت استفاده کردم ولی در عمل معمولا چنین اتفاقی نمی‌افتد. در واقع طبق قواعد نوشتاری می‌توانیم در صورت تمایل نام نامماکن را حذف کنیم که در این صورت نام آن بصورت پیش فرض هم‌نام با نوع در نظر گرفته می‌شود. پس بیایید تعریف نوع QTY را تغییر دهیم:

TYPE QTY POSSREP { Q INTEGER CONSTRAINT Q ≥ 0 AND Q ≤ 5000 } ;

حال نامماکن و انتخابگر متناظر آن، هر دو QTY نامیده می‌شوند و اجرای انتخابگر فقط بصورت QTY(250) در می‌آید. این حالتی است که در این کتاب قبل از تشریح این نکته هم برای نمایش انتخابگرها به کار می‌رفت. من فرض می‌کنم که این تعریف جدید برای QTY در سراسر این فصل معتبر است.

QTY (A + B)

آرگومان انتخابگر QTY می‌تواند به اندازه دلخواه پیچیده باشد (البته در حدی که از نوع INTEGER خارج نشود). اگر آرگومان ثابت باشد – مانند مثال قبل- اجرای انتخابگر هم ثابت خواهد بود به بیان دیگر فرم ثابت حالت خاصی از اجرای یک انتخابگر است، چیزی که در فصل 3 مشاهده کردیم.

THE_Q (QZ)

این عبارت اجرای عملگر _THE برای نوع QTY را نشان می‌دهد. دلیل اینکه عملگر THE_Q نامیده شده این است که جز اصلی نامماکن QTY، Q نام دارد و آرگومانی از نوع QTY (با هر مقدار پیچیدگی) دارد و مقداری از نوع عدد صحیح را بازمی‌گرداند.

حال بیایید نوع PIONT را به گونه‌ای تعریف کنیم که نام نماممکن آن با نام نوع یکی باشند. (برای سادگی فعلا از قید نوع صرفنظر می‌کنیم):

TYPE POINT POSSREP { X NUMERIC, Y NUMERIC } ;

با مثال مطلب را ادامه می‌دهیم:

```
POINT ( 5.7, -3.9 )
```

این یک اجرا از انتخابگر POINT (در واقع یک ثابت از نوع POINT) است.

```
THE_X ( P )
```

این عبارت مقدار عددی X (طول) نقطه را از زوج دکارتی نقطه P را بر می‌گرداند. (P مقداری است از نوع POINT).

و اما برای نمایش نوع‌هایی که می‌توانند بیش از یک نماممکن داشته باشند، POINT نمونه خوبی است. مثلا:

TYPE POINT POSSREP CARTESIAN { X NUMERIC, Y NUMERIC }
POSSREP POLAR { R NUMERIC, THETA NUMERIC } ;

دو نماممکن موجود در اینجا نتیجه این واقعیت هستند که نقاط در فضای دو بعدی به دو صورت «ممکن است نمایش» داده شوند. مختصات دکارتی و مختصات قطبی. هر نماممکن دارای دو جز است که در هر صورت اجزا از نوع عددی و قابل مشاهده برای کاربر هستند. با مثال ادامه می‌دهیم:

```
POLAR ( H, K )
```

این یک اجرا از انتخابگر POINT است و مقداری از نوع POINT باز می‌گرداند.

```
THE_THETA ( P )
```

این عبارت مقدار عددی تتا از زوج نمایش قطبی نقطه P را برمی‌گرداند (که باز هم بایستی از نوع POINT باشد).

قیدهای همیشگی نوع

بیایید به بحث قیدهای نوع برگردیم. فرض کنید که من نوع QTY را بدون هیچ قید صریحی تعریف کرده‌ام:

```
TYPE QTY POSSREP { Q INTEGER } ;
```

این تعریف مختصر شده تعریف زیر است:

```
TYPE QTY POSSREP { Q INTEGER CONSTRAINT TRUE } ;
```

با هر کدام از این تعریف‌ها، هر مقدار عددی صحیح می‌تواند به صورت مقداری از نوع QTY نمایش داده شود (QTY هنوز هم تحت نظارت یک قید قرار دارد ولی این قید خیلی شل است). به بیان دیگر این نماممکن خاص ملزم به رعایت قیدی از نوع *اصل موضوعه* (قانونی که همه درست بودن آن را قبول داریم) است و اگر قید خاصی هم وجود داشته باشد بر این قید افزوده می‌شود و زمانی که از کلمه قید استفاده می‌کنیم هم منظور همین قید افزوده است.

موضوع مهم دیگر این است که قیدهای نوع در چه زمانی کنترل می‌شوند. در واقع این قیدها *هر وقت که یک انتخابگر اجرا شود* مورد بررسی قرار می‌گیرند. دوباره فرض کنید که نوع QTY تحت نظر قیدی قرار دارد که فقط اعداد صحیح بین 0 تا 5000 را مجاز می‌شمارد. حال عبارت QTY(250) یک اجرا از انتخابگر QTY است که با موفقیت به انجام می‌رسد، در مقابل QTY(6000) هم یک اجرا است ولی با شکست روبرو می‌شود. بطور کلی ما هرگز نمی‌توانیم عبارتی را تحمل کنیم که وانمود می‌کند دارای مقداری از نوع T است، در حالی که واقعا چنین نیست. «مقداری از نوع T» که مقداری از نوع T نیست دارای تناقض در کلام است. در نتیجه به هیچ متغیری – و بطور خاص هیچ مرابطه‌ای- نمی‌توان مقداری از نوع نادرست، منتسب کرد.

تذکری درباره SQL: به شما گفته شد که تمامی مثال‌های این قسمت با توتوریال‌دی –و نه با SQL- نوشته شده‌اند. علت این کار آن است که SQL –چه باورتان بیاید و چه نیاید- اصلا از قیدهای نوع پشتیبانی نمی‌کند* (البته بجز اصل موضوعه). به زبان دیگر، گر چه SQL به شما اجازه می‌دهد که نوعی به نام QTY از نوع عدد صحیح تعریف کنید، ولی نمی‌گذارد که بگویید این اعداد صحیح بایستی در محدوده خاصی باشند. به همین دلیل و دلایل دیگر من در این کتاب

* از نماممکن‌ها هم نمی‌کند، از انتخابگرها و عملگرهای _THE هم نمی‌کند. اوضاع مغشوش‌تر از آن است که در اینجا بحث کنم فقط به همین بسنده می‌کنم که معمولا نظیر این عملگرها وجود دارند ولی مانند توتوریال‌دی به صورت خودکار آماده نمی‌شوند.

وارد بحث نوع‌های تعریف شده توسط کاربر، در زبان SQL نمی‌شوم و فقط معادل تعریف QTY و PIONT را به شما نشان می‌دهم:

```
CREATE TYPE QTY AS INTEGER FINAL ;

CREATE TYPE POINT AS ( X NUMERIC, Y NUMERIC ) NOT FINAL ;
```

و نکته آخر قبل از پایان این بخش؛ تعریف هر چیز از یک نوع خاص، قیدی بر پای آن می‌بندد. پس تعریف ویژگی QTY در مرابطه SP از نوع QTY، موجب می‌شود که هیچ تاپلی در SP پذیرفته نشود مگر آنکه مقدار موجود در QTY آن با شرایط نوع QTY مطابقت داشته باشد. (این مورد گاهی اوقات *قید ویژگی* نامیده می‌شود.)

قیدهای پایگاه

برای یادآوری، یک قید پایگاه، مقدارهایی که می‌توانند در پایگاه ظاهر شوند را مقید و محدود می‌کند. در توتوریال‌دی چنین قیدهایی با کلمه CONSTRAINT (و یا مختصرنویسی‌هایی که معادل آن هستند) مشخص می‌شوند. در SQL نیز این قیدها به کمک CREATE ASSERTION (و یا باز هم معادل‌های مختصر شده آن) شناخته می‌شوند. من قصد ندارم وارد جزئیات مختصرنویسی‌ها شوم (دست‌کم الان) چرا که این بحث بیشتر به نحوه دستورنویسی مربوط است. اجازه دهید فعلا با حالت مختصر نشده سر کنیم. کار را با تعدادی مثال شروع می‌کنم (مثل همیشه توتوریال‌دی سمت چپ و SQL سمت راست):

مثال 1

رتبه‌ها بایستی در محدوده 1 تا 100 باشند:

```
CONSTRAINT C1 IS_EMPTY|CREATE ASSERTION C1 CHECK
( S WHERE STATUS < 1  |( NOT EXISTS
  OR   STATUS > 100 ) ;|( SELECT S.* FROM S
                       |  WHERE  S.STATUS < 1
                       |  OR     S.STATUS > 100 ) ) ;
```

این قید فقط یک ویژگی از یک مرابطه را درگیر کرده است. بنابراین بررسی آن برای یک تاپل توزیع‌کننده، فقط با کنترل آن تاپل به تنهایی امکان پذیر است و نیازی نیست که تاپل‌های دیگر این مرابطه و یا مرابطه‌های دیگر بررسی شوند. (به همین دلیل برخی اوقات چنین قیدهایی، *قید تاپل* نامیده می‌شوند.)

مثال 2

توزیع‌کنندگان لندنی بایستی رتبه 20 را دارا باشند.

```
CONSTRAINT C2 IS_EMPTY          | CREATE ASSERTION C2 CHECK
( S WHERE CITY = 'London'       |   ( NOT EXISTS
  AND   STATUS ≠ 20 ) ;         |   ( SELECT S.* FROM S
                                |     WHERE  S.CITY = 'London'
                                |     AND    S.STATUS <> 20 ) ) ;
```

این قید دو ویژگی مجزا (از یک مرابطه) را درگیر خود می‌کند. با این وجود این حالت هنوز (مانند C1) با وارسی یک تاپل توزیع‌کننده –به تنهایی– قابل کنترل است.

مثال 3

هیچ دو تاپلی از S نبایستی شماره توزیع‌کننده یکسان داشته باشند (به بیان دیگر {SNO} برای یکی از کلیدهای این مرابطه است):*

```
CONSTRAINT C3                   |  CREATE ASSERTION C3 CHECK
  COUNT ( S ) =                 |    ( UNIQUE ( SELECT S.SNO
  COUNT ( S { SNO } ) ;         |          FROM   S ) ) ;
```

مانند قیدهای C1 و C2، این قید هم فقط در محدوده یک مرابطه قرار دارد ولیکن روشن است که بررسی آن برای یک تاپل دیگر با کنترل همان تاپل به تنهایی، امکان‌پذیر نیست. البته بسیار بعید است که در عمل قید C3 صورت خلاصه نشده آورده شود. بدیهی است که در این مورد استفاده از اختصار KEY بهتر است. من حالت مختصر نشده را به این دلیل آورده‌ام که نشان دهم این اختصارها در نهایت فقط اختصاراند.

* و یا یک فراکلید. فراکلید مجموعه مادر کلید است. در فصل هفتم ادامه بحث را ببینید.

با این حال عبارت SQL مربوط به C3 نیاز به کمی توضیح دارد. UNIQUE یک عملگر SQL است که مقدار صحیح برمی‌گرداند اگر و تنها اگر ستونی که بعنوان آرگومان آن ذکر شده است، دارای مقدارهای یکتا باشد. اجرای عبارت زیر صحیح برمی‌گرداند اگر و تنها اگر هیچ دو سطری از جدول S دارای شماره توزیع‌کننده یکسان نباشند. ولی توجه داشته باشید اگر به علت انضباط همیشگی توصیه شده توسط من، از DISTINCT استفاده می‌کردید:

```
UNIQUE ( SELECT DISTINCT S.SNO FROM S )
```

دیگر UNIQUE در هیچ شرایطی نمی‌توانست غلط برگرداند! در حقیقت مدل رابطه‌ای هیچ نسبتی با UNIQUE ندارد، ولی SQL به آن نیاز دارد چرا که جدول‌های SQL بطور کلی رابطه نیستند.

در اینجا یک عبارت SQL می‌بینید که برای قید C3 نوشته شده است و به عبارت‌های توتوریال‌دی شباهت بیشتری دارد:

```
CREATE ASSERTION C3 CHECK
( ( SELECT COUNT ( * ) FROM S ) =
  ( SELECT COUNT ( SNO ) FROM S ) ) ;
```

مثال 4

هیچ توزیع‌کننده‌ای با رتبه کمتر از 20 نمی‌تواند قطعه P6 را توزیع کند.

```
CONSTRAINT C4 IS_EMPTY
( ( S JOIN SP )
  WHERE STATUS < 20
  AND   PNO = PNO('P6') ) ;
```

```
| CREATE ASSERTION C4 CHECK
|   ( NOT EXISTS
|   ( SELECT *
|     FROM   S, SP
|     WHERE  S.SNO = SP.SNO
|     AND  S.STATUS < 20
|     AND  SP.PNO = PNO('P6') ) ) ;
```

این قید دو مرابطه S و SP را درگیر – و در واقع به هم وابسته- می‌کند. یک قید پایگاه می‌تواند هر تعداد مرابطه مستقل را درگیر خود و یا وابسته کند.

توجه

اصطلاح فنی: به قیدی که فقط یک مرابطه را درگیر کند بطور غیر رسمی *قید مرابطه* می‌گویند (قید *تک* مرابطه‌ای). قیدی که دو یا چند مرابطه مجزا را درگیر خود کند بطور غیر رسمی قید *چند مرابطه‌ای* نامیده می‌شود.

مثال 5

هر شماره توزیع‌کننده حاضر در مرابطه SP بایستی در مرابطه S هم وجود داشته باشد.

```
CONSTRAINT C5                    | CREATE ASSERTION C5 CHECK
   SP { SNO } ⊆ S { SNO } ;      |   ( NOT EXISTS
                                 |     ( SELECT SP.SNO
                                 |       FROM   SP
                                 |       EXCEPT
                                 |       SELECT S.SNO
                                 |       FROM   S ) ) ;
```

همانطور که در عبارت توتوریال‌دی می‌بینید، در این قید از مقایسه رابطه‌ای استفاده شده است. SQL از مقایسه‌های رابطه‌ای پشتیبانی نمی‌کند، پس نمی‌توان به طولانی بودن عبارت SQL معترض شد. می‌دانیم که {SNO} یک کلید -و در حقیقت تنها کلید- مرابطه S است، پس قید C5 اساسا قید کلید خارجی بین SP و S است. دستور FOREIGN KEY نیز تنها یک اختصار برای C5 است.

قیدهای پایگاه کی بررسی می‌شوند؟ اعتقاد عمومی این است که بررسی قید تک مرابطه‌ای باید فوری باشد (بدین معنا که در زمان به‌روزرسانی مرابطه مورد نظر انجام شود) و بررسی قید چند مرابطه‌ای تا پایان تراکنش به *تعویق* افتد (زمان تکمیل). من می‌خواهم در اینجا عقیده خود را همراه با دلیل مطرح کنم مبنی بر اینکه بررسی *تمامی* قیدهای پایگاه بایستی فوری باشد و به تعویق انداختن این بررسی -که توسط SQL استاندارد و برخی محصولات SQL انجام می‌شود- یک اشتباه منطقی است. برای طرح این دیدگاه غیر معمول، ابتدا بایستی بحث تراکنش‌ها را پیش بکشم.

تراکنش‌ها

موضوع تراکنش‌ها بحثی طولانی است و همانطور که در فصل 4 ذکر شد (دست‌کم مستقیما) ربطی به مدل رابطه‌ای ندارد و به همین علت در اینجا قصد ندارم این بحث را با تمام جزئیات باز کنم. در هر صورت شما یک حرفه‌ای پایگاه داده هستید و مطمئنا با مبانی بحث تراکنش‌ها آشنایی دارید. در این مورد کتاب مرجع توصیه شده، «پردازش تراکنش‌ها: نظریه‌ها و

تکنیک‌ها» نوشته جیم گری و آندریاس رویتر است. من در اینجا تنها مروری مختصر دارم بر ویژگی‌های *ACID* در تراکنش‌ها.

ACID (اثتم) خلاصه‌ای است از کلمات *atomicity*، *consistency*، *isolation* و *durability* و به معنای اتمی‌بودن، ثبات (سازگاری)، تنهایی و ماندگاری

اتمی بودن

تراکنش‌ها تابع قانون «همه یا هیچ» اند.

ثُبات

هر تراکنش پایگاه داده را از یک وضعیت با ثبات به وضعیت با ثبات دیگر می‌برد، بدون اینکه لزوما ثبات در نقاط بین این دو حالت برقرار باشد.*

تنهایی

تا زمانی که یک تراکنش تکمیل نشده است، به روزرسانی‌های مربوط به آن تراکنش بایستی از دید بقیه تراکنش‌ها پنهان باشند.

ماندگاری

وقتی که یک تراکنش تکمیل می‌شود، به روزرسانی‌های صورت گرفته توسط آن در پایگاه باقی می‌مانند، حتی اگر عملیات بعد از آن با شکست مواجه شود.

یک ادعای معروف تراکنش‌ها را به عنوان «واحد جامعیت» معرفی می‌کند (ویژگی ثبات به طور کلی در همین مورد است). ولی من با این ادعا موافق نیستم، همانطور که تاکنون کم و بیش گفته‌ام معتقدم که عبارت‌ها بایستی واحد جامعیت باشند (نه تراکنش‌ها) و از همین رو قیدهای پایگاه بایستی در سطح عبارت باشند. در قسمت بعد نظرم را در این مورد بیان می‌کنم.

* یک پایگاه داده با ثبات است اگر و تنها اگر تمام قیدهای تعریف شده در آن رعایت شده باشند (در این زمینه ثبات هم‌معنای جامعیت است).

چرا بررسی قیدهای پایگاه بایستی فوری انجام شود؟

من برای اینکه قیدهای پایگاه بایستی در سطح عبارت بررسی شوند، پنج دلیل دارم. اولین و مهمترین آنها چنین است. همانطور که در فصل 4 دیدیم به پایگاه داده‌ها می‌توان به شکل مجموعه‌ای از گزاره‌ها نگریست، گزاره‌هایی که ما به درست بودن آنها ایمان داریم و اگر کوچکترین ناسازگاری (تناقضی) بتواند به مجموعه راه یابد، *همه چیز بهم می‌خورد* (از همین رو ناسازگاری و بی‌ثباتی در پایگاه یک معنا دارند). همانطور که در بخش «قیدها و گزاره‌نماها» نشان خواهم داد، هیچگاه نمی‌توان به پاسخ‌هایی که از یک پایگاه بی‌ثبات دریافت می‌شود، اعتماد کرد! به همین دلیل باید مدیون ویژگی *تنهایی* باشیم که موجب می‌شود که یک بی‌ثباتی خاص در بیش از یک تراکنش دیده نشود. ولی هنوز بی‌ثباتی در یک تراکنش خاص می‌تواند مشاهده شود و همین امر ممکن است موجب ایجاد پاسخ‌های غلط شود.

تصور می‌کنم اولین دلیل به اندازه کافی محکم باشد ولی چهار دلیل دیگر نیز دارم که برای کامل بودن کتاب به عنوان مرجع به آنها اشاره می‌کنم. دوم این‌که، من اعتقادی به ویژگی تنهایی ندارم بنابراین تصور نمی‌کنم که یک بی‌ثباتی را فقط یک تراکنش می‌بینند. کلمات تراکنش و تنهایی حتی در معنای لغوی بر ضد هم هستند و هیچ قانونی هم نمی‌گوید که تراکنش‌ها نبایستی با یکدیگر ارتباط داشته باشند. اگر در پایگاه داده‌ها و یا هر جای دیگر تراکنش *T1* نتیجه‌ای تولید کند و سپس تراکنش *T2* از آن استفاده کند، آنگاه *T1* و *T2* واقعا تنها و مجزا نیستند (بدون توجه به این که این دو تراکنش بصورت همروند اجرا می‌شوند یا خیر). بنابراین اگر الف)*T1* یک وضعیت بی‌ثبات را در پایگاه ببیند و نتیجه‌ای اشتباه تولید کند. و بعد ب)نتیجه توسط *T2* دیده شود. آنگاه ج)بی‌ثباتی دیده شده توسط *T1* در عمل به *T2* هم سرایت کرده است. به بیان دیگر هیچ تضمینی وجود ندارد که یک بی‌ثباتی - اگر به آن اجازه ظهور داده شود- فقط در محدوده یک تراکنش باقی بماند.

سوم این‌که، ما مطمئنا نمی‌خواهیم که اجرای یک برنامه (قطعه کد) موجب بی‌ثباتی پایگاه شود. این دلیل عدم وجود استقلال است که به دلیل تعویق بررسی قیدها، برنامه‌ای نمی‌تواند از ثبات لازم برخوردار شود. به بیان دیگر من می‌خواهم یک قطعه کد از محیطی که برای اجرا دارد مستقل باشد. خواه خود یک تراکنش باشد و خواه قسمتی از یک تراکنش. (در واقع به دنبال پشتیبانی از *تراکنش‌های تودرتو* هستم ولی این بحث را به وقتی دیگر موکول می‌کنم.)

چهارم این‌که، از *اصل تعویض‌پذیری* (از بخش مرابطه‌های پایه و ویوها در فصل 4) نتیجه‌گیری می‌شود که قیدهای تک مرابطه‌ای پایگاه و قیدهای چند مرابطه‌ای شکل دیگر پایگاه، ممکن است در واقع یکی باشند. بعنوان مثال دوباره دو ویو از فصل 4 را ملاحظه کنید:

```
VAR LS  VIRTUAL ( S WHERE CITY = 'London' ) ;
VAR NLS VIRTUAL ( S WHERE CITY ≠ 'London' ) ;
```

این ویوها تابع این قید هستند که هیچ شماره توزیع‌کننده‌ای در هر دو آنها وجود ندارد. ولی لازم نیست این قید بطور صریح آورده شود چرا که به همین موضوع با قید تک مرابطه‌ای {SNO} کلید مرابطه S است، اشاره شده است (با توجه به این واقعیت که هر توزیع‌کننده لزوما دارای دقیقا یک شهر است). حال تصور کنید که ما LS و NLS را به مرابطه‌های پایه تبدیل و اجتماع آنها را بعنوان ویو S تعریف کنیم. حال بایستی این قید بطور صریح آورده شود:

```
CONSTRAINT C6 IS_EMPTY
( LS { SNO } JOIN
 NLS { SNO } ) ;
```

```
| CREATE ASSERTION C6 CHECK
| ( NOT EXISTS
|   ( SELECT *
|   FROM   LS, NLS
| WHERE  LS.SNO = NLS.SNO ) ) ;
```

حال بعد از تغییر قید تک مرابطه‌ای اول به قید چند مرابطه‌ای دوم، آیا چیزی تغییر کرد؟ بنابراین اگر بر این عقیده هستید که قیدهای تک مرابطه‌ای باید فوری بررسی شوند، قیدهای چند مرابطه‌ای هم بایستی به همان صورت یعنی فوری مورد بررسی قرار گیرند.

پنجم این‌که، یک روش بهینه‌سازی به نام *بهینه‌سازی معنایی* وجود دارد (این بحث در مورد تبدیل عبارت‌ها است ولی من عمدا در مورد آن در فصل 5 چیزی نگفتم). برای مثال عبارت زیر را در نظر بگیرید:

```
(SP JOIN S){PNO}
```

بدیهی است که این پیوند کلید خارجی به کلید اصلی است. بنابراین هر تاپل SP با تاپل‌هایی از S پیوند می‌خورد و فقط یک شماره قطعه در نتیجه نهایی ظاهر می‌شود. ولی در اینجا اصلا نیازی به پیوند وجود ندارد. این عبارت می‌تواند فقط به این شکل نوشته شود:

```
SP {PNO}
```

نکته قابل توجه این است که این تبدیل عبارت *فقط* به دلیل وضعیت خاص معنایی مجاز است. بطور کلی هر عملوند در پیوند احتمالا دارای تاپل‌هایی است که در عملوند دیگر متناظری ندارد و طبیعتاً در نتیجه نهایی نیز ظاهر نمی‌شوند. پس چنین تبدیل عبارتی نمی‌تواند مجاز باشد.

ولی در این مورد هر تاپل SP *بایستی* دارای متناظری در S باشد. دلیل این امر قید جامعیت –در واقع قید کلید خارجی– است که می‌گوید هر سفارش حتما بایستی دارای یک توزیع‌کننده باشد. در این صورت است که انجام چنین تبدیلی امکان‌پذیر است. تبدیلی که به دلیل فعال بودن قید جامعیت مجاز است، تبدیل معنایی و نتیجه این تبدیل بهینه‌سازی معنایی نامیده می‌شود.

در اصل هر قیدی می‌تواند در بهینه‌سازی معنایی به کار گرفته شود (این کار به قید کلید خارجی محدود نمی‌شود). برای مثال فرض کنید این قید در پایگاه توزیع‌کنندگان و قطعات برقرار است «تمام قطعات قرمز بایستی در لندن نگهداری شوند»، و کوئری زیر را در نظر بگیرید:

توزیع‌کنندگانی را بده که فقط قطعات قرمز توزیع می‌کنند و در شهری قرار دارند که حداقل یکی از قطعاتی که توزیع می‌کنند در آنجا واقع است.

این کوئری واقعا پیچیده است. ولی با استفاده از قید جامعیت می‌تواند به سادگی به کوئری زیر تبدیل –منظور تبدیل بوسیله بهینه‌ساز است نه کاربر– شود:

توزیع‌کنندگان لندنی که حداقل یک قطعه قرمز توزیع می‌کنند را بده.

در اینجا می‌توان بهبود بسیار زیادی را در کارایی مشاهده کرد با این حال تا زمان نوشتن این کتاب (تا آنجا که من می‌دانم) محصولات اندکی آن را به کار گرفته‌اند ولی برای من مثل روز روشن است که در آینده این کار بیش از این انجام خواهد شد چرا که کاهش هزینه‌ها بوسیله آن غافلگیرکننده خواهد بود.

جهت بازگشت به بحث اصلی، حال می‌دانیم که برای استفاده از قیدها در بهینه‌سازی معنایی این قیدها بایستی *همواره* برقرار باشند (در سطح عبارت)، نه فقط در سطح تراکنش‌ها. همانطور که دیدیم بهینه‌سازی معنایی یعنی استفاده از قیدها برای ساده کردن کوئری‌ها و به منظور بالا بردن کارایی. بدیهی است که اگر قیدی در زمانی زیرپا گذاشته شود، آنگاه در آن زمان ساده‌سازی بر پایه آن قید معتبر نخواهد بود و کوئری ساخته شده با استفاده از آن ساده‌سازی در آن زمان غلط خواهد بود (بطور کلی).

توجه

البته می‌توانیم خود را با شرایط نامطلوب قیدهای معوقه (یعنی قیدهایی که بررسی آنها به تعویق افتاده است) سازگار کنیم، بدین معنا که این‌گونه قیدها را در بهینه‌سازی معنایی شرکت ندهیم ولی تصور می‌کنم با این کار به دردسر خواهیم افتاد.

در مجموع قیدهای پایگاه بایستی *در محدوده عبارت* ارضا شوند (خیلی غیر رسمی: «قبل از سمیکالن»). ارضای قید یعنی با مقدارهای فعلی موجود در پایگاه، عبارت منطقی قید مقدار صحیح را برگرداند. با بیانی دیگر آنها بایستی در انتهای هر عبارتی که ممکن است آنها را زیرپا بگذارد، مورد ارزیابی قرار گیرند. اگر این بررسی با شکست مواجه شود، اجرای دستور خطاکار متوقف می‌شود و یک پیام خطا به نمایش در می‌آید.

آیا نباید برخی بررسی‌ها به تعویق بیافتند؟

همانطور که گفتم بر اساس باور عامه، بررسی قیدهای چند مرابطه‌ای بایستی -لااقل- تا زمان تکمیل به تعویق افتند (با وجود استدلال‌های بخش قبل). فرض کنید پایگاه توزیع‌کنندگان و قطعات تحت چنین قیدی قرار دارد:

```
CONSTRAINT C7
 COUNT ( ( S WHERE SNO = SNO('S1') ) { CITY }
   UNION
   ( P WHERE PNO = PNO('P1') ) { CITY } ) < 2 ;
```

این قید می‌گوید توزیع‌کننده S1 و قطعه P1 هرگز نباید از دو شهر مختلف باشند. موشکافی؛ اگر مرابطه‌های S و P به ترتیب دارای تاپل‌های S1 و P1 باشند آنگاه این تاپل‌ها بایستی در CITY دارای یک مقدار باشند (اگر چنین نباشد اجرای COUNT مقدار دو را بازمی‌گرداند). البته S می‌تواند فاقد S1 و P می‌تواند فاقد P2 (یا هر دو) باشد (در این صورت COUNT مقدار صفر یا یک بازمی‌گرداند). با مقدارهای همیشگی مثال خودمان، هر دو به‌روزرسانی‌های SQL زیر هنگام بررسی فوری با شکست مواجه می‌شوند:*

```
UPDATE S SET CITY = 'Paris' WHERE SNO = SNO('S1') ;
UPDATE P SET CITY = 'Paris' WHERE PNO = PNO('P1') ;
```

راه حل متعارف این مشکل، تعویق قیدهای جامعیت و بسته‌بندی دو به‌روزرسانی درون یک تراکنش است. مثل این:

* من عمدا در اینجا از SQL استفاده کرده‌ام چرا که در توتوریال‌دی بررسی فوری است و راه حل متعارفی که مورد بحث قرار داده‌ام در آنجا به کار نخواهد آمد.

```
BEGIN TRANSACTION ;
UPDATE S SET CITY = 'Paris' WHERE SNO = SNO('S1') ;
UPDATE P SET CITY = 'Paris' WHERE PNO = PNO('P1') ;
COMMIT ;
```

در راه حل متعارف، قیدها در زمان تکمیل یا commit بررسی می‌شوند. با این حال اگر تراکنش در بین دو به‌روزرسانی مورد سوال قرار گیرد که «آیا S1 و P1 از دو شهر متفاوت هستند؟» (با فرض اینکه S1 و P1 هر دو وجود دارند) پاسخ *آری* خواهد بود.

انتساب چندتایی

راه حل بهتر برای این مشکل، پشتیبانی از حالت چندتایی انتساب است که انجام هر تعداد انتساب منفرد را بصورت «همروند» امکان‌پذیر می‌کند. مثال (دوباره به *توتوریال‌دی* تغییر موضع می‌دهیم):

```
UPDATE S WHERE SNO = SNO('S1') ( CITY := 'Paris' ) ,
UPDATE P WHERE PNO = PNO('P1') ( CITY := 'Paris' ) ;
```

توضیح: اولا به ویرگول توجه کنید که نشان می‌دهد هر دو دستور بخش‌هایی از یک عبارت هستند. ثانیا به‌روزرسانی در واقع نوعی انتساب است و این «به‌روزرسانی دوتایی» مختصرنویسی است از حالت عمومی‌تر انتساب دوتایی به صورت زیر:

```
S := ... , P := ... ;
```

این انتساب دوتایی یک مقدار به S و یک مقدار دیگر به P منتسب می‌کند. بطور کلی معنای انتساب چندتایی چنین است:

- اول، تمامی عبارت‌های سمت راست محاسبه و ارزیابی می‌شوند.
- دوم، اجزا سمت چپ به ترتیبی که نوشته شده‌اند مقدار می‌گیرند.*

درک کنید، به این دلیل که قبل از انجام هر انتسابی تمام عبارات سمت راست ارزیابی می‌شوند، هیچ انتسابی به انتساب دیگر وابسته نمی‌شود. علاوه بر این از آنجا که انتساب چندتایی

* این تعریف برای موردی که دو یا چند مقدار مختلف بخواهند به یک مقصد منتسب شوند، نیاز به کمی تغییر دارد. جزئیات این مورد از بحث این کتاب خارج است.

یک دستور به حساب می‌آید، هیچ بررسی جامعیتی «در میان» انجام آن، صورت نمی‌گیرد. (این واقعیت اصلی‌ترین دلیل استفاده از انتساب چندتایی در این مورد است.) در این مثال اجرای انتساب دوتایی با موفقیت انجام می‌شود در حالی که اجرای دو انتساب مجزا با شکست روبرو می‌شود.

به خاطر داشته باشید که در این مثال تراکنش‌ها هیچ راهی برای رسیدن به حالت بی‌ثبات، بین دو به‌روزرسانی ندارند، چرا که تصور حالت «بین دو به‌روزرسانی» در اینجا بی‌معناست. و این را هم به خاطر بسپارید که دیگر هیچ نیازی به تعویق بررسی وجود ندارد.

SQL تا حدودی از انتساب چندتایی پشتیبانی می‌کند و در واقع سالهاست که این *تا حدودی* پشتیبانی را در خود دارد. اول، عملیات ارجاعی مانند CASCADE بر این موضوع دلالت دارد که یک حذف یا به‌روزرسانی می‌تواند موجب تغییرات متعددی در جدول‌های مختلف شود و این به معنای «یک مجموعه در یک لحظه» و با یک دستور است. دوم، قابلیت به‌روزرسانی (مثلا) یک ویو حاوی پیوند با همان استدلال مورد قبل. سوم، FETCH INTO و SELECT INTO که هر دو از قماش انتساب چندتایی هستند. چهارم، SQL:2003 انتساب چندتایی برای دستورات مجموعه‌ای را معرفی کرده است. و غیره (این فهرست شامل همه موارد نیست). با این وجود SQL هنوز از انتساب صریح مقدار به چند جدول مختلف پشتیبانی نمی‌کند (این امر به صورت مختصر و مفید در مثال قبل شرح داده شد).

و نکته آخر: خواهشمندم بفهمید که استفاده از انتساب چندتایی به معنای منسوخ شدن تراکنش‌ها نیست. تراکنش‌ها هنوز –حداقل– برای ترمیم و همزمانی مورد نیاز هستند. تمام چیزی که من می‌گویم این است که، برخلاف آنچه معمولا پنداشته می‌شود، تراکنش‌ها *واحد جامعیت* نیستند.

قیدها و گزاره‌نماها

از فصل 4 یادآوری می‌کنم که گزاره‌نمای یک مرابطه بسط داخلی آن مرابطه نامیده می‌شود. برای مثال گزاره‌نمای مرابطه S1 به این شکل است:

توزیع‌کننده SNO که طرف قرارداد است، نامش SNAME، رتبه‌اش STATUS و محل استقرارش CITY است.

در دنیای ایده‌آل این گزاره‌نما بعنوان معیاری برای پذیرش به‌روزرسانی‌های مرابطه S به شمار می‌رود بدین معنا که می‌گوید کدام INSERTها، DELETEها و UPDATEها قابل قبول هستند. ولی این هدف دست یافتنی نیست.

- یک دلیل آن است که سیستم نمی‌تواند معنای «توزیع‌کننده طرف قرارداد» و یا «مستقر» را درک کند و این امر در تفسیر واقعیت دارای اهمیت است. مثلا اگر شماره توزیع‌کننده S1 و نام شهر لندن با هم در یک تاپل دیده شوند، کاربر این واقعه را چنین تفسیر می‌کند که توزیع‌کننده S1 در شهر لندن مستقر است.* ولی سیستم به هیچ‌وجه نمی‌تواند چنین کاری را انجام دهد.
- دلیل دیگر آن است که حتی اگر سیستم بتواند معنای توزیع‌کننده طرف قرارداد و یا مستقر در فلان مکان را برای خود تفسیر کند، نمی‌تواند اصول موضوعه – که کاربر به سیستم می‌گوید درست هستند– را درک کند. وقتی کاربر در برابر سیستم ادعایی را مطرح می‌کند، مثلا هنگام به‌روزرسانی ادعا می‌کند که توزیع‌کننده S6 به نام لوپز با رتبه 20 در مادرید مستقر است، سیستم نمی‌تواند درست یا غلط بودن این ادعا را بفهمد. تنها کاری که از دست سیستم بر می‌آید این است که بررسی کند آیا این ادعا قیدهای جامعیت را زیرپا می‌گذارد یا خیر. فرض کنید که این اتفاق (زیرپا گذاشتن قیدها) نیفتد، در این صورت سیستم ادعای کاربر را می‌پذیرد و *آن را واقعیت می‌انگارد* (تا وقتی که کاربر با یک به‌روزرسانی دیگر به سیستم بگوید که این امر از این پس واقعیت ندارد).

قابل قبول بودن به‌روزرسانی در دنیای ایده‌آل یک چیز است و در دنیای واقعی یک چیز دیگر. ملاک پذیرش به‌روزرسانی در دنیای واقعی گزاره‌نما نیست، بلکه مجموعه‌ای از چند قید است که رعایت آنها موجب شباهت و نزدیکی به گزاره‌نما خواهد شد.† بنابراین:

---

* یا توزیع‌کننده S1 همواره در لندن مستقر بوده است، یا فقط یک دفتر در لندن دارد، یا دفتر در لندن ندارد، یا تعداد بی‌شماری حالات دیگر (متناظر با تعداد بی‌شماری گزاره‌نمای ممکن).

† به همین دلیل من در جاهای دیگر از اصطلاح *گزاره‌نمای داخلی* برای اشاره به قیدها و *گزاره‌نمای خارجی* برای اشاره به آنچه در اینجا گزاره‌نمای مرابطه نامیده شده است، استفاده کرده‌ام.

ثبات سیستم نمی‌تواند نشان دهنده درستی اطلاعات موجود در سیستم باشد.

به بیان دیگر سیستم نمی‌تواند اینکه پایگاه فقط محتوی گزاره‌های درست است را تضمین کند. چیزی که تضمین شده است این است که چیزی که قیدها را زیرپا بگذارد در پایگاه وجود ندارد (یعنی هیچ بی‌ثباتی و تناقضی وجود ندارد). متاسفانه واقعیت و ثبات دو چیز متفاوت‌اند. پس:

- اگر پایگاه فقط شامل گزاره‌های درست باشد آنگاه با ثبات هم خواهد بود ولی عکس این قضیه لزوما برقرار نیست.
- اگر پایگاه بی‌ثبات باشد دارای دست‌کم یک گزاره غلط خواهد بود ولی عکس این قضیه لزوما برقرار نیست.

خلاصه‌تر: درستی موجب ثبات می‌شود (ولی برعکس نه) و بی‌ثباتی موجب نادرستی می‌شود (باز هم برعکس نه). این که بگوییم پایگاه داده دارای ویژگی درستی است مانند این است که بگوییم این پایگاه امری از دنیای واقعی را به صورت صادقانه منعکس می‌کند. نه کمتر و نه بیشتر.

حال بیایید این نظریه را دقیق‌تر بررسی کنیم. فرض کنید R یک مرابطه پایه است (به زودی سراغ ویوها هم خواهم رفت) و C_1، C_2،..C_n تمامی قیدهای -تک و چند مرابطه‌ای- تعریف شده پایگاه بر روی R هستند. برای سادگی فرض کنید هر C_i فقط شامل یک عبارت بولی است (یعنی از نام قیدها صرفنظر می‌کنیم). آنگاه عبارت بولی:

(C_1) AND (C_2) AND ... AND (C_n) AND TRUE

قید کلی مرابطه R خواهد بود (ولی با توجه به اهداف این کتاب من فقط آن را *قید شناخته‌شده* R می‌نامم). به AND TRUE پایانی توجه کنید که معنایش این است که اگر هیچ قیدی وجود نداشته باشد مقدار پیش‌فرض صحیح خواهد بود.

فرض کنید قید کلی مرابطه R را RC نامیده‌ایم. بدیهی است R هرگز اجازه ندارد مقداری داشته باشد که RC غلط شود. این امر انگیزه‌ای است برای (نگارش اول) آنچه من مایلم آنرا *قاعده طلایی* بنامم:

هیچگاه،هیچ عمل به‌روزرسانی حق ندارد موجب غلط‌شدن هیچیک از قیودمرابطه‌ها شود.

حال فرض کنید که V یک ویو است. آنگاه V هم دارای یک «قید کلی مرابطه» است (که من برای سادگی فقط آن را *قید شناخته‌شده* V می‌نامم) و به روشی بدیهی از مرابطه‌هایی که

V بر روی آنها تعریف شده است، بدست می‌آید. برای مثال فرض کنید SC قید کلی مرابطه پایه S باشد آنگاه قید کلی ویو توزیع‌کنندگان لندنی LS چنین است:

(SC) AND (CITY = 'London')

حال فرض کنید که DB یک پایگاه داده است و DB شامل مرابطه‌های R_1، R_2،...R_n است (فقط). همچنین قیدهای این مرابطه‌ها عبارتند از RC_1، RC_2،...RC_n. آنگاه قید کلی پایگاه DB یا DBC (که با توجه به اهداف این کتاب من فقط آن را *قید شناخته‌شده* DB می‌نامم) برابر است با AND تمام قیدهای مرابطه:

(RC_1) AND (RC_2) AND ... AND (RC_n)

و در اینجا به نگارش توسعه‌یافته (در حقیقت نهایی) قاعده طلایی می‌رسیم:

هیچگاه، هیچ عمل به‌روزرسانی حق ندارد موجب غلط شدن هیچیک از قیود پایگاه شود.

خصوصا توجه داشته باشید که این قانون –طبق نظر من که بررسی تمام قیدهای جامعیت را فوری می‌دانم– عملگرهای به‌روزرسانی را مورد توجه قرار داده است نه تراکنش‌ها را.

حال می‌توانم به یک کار ناتمام بپردازم. گفتم که هرگز نمی‌توان به پاسخ‌هایی که از یک پایگاه بی‌ثبات دریافت می‌شود، اعتماد کرد. حال به اثبات آن می‌رسیم. همانطور که می‌دانیم پایگاه داده‌ها می‌تواند به صورت مجموعه‌ای از گزاره‌ها در نظر گرفته شود. فرض کنید که این مجموعه بی‌ثبات است. یعنی در حالی که p یک گزاره است، p و NOT p هر دو صحیح هستند. حال فرض کنید q یک گزاره دلخواه است. آنگاه:

- از درستی p نتیجه می‌گیریم p یا q صحیح است.
- از درستی p یا q و همچنین درستی NOT p نتیجه می‌گیریم q درست است.

ولی q دلخواه بود! بنابراین هر گزاره‌ای (حتی اگر آشکارا غلط باشد مانند 0=1) می‌تواند در یک سیستم بی‌ثبات، صحیح نشان داده شود.

نکات متفرقه

در این بخش می‌خواهم درباره جامعیت به مطالبی بپردازم که در هیچ‌کدام از مباحث قبلی جای نمی‌گیرند.

اول، از آنجا که قید عبارتی است که بایستی صحیح باشد از دیدگاه رسمی قید یک *گزاره* است. برای مثال قید C1 از بخش «قیدهای پایگاه»:

CONSTRAINT C1 IS_EMPTY (S WHERE STATUS < 1 OR STATUS > 100) ;

یک مقدار خاص برای مرابطه S در نظر بگیرید، عبارتی بولی:

IS_EMPTY (S WHERE STATUS < 1 OR STATUS > 100)

(که یک قید است) به ازای آن، بدون شک یا درست است یا غلط. این تعریف همان چیزی است که به آن گزاره می‌گوییم (فصل 4 را ببینید).

دوم، فرض کنید به هنگام برقرار کردن قید فوق، مرابطه S دارای تاپلی است که قید C1 را زیرپا می‌گذارد. در این صورت اجرا بایستی با شکست مواجه شود. بطور کلی زمانی که می‌خواهیم یک قید جدید برای پایگاه تعریف کنیم، سیستم بایستی بررسی کند که آیا این قید با توجه به محتویات فعلی پایگاه، ارضا می‌شود یا خیر. اگر خیر قید بایستی مردود شود و اگر آری قید بایستی پذیرفته شود و از این پس اعمال گردد.

سوم، از فصل یک یادآوری می‌کنم مدل رابطه‌ای دارای قیدی است که من آن را قید «ذاتی» جامعیت می‌نامم؛ همان قاعده جامعیت ارجاعی (من عمدا از قاعده جامعیت وجودی صرفنظر می‌کنم). ولی باید برای همه روشن شود که قاعده جامعیت ارجاعی با تمام قیدهایی که در این فصل بررسی شدند، تفاوت دارد و در واقع یک *ابرقید* است. برای مثال در پایگاهی که مرابطه‌های S، P و SP حضور دارند، بایستی قیدهای مشخصی (کلید خارجی) بین SP و S و یا بین SP و P برقرار باشند، چرا که در غیر این صورت ممکن است پایگاه ابرقید جامعیت ارجاعی را زیرپا بگذارد. همچنین پایگاهی که دارای مرابطه‌های EMP و DEPT است (فصل 1 را ببینید) بایستی دارای قید کلید خارجی باشد چرا که در اینجا هم ممکن است ابرقید جامعیت ارجاعی زیرپا گذاشته شود.

چهارم، نکته‌ای که هرگز آن را به صراحت نگفته‌ام ولی مطمئنم که بدیهی است، قیدها بایستی به صورت اعلانی تعریف شوند. استاندارد SQL هم پشتیبانی گسترده‌ای از قیدهای اعلانی دارد و این در حالی است که دست‌کم برخی از محصولات عمده SQL فاقد این پشتیبانی هستند. آنها فرض را بر آن گذاشته‌اند که شما برای اِعمال جامعیت از پروسیجرهای تریگردار –که به‌نام تریگرها شناخته شده‌اند– استفاده می‌کنید. (استاندارد پشتیبانی گسترده‌ای هم از تریگرها دارد.) همانطور که در فصل 1 شرح دادم روش‌های اعلانی (اگر در دسترس باشند) همواره بر روش‌های پروسیجری (روندگرا) برتری دارند. همچنین استفاده از قیدهای اعلانی مانند دری است که بر بهینه‌سازی معنایی گشوده می‌شود، در حالی که تریگرها چنین نیستند.

نکته دیگری که تاکنون بیان نکرده‌ام امکان پشتیبانی از *قیدهای گذار* است. قید گذار قیدی است که برای یک نوع متغیر (مثلا مرابطه) مجاز بودن گذار از یک مقدار به مقدار دیگر را تعیین می‌کند. برای مثال وضعیت تاهل افراد با وضعیت «تاکنون ازدواج نکرده» می‌تواند به «متاهل» تغییر یابد ولی عکس آن امکان‌پذیر نیست. در این مثال «رتبه هیچ توزیع‌کننده‌ای نمی‌تواند کم شود»:

```
CONSTRAINT C8 IS_EMPTY
  ( ( ( S' { SNO, STATUS } RENAME ( STATUS AS STATUS' ) )
    JOIN
    ( S { SNO, STATUS } ) )
  WHERE STATUS' > STATUS ) ;
```

توضیح: این قاعده را می‌پذیریم که نام مرابطه با پریم (مثل ’S) به *مرابطه تحت قید، قبل از انجام به‌روزرسانی* اشاره می‌کند. قید C8 می‌گوید: «اگر مقدار قبلی S و مقدار جدید آن را پیوند دهیم و فقط تاپل‌هایی که رتبه قبلی آنها از رتبه فعلی بزرگتر باشد را گزینش کنیم، نتیجه نهایی باید خالی باشد» (اگر پیوند بر روی SNO انجام شود، در نتیجه پیوند هر تاپلی که رتبه قبلی‌اش از رتبه فعلی بزرگتر است، نماینده توزیع‌کننده‌ای می‌باشد که تنزل کرده است.)

آخر، امیدوارم با توجه به مباحث این فصل با این حرف موافق باشید که قیدها *واجب و ضروری*‌اند و با این وجود محصولات تجاری پشتیبانی بسیار ضعیفی از آنها به عمل می‌آورند. اگر نخواهیم بگوییم که قیدها به درستی درک نشده‌اند، در بهترین حالت به نظر می‌رسد که آنها کمتر از آنچه شایسته آن هستند مورد توجه قرار گرفته‌اند. آنچه در دنیای تجاری مورد توجه است عبارت است از *کارایی*، *کارایی* و باز هم *کارایی*. بقیه چیزها مانند استفاده آسان، استقلال داده‌ای و خصوصا جامعیت قربانی اهداف مهم‌تر شده‌اند.[*] –و در بهترین حالت در اولویت کمتر قرار گرفته‌اند– ولی وقتی نتوانیم از صحت اطلاعات دریافتی مطمئن شویم کارایی به چه دردمان می‌خورد؟ بدون رودربایستی برای من مهم نیست، سیستمی که مطمئن نیستم پاسخ‌هایش به کوئری‌های من درست است، چقدر سرعت دارد.

---

[*] منظورم این نیست که پشتیبانی مناسب از جامعیت توجیهی برای کارایی بد است. من شدیدا به بهبود کارایی معتقدم و فقط منظورم این است که ضمن توجه به مسائل مربوط به کارایی نبایستی سایر مسائل مانند جامعیت را به دست فراموشی سپرد.

خلاصه

من به شرح دو بحث پایه در مورد قیدها پرداختم، قیدهای نوع و قیدهای پایگاه. یک قید نوع مجموعه مقدارهایی که آن نوع را تشکیل می‌دهند، را تعریف می‌کند. در توتوریال‌دی این قید جزئی از تعریف نوع مورد نظر است و به عنوان یک *نماممکن* (نمایش امکان‌پذیر) برای آن نوع شناخته می‌شود. نماممکن خود، یک *قید بدیهی* بر نوع تحمیل می‌کند (SQL به جز قید بدیهی از هیچ قیدی برای نوع پشتیبانی نمی‌کند). بررسی قیدهای نوع به عنوان بخشی از اجرای انتخابگر تلقی می‌شوند. به عنوان حاشیه به موشکافی انتخابگرها و عملگرهای _THE پرداختم، که هر دو به نحوی به نماممکن‌ها مربوط هستند.

قیدهای پایگاه مقدارهایی که می‌توانند در پایگاه ظاهر شوند را مقید می‌کنند (اگر فقط به یک مرابطه اعمال شوند بطور غیر رسمی قید مرابطه خوانده می‌شوند). آنها در توتوریال‌دی با CONSTRAINT و در SQL با CREATE ASSERTION تعریف می‌شوند و انتظار میرود «هنگام سمیکالن» مورد بررسی قرار گیرند (اگر چه در SQL ممکن است چنین نباشد). بررسی قیدها هنگام سمیکالن به معنای بررسی آنها در انتهای هر عبارتی است که ممکن است آنها را زیرپا بگذارد، یعنی هنگام انتساب رابطه‌ای، به شرطی که انتساب رابطه‌ای را فقط شامل عملگرهایی بدانیم که پایگاه را به‌روزرسانی می‌کنند. دلایلی آوردم برای رد این تصور عمومی که قیدهای چند مرابطه‌ای نبایستی تا زمان تکمیل مورد بررسی قرار گیرند (و در کنار آن *بهینه‌سازی معنایی* را بطور خلاصه شرح دادم). همچنین نظریه مهم *انتساب چندتایی* را توضیح دادم.

بعد، نشان دادم قید کلی مرابطه R، که می‌تواند معادل سیستمی گزاره‌نمای R تصور شود، معیار پذیرش به‌روزرسانی‌ها در R است. «ثبات به تنهایی نمی‌تواند سیستم را وادار به درستی کند.» *قاعده طلایی* می‌گوید:

هیچگاه، هیچ عمل به‌روزرسانی حق ندارد موجب غلط شدن هیچیک از قیدهای پایگاه شود.

بحث را با این موضوع به پایان رساندم که قیدها واجب و ضروری‌اند. از نظر من و در واقع، آنها همه چیز پایگاه داده هستند. دوباره می‌گویم برای من مهم نیست، سیستمی که مطمئن نیستم پاسخ‌هایش به کوئری‌های من درست است، چقدر سرعت دارد.

تمرین‌ها

1- اصطلاحات قید نوع و قید پایگاه را شرح دهید. این قیدها چه وقت بررسی می‌شوند؟ اگر این بررسی شکست بخورد چه اتفاقی می‌افتد؟

2- قاعده طلایی را شرح دهید.

3- از قید ویژگی، قید تاپل، قید مرابطه، قید شناخته‌شده پایگاه، قید شناخته‌شده مرابطه، قید تک‌مرابطه‌ای، قید چندمرابطه‌ای چه فهمیده‌اید؟

4- تفاوت بین نمایش‌های امکان‌پذیر و فیزیکی را شرح دهید.

5- با دقت شرح دهید: الف) انتخابگر چیست ب) عملگر _THE چیست.

6-فرض کنید شهرهای مجاز فقط لندن، پاریس، رم، آتن، اسلو، استکهلم، مادرید و آمستردام هستند. نوع CITY را به گونه‌ای تعریف کنید که این شرط را برآورده کند. آیا راهی وجود دارد که این قید را بدون داشتن نوع CITY اعمال کنیم. اگر آری این دو روش را با هم مقایسه کنید.

7- در همه جای این کتاب فرض کرده‌ام SNO یک نوع تعریف شده بوسیله کاربر است. تعریفی برای این نوع ارائه دهید. فرض کنید فقط شماره توزیع‌کنندگانی مجاز هستند که رشته کاراکتری به طول حداقل دو حرف باشند. قسمت اول حرف S و قسمت دوم عددی صحیح و ده‌دهی بین 1 تا 9999.

8- پاره خط، خط راستی است که در صفحه مسطح دو نقطه را به هم وصل می‌کند. نوعی برای آن تعریف کنید.

9- آیا می‌توان یک نوع را با دو نامممکن متفاوت تصور کرد؟ اگر یک نوع با چند نامممکن وجود داشته باشد آیا بایستی برای هر یک از آنها یک قید نوع تعریف شود؟

10- آیا می‌توانید نوعی را مثال بزنید که نامممکن‌های مختلف آن دارای تعداد متفاوتی جزء باشند؟

11- چه عملگرهایی می‌توانند موجب نقض قیدهای C1 تا C8 که در این فصل آمده‌اند، گردند.

12- قید C1 (برای نمونه) دارای ویژگی‌ای است که می‌تواند در یک تاپل داده شده به تنهایی بررسی شود. قید C4 (برای نمونه) چنین نیست. معنای این تفاوت را بطور رسمی شرح دهید. این تفاوت از دیدگاه علمی چه اهمیتی دارد؟ (اگر دارد)

13- آیا می‌توانید قید پایگاهی در توتوریال‌دی بنویسید که دقیقا معادل KEY{SNO} در مرابطه S باشد؟

14- به زبان SQL عبارتی برای قید C7 متن این فصل، بنویسید.

15- به زبان SQL عبارتی برای قید C8 متن این فصل، بنویسید.

16- به SQL و/یا توتوریال‌دی قیدهایی جهت پایگاه توزیع‌کنندگان و قطعات بنویسید که نیازهای زیر را برآورده سازند:

الف) تمامی قطعات قرمز رنگ بایستی سبک‌تر از 50 پوند باشند.

ب) هر توزیع‌کننده لندنی بایستی P2 را توزیع نماید.

ج) دو توزیع‌کننده نمی‌توانند در یک شهر مستقر باشند.

د) در هر زمان بیش از یک توزیع‌کننده نمی‌تواند در آتن مستقر باشد.

ه) حداقل یک توزیع‌کننده بایستی در لندن وجود داشته باشد.

و) حداقل یک قطعه قرمز بایستی وزنی کمتر از 50 پوند داشته باشد.

ز) متوسط رتبه توزیع‌کنندگان بایستی حداقل 10 باشد.

ح) تعداد هیچ سفارشی نمی‌تواند بیش از دو برابر متوسط تعداد موجود در سفارش‌ها باشد.

ط) هیچ توزیع‌کننده‌ای که دارای حداکثر رتبه است نمی‌تواند در شهری مستقر باشد که توزیع‌کننده‌ای با حداقل رتبه در آن شهر مستقر است.

ی) هر قطعه بایستی در شهری باشد که حداقل یک توزیع‌کننده در آن مستقر است.

ک) هر قطعه بایستی در شهری باشد که حداقل یک توزیع‌کننده آن قطعه در آن شهر مستقر است.

ل) قطعات توزیع‌کنندگان لندنی بایستی تنوع بیشتری از قطعات توزیع‌کنندگان پاریسی داشته باشند.

م) توزیع‌کنندگان لندنی بایستی در مجموع تعداد بیشتری از قطعات را نسبت به توزیع‌کنندگان پاریسی داشته باشند.

در هر مورد بگویید کدام عملگرها می‌توانند قید را زیرپا بگذارند.

17- در پانویس بخش «قیدها و گزاره‌نماها» گفتم اگر مقدارهای S1 و لندن در یک تاپل ظاهر شوند ممکن است به این صورت استنباط شود (از جمله تفسیرهای مختلف) که توزیع‌کننده S1 در لندن دفتر ندارد. این تفسیر خاص بسیار بعید است. چرا؟ (راهنمایی: فرض بسته بودن جهان را به یاد بیاورید.)

18- فرض کنید قاعده cascade delete در مورد توزیع‌کنندگان و سفارش‌ها برقرار نیست. عبارتی به توتوریال‌دی بنویسید که یک توزیع‌کننده خاص را به همراه تمامی سفارش‌های مربوط به او با یک دستور تکی پاک کند.

19- با استفاده از ساختار آزمایشی قیدهای گذار در بخش نکات متفرقه، برای اهداف زیر قید گذار بنویسید:

الف) توزیع‌کنندگان آتنی فقط می‌توانند به لندن یا پارس منتقل شوند و توزیع‌کنندگان لندنی فقط می‌توانند به پاریس منتقل شوند.

ب) مجموع تعداد سفارش‌های یک قطعه هرگز کم نمی‌شوند.

ج) جمع تعداد سفارش‌های یک قطعه خاص با یک دستور به‌روزرسانی تکی نمی‌تواند به کمتر از نصف مقدار جاری کاهش یابد. (در مورد معنای «دستور به‌روزرسانی تکی» در اینجا، چه فکر می‌کنید؟ چه اهمیتی دارد؟ آیا اصلا مهم است؟)

20- تفاوت ثبات و درستی را شرح دهید.

21- فکر می‌کنید این تعریف نوع در توتوریال‌دی صحیح است؟

```
TYPE TTT POSSREP { ... CONSTRAINT FALSE } ;
```

22- فکر می‌کنید این تعریف قید در توتوریال‌دی صحیح است؟

```
CONSTRAINT FFF FALSE ;
```

23- محصول SQLای که به آن دسترسی دارید را بررسی کنید. چه بهینه‌سازی معنایی توسط آن پشتیبانی می‌شود (اگر می‌شود)؟

24- چرا فکر می‌کنید SQL در پشتیبانی از قیدها شکست می‌خورد؟ این امر چه پی‌آمدهایی دارد؟

25- بحث این فصل بطور عام در مورد نوع‌ها و بطور خاص در مورد قیدهای نوع تلویحا این فرض را داشت که نوع‌ها الف) اسکالراند ب) تعریف شده بوسیله کاربر هستند. آیا این بحث به نوع‌های غیراسکالر و/یا تعریف شده در سیستم قابل تعمیم است؟

فصل هفتم

نظریه طراحی

پایگاه داده‌ها

رسیدن به هدف استقلال داده‌ای مستلزم آن است که طراحی‌های منطقی و فیزیکی پایگاه دارای نظم و ساختار متفاوتی باشند. طراحی منطقی در ارتباط با آن چیزی است که کاربر می‌بیند و طراحی فیزیکی می‌گوید که طراحی منطقی چگونه با وسیله ذخیره و بازیابی فیزیکی ارتباط برقرار می‌کند. عمده‌ترین موضوع این فصل درباره طراحی منطقی است و من تا اطلاع ثانوی از کلمه طراحی بجای طراحی منطقی استفاده خواهم کرد.

یک نکته که می‌خواهم هرچه زودتر به آن اشاره کنم از این قرار است. یادآوری می‌کنم قید کلی مرابطه R می‌تواند مشابه سیستمی گزاره‌نمای R باشد. گزاره‌نمای R تفسیر و یا معنای مرابطه R است. بنابراین قیدها و گزاره‌نماها با طراحی منطقی کاملا ارتباط دارند. طراحی –به طور خلاصه– فعالیتی است که در آن گزاره‌نماها با دقت هر چه بیشتر موشکافی می‌شوند و سپس به مرابطه‌ها و قیدها منتسب می‌شوند. البته این گزاره‌نماها به ناچار تا حدودی غیر رسمی‌اند (همان چیزی که برخی آن را «قواعد تجاری» می‌نامند). در مقابل مرابطه‌ها و قیدها لزوما رسمی هستند.

این امر نشان می‌دهد که چرا من چندان طرفدار مدل‌سازی شی/رابطه ER و سایر متدهای تصویری، نیستم. مشکل نمودار ER و تصاویر مشابه آن است که فقط می‌توانند برخی قیدهای خاص را نشان دهند و از ارائه یک نمایش کامل عاجزند. بنابراین اگر چه این نمودارها ممکن است برای روشن کردن طرح در سطح بالای تجرید، مناسب باشند ولی این فکر که چنین نمودارهایی هسته اولیه طراحی هستند، گمراه کننده و در مواردی خطرناک است. طراحی شامل مرابطه‌هایی است که در نمودار دیده می‌شوند *به همراه قیدهایی* که دیده نمی‌شوند.

نکته دیگری هست که بایستی آن را زودتر روشن کنم. از فصل 4 یادآوری می‌کنم که ویوها دقیقا مانند مرابطه‌های پایه، فرض و احساس می‌شوند (منظور این نیست که ویوها فقط برای مختصرنویسی تعریف می‌شوند، بلکه منظور این است که ویوها کاربر را از پایگاه داده «واقعی» به نحوی دور نگاه می‌دارند). بطور کلی کاربر با پایگاهی که فقط محتوی مرابطه‌های پایه است (پایگاه داده «واقعی») سروکار ندارد، بلکه با چیزی که *پایگاه داده کاربری* نامیده می‌شود و

محتوی مخلوطی از مرابطه‌های پایه و ویوهاست، سروکار دارد. از نظر کاربر، این پایگاه بایستی دقیقا مانند پایگاه داده واقعی باشد و تمامی اصول طراحی که در این فصل مورد بحث قرار می‌گیرند بایستی در مورد آن (نه فقط پایگاه واقعی) صادق باشند.

احساس می‌کنم که باید یک نکته دیگر را نیز به عنوان مقدمه شرح دهم. بعضی بازبین‌های کتاب به هنگام مطالعه این فصل تصور کرده‌اند که من قصد دارم اصول اولیه طراحی پایگاه را شرح دهم در حالی که چنین نیست. شما در زمینه پایگاه داده‌ها حرفه‌ای هستید و فرض بر آن است که با اصول طراحی آشنایی دارید. هدف این فصل تشریح عملی روند طراحی نیست، بلکه هدف این است که با شرح موضوع‌هایی که تاکنون با آنها سروکار نداشته‌اید و همچنین نگریستن به سایر موضوعات آشنا، از دیدگاهی جدید، به هنگام طراحی از موضع مستحکم‌تری برخورد کنید. من نمی‌خواهم وقت را زیادی صرف موضوعاتی کنم که با آنها آشنایی دارید پس برای مثال عمدا وارد جزئیات فرم دوم و سوم نرمال نمی‌شوم چرا که آنها جزء اطلاعات عمومی طراحی هستند که موشکافی آنها با طبیعت این کتاب سازگاری ندارد (در هر صورت، آنها خودشان هم مهم نیستند و فقط یک گام جلوتر یعنی فرم نرمال بویس کاد اهمیت دارد که در این فصل هم مورد بحث قرار خواهد گرفت).

جایگاه نظریه طراحی

نظریه طراحی هم جزئی از مدل رابطه‌ای نیست. بلکه نظریه مستقلی است که بر روی این مدل بنا شده است (می‌توان آن را بصورت سرپوشی برای مدل رابطه‌ای در نظر گرفت ولی تکرار می‌کنم جزء مدل نیست.)، با این وجود بخش‌های بنیادی مدل با آن در ارتباط‌اند مثلا عملگرهای پرتو و پیوند که جزء مدل رابطه‌ای *هستند*.

و یک نکته دیگر: نظریه طراحی که در مورد آن صحبت می‌کنم واقعا به شما نمی‌گوید چطور طراحی کنید! بلکه فقط به شما می‌گوید اگر پایگاه را به شیوه نامتعارف طراحی کنید، چه مشکلاتی پیش خواهد آمد. برای مثال پایگاه توزیع‌کنندگان و قطعات را در نظر بگیرید. طراحی متعارف همان است که در همه جای این کتاب در نظر گرفته‌ام. منظورم از «متعارف» این است که سه جدول مورد نیاز است، ویژگی STATUS متعلق به مرابطه S، ویژگی COLOR متعلق به مرابطه P و ویژگی QTY متعلق به مرابطه SP است و غیره. ولی چرا فقط اینها متعارف‌اند؟

تصور کنید که ما می‌خواهیم طراحی دیگری داشته باشیم، مثلا ویژگی STATUS را از مرابطه S به مرابطه SP منتقل می‌کنیم (به صورت احساسی به نظر می‌رسد که جای درستی برای آن نیست چرا که رتبه به توزیع‌کننده مربوط است نه به سفارش). شکل 7-1 مقدارهای نمونه این مرابطه اصلاح شده را نشان می‌دهد (که برای جلوگیری از اشتباه آن را STP نامیده‌ام).

STP

SNO	STATUS	PNO	QTY
S1	20	P1	300
S1	20	P2	200
S1	20	P3	400
S1	20	P4	200
S1	20	P5	100
S1	20	P6	100
S2	10	P1	300
S2	10	P2	400
S3	30	P2	200
S4	20	P2	200
S4	20	P4	300
S4	20	P5	400

شکل 7-1. مرابطه STP با مقدارهای نمونه

یک نگاه اجمالی به شکل کافیست تا بفهمید که طراحی چه اشکالی دارد؛ این مرابطه دارای *افزونگی* است، هر تاپل دارای S1 به ما می‌گوید S1 دارای رتبه 20 است، هر تاپل دارای S2 می‌گوید S2دارای رتبه 10 است و غیره. نظریه طراحی هم به ما می‌گوید این پایگاه به شیوه متعارف طراحی نشده است و در نهایت موجب افزونگی می‌شود و ما را دچار پی‌آمدهای افزونگی خواهد کرد. به بیان دیگر نظریه طراحی اساسا در صدد کاهش افزونگی است، همان‌طور که به زودی خواهیم دید. به همین دلیل نظریه طراحی با *مجموعه‌ای خوب از مثال‌های بد* شناسایی می‌شود. انتقادهایی در مورد ذاتی بودن طراحی وجود دارد که من در بخش بعد به آنها خواهم پرداخت.

جهت درک بهتر موضوع، نظریه طراحی می‌تواند محصول طراحی شده را مورد بازبینی قراردهد تا مطمئن شود هیچ کدام از اصول رسمی طراحی را زیرپا نمی‌گذارد. و بازهم... واقعیت متاثر کننده این است که اصول رسمی طراحی فقط بخش علمی فرآیند طراحی را شامل می‌شوند و موضوعات بی‌شماری هستند که در این مبحث پوشش داده نمی‌شوند. *طبیعت کار طراحی*

پایگاه هنوز هم یک فن فردی به شمار می‌رود. فن طراحی بیشتر شبیه یک هنر است و اصول رسمی که در این فصل توضیح داده می‌شوند فقط بخش کوچکی از این دانش هستند.

حال می‌خواهم وارد بخش علمی طراحی شوم. بطور خاص دو موضوع گسترده را بررسی خواهیم کرد، *نرمال‌سازی* و *تعامد (تفکیک)*. تصور می‌کنم که دست کم در مورد اولی اطلاعاتی داشته باشید. فکر می‌کنم شما می‌دانید که:

- چندین *فرم نرمال* وجود دارد (اول، دوم، سوم و...)
- صحبت تقریبی، اگر مرابطه R در فرم نرمال n+1 باشد آنگاه قطعا در فرم نرمال n هم هست.
- ممکن است یک مرابطه در nمین فرم نرمال باشد و در (n+1)مین فرم نرمال نباشد.
- از دیدگاه طراحی، هر چه فرم نرمال بالاتر باشد بهتر است.
- این ایده با *وابستگی* ارتباط مستقیم دارد (این کلمه فقط نام دیگری است برای قید جامعیت).

می‌خواهم آخرین نکته را موشکافی کنم. گفتم که بطور کلی روند طراحی ارتباط زیادی با قیدها دارد. نوع خاصی از قید در اینجا وابستگی نامیده می‌شود ولی وابستگی دارای یک ویژگی خاص است که قیدهای عمومی فاقد آن هستند. من در اینجا نمی‌توانم خیلی عمیق وارد این بحث شوم اما نکته اصلی آن است که می‌توان برای هر وابستگی *قاعده استنتاج* تعریف کرد و وجود قاعده استنتاج راه را برای تشریح نظریه طراحی هموار می‌کند.

تکرار می‌کنم، فرض بر آن است که شما در این مورد چیزهایی می‌دانید و قصد من این است که بر موضوعاتی که احتمالا نمی‌دانید، تمرکز کنم. من نکات مهم را برجسته خواهم کرد و از بقیه به سرعت خواهم گذشت و به طور کلی می‌خواهم از دیدگاهی به این موضوع نگاه کنم که تا حدودی با آنچه تاکنون دیده‌اید تفاوت دارد.

وابستگی تابعی و فرم نرمال بویس کاد

بدیهی است که فرم دوم نرمال 2NF، فرم سوم نرمال 3NF و فرم نرمال بویس کاد BCNF با *وابستگی تابعی* در ارتباطند. یک تعریف مختصر و دقیق:

تعریف: فرض کنید A و B دو زیرمجموعه از عنوان R هستند. آنگاه مرابطه R دارای FD (وابستگی تابعی) $A \rightarrow B$ است اگر و تنها اگر برای هر مقدار مجاز R، اگر در دو تاپل در A دارای مقدار یکسان باشند، در B نیز حتما مقداری یکسان خواهند داشت.

وابستگی $A \rightarrow B$ ، «B وابسته تابعی به A» و یا «A دترمینان B» یا به راحتی «A فلش B» خوانده می‌شود.

برای مثال فرض کنید این قید برقرار است که اگر دو توزیع‌کننده همشهری باشند رتبه آنها یکسان خواهد بود (شکل 7-2 را ببینید که در آن من رتبه توزیع‌کننده S2 را از 10 به 30 تغییر داده‌ام تا با قید جدید هماهنگی داشته باشد). آنگاه FD :

{ CITY } → { STATUS }

با حالت جدید مرابطه S – که آن را RS می‌نامیم- مطابقت دارد. به آکولادها توجه کنید. من از آکولاد استفاده کردم تا نشان دهم هر دو طرف FD مجموعه‌ای از ویژگی‌ها هستند حتی اگر مجموعه مورد نظر فقط یک عضو داشته باشد (مانند همین مثال).

RS

SNO	SNAME	STATUS	CITY	
S1	Smith	20	London	
S2	Jones	30	Paris	← note the change
S3	Blake	30	Paris	
S4	Clark	20	London	
S5	Adams	30	Athens	

شکل 7-2. RS مرابطه توزیع‌کنندگان به صورت تغییر یافته با مقدارهای نمونه

همانطور که مثال نشان داد، این که در مرابطه R یک FD برقرار باشد مستلزم آن است که یک قید پایگاه -و یا دقیق‌تر یک قید تک مرابطه‌ای روی R- در آن برقرار باشد. به عنوان مثال FD فوق معادل قید زیر به زبان *توتوریال‌دی* است:

```
CONSTRAINT RSC COUNT ( RS { CITY } ) =
           COUNT ( RS { CITY, STATUS } ) ;
```

یک مطلب سودمند که باید به خاطر سپرده شود: اگر R وابستگی $A \rightarrow B$ را برآورده نماید آنگاه وابستگی $A' \rightarrow B'$ را هم برآورده خواهد کرد اگر A' مجموعه مادر A و B' زیرمجموعه B باشد. به بیان دیگر همیشه می‌توانید به سمت چپ ویژگی‌هایی اضافه کنید و یا از سمت راست ویژگی‌هایی حذف کنید، و همچنان وابستگی تابعی برقرار بماند.

در اینجا لازم است دو اصطلاح را توضیح دهم. اول *فراکلید*. فراکلید اساسا یک مجموعه مادر برای کلید است (نه لزوما مجموعه مادر محض). بنابراین (طبق تعریف *کلید* از فصل 4) زیرمجموعه‌ای از عنوان مرابطه R به نام SK فراکلید خواهد بود اگر و تنها اگر خاصیت یکتایی را داشته باشد و معلوم نباشد که خاصیت کاهش‌ناپذیری را هم دارد یا نه. پس هر کلیدی فراکلید هم هست ولی اکثر فراکلیدها، کلید نیستند. مثلا {SNO,CITY} برای مرابطه S یک فراکلید است ولی کلید نیست. به خصوص توجه داشته باشید که عنوان مرابطه R همواره فراکلید R است.

توجه

مهم: اگر SK فراکلید و A زیرمجموعه‌ای دلخواه از R باشد، وابستگی $SK \rightarrow A$ حتما در R وجود دارد چرا که اگر دو تاپل دارای مقدار یکسانی در SK باشند آنگاه طبق تعریف دو تاپل یکسان هستند و طبیعتاً در A هم مقدار یکسانی دارند. (در فصل 4 این نکته را شرح ندادم، ولی در آنجا من درباره کلید صحبت می‌کردم نه فراکلید.)

اصطلاح جدید دیگر وابستگی *بی‌اهمیت* است. اساسا وابستگی در صورتی بی‌اهمیت است که هیچ راهی برای زیرپا گذاشتن آن وجود نداشته باشد. در هر مرابطه‌ای که ویژگی‌هایی با نام‌های SNO، STATUS و CITY وجود داشته باشد، فارغ از مقدارهای موجود، وابستگی‌های بی‌اهمیت زیر برقرارند:

{ CITY, STATUS } → { CITY }
{ SNO, CITY } → { CITY }
{ CITY } → { CITY }
{ SNO } → { SNO }

در مورد اول، اگر دو تاپل دارای مقدار یکسان در CITY و STATUS باشند، مسلما در CITY دارای مقدار یکسانی خواهند بود. در واقع یک وابستگی بی‌اهمیت است اگر و تنها اگر سمت چپ، مجموعه مادر سمت راست باشد (مجددا، نه لزوما مجموعه مادر محض). البته ما معمولا به هنگام طراحی پایگاه به فکر وابستگی‌های بی‌اهمیت نیستیم، به این دلیل که آنها بی‌اهمیت‌اند. ولی وقتی می‌خواهیم بطور رسمی و دقیق مسائل را بررسی کنیم، بایستی *تمامی* وابستگی‌ها را در نظر بگیریم، چه بااهمیت و چه بی‌اهمیت.

حال که ماهیت وابستگی تابعی مورد بررسی قرار گرفت، می‌توانم به ارائه فرم نرمال بویس کاد BCNF بپردازم. این فرم که فرم شناخته‌شده نرمال هم نامیده می‌شود و در ارتباط با وابستگی‌هاست به این صورت تعریف می‌شود:

تعریف: مرابطه R، BCNF است اگر و تنها اگر برای تمام وابستگی‌های بااهمیت موجود در R که به شکل $A \rightarrow B$ هستند، A فراکلید R باشد.

به بیان دیگر در یک رابطه BCNF، وابستگی‌های موجود یا بی‌اهمیت‌اند و یا «فلش آنها از فراکلید خارج شده است» (از هیچیک از این دو حالت راه گریز و خلاصی وجود ندارد). ممکن است برخی در تعریف آن بگویند: *همه چیز به کلید بستگی دارد، وجود یک کلید سراسری و عدم وجود هیچ چیز غیر از آن کلید.* من بایستی در ادامه این تعریف غیر رسمی فوری بگویم، این تعریف ظاهرا خوش‌آیند در واقع دقیق نیست چرا که به غیر از خیلی چیزها، فرض را بر آن گذاشته شده است که فقط یک کلید وجود دارد.

لازم می‌بینم پاراگراف قبل را کمی موشکافی کنم. بعد از این که گفتم از یک وابستگی «گریزی نیست»، ترسیدم که کمی به خطا رفته باشم. برای مثال توزیع‌کنندگان تغییر یافته RS در شکل 7-2 دارای وابستگی {SNO}→{STATUS} می‌باشد ولی اگر آن را به دو مرابطه SNC و CS تجزیه کنیم – کاری که چند لحظه دیگر انجام خواهیم داد- این وابستگی «ناپدید» می‌شود و به نحوی «راه گریز پیدا شده است». (تجزیه به این صورت که ویژگی‌های SNO، SNAME و CITY در SNC و ویژگی‌های CITY و STATUS در CS قرار گیرند.) ولی ناپدید شدن وابستگی چه معنایی دارد؟ چیزی که اتفاق افتاده این است که وابستگی به یک قید چند مرابطه‌ای (قیدی که چند مرابطه را درگیر می‌کند) تبدیل شده است.* این قید هنوز وجود دارد ولی دیگر یک وابستگی نیست. این مساله در مورد تمامی «گریزی نیست و خلاصی نیست»های این فصل صادق است.

در نهایت، فرض را بر آن گذاشته‌ام که می‌دانید قواعد نرمال‌سازی می‌گویند اگر مرابطه R، BCNF نباشد بایستی به مرابطه‌های کوچکتری (با ویژگی‌های کمتری) تبدیل شود که BCNF هستند. برای مثال:

---

* در این مثال یک قید ضمنی، که از ترکیب قید KEY{SNO} در SNC، قید KEY{CITY} در CS و قید کلید خارجی SNC و CS بدست می‌آید.

- مرابطه STP (در شکل 7-1) دارای وابستگی {STATUS}→{SNO} است که نه این وابستگی بی‌اهمیت است و نه «فلش از فراکلید خارج شده است» چرا که {SNO} فراکلید STP نیست. پس این مرابطه BCNF نیست (و همان‌گونه که به قبلا دیدیم دچار افزونگی است). ما آن را به دو مرابطه SP و SS تجزیه می‌کنیم به گونه‌ای که SP صاحب ویژگی‌های SNO، PNO و QTY (مطابق معمول) و SS صاحب ویژگی‌های SNO و STATUS خواهد شد. (به عنوان یک تمرین مقدار مرابطه‌های SP و SS را بر مبنای شکل 7-1 پیدا کنید. خود را قانع کنید که SP و SS، BCNF هستند و افزونگی از میان رفته است.)
- به همین طریق، مرابطه RS (در شکل 7-2) دارای وابستگی {STATUS}→{CITY} است و بایستی به دو مرابطه SNC با ویژگی‌های SNO، SNAME و CITY و SS با ویژگی‌های CITY و STATUS تجزیه شود. (به عنوان یک تمرین مقدار مرابطه‌های SNC و CS را بر مبنای شکل 7-2 پیدا کنید. خود را قانع کنید که SNC و CS، BCNF هستند و افزونگی از میان رفته است.)

تجزیه بی‌کم‌وکاست

می‌دانیم اگر را مرابطه‌ای BCNF نباشد بایستی آن را به مرابطه‌های کوچکتری که BCNF هستند تجزیه کنیم. البته مهم است که چنین تجزیه‌ای بایستی بی‌کم‌وکاست باشد: باید بدون از دست دادن اطلاعات، بتوانیم به جایی که از آن آمده‌ایم برسیم. یک‌بار دیگر RS را با وابستگی {STATUS}→{CITY}، در نظر بگیرید (شکل 7-2). فرض کنید که ما آن را تجزیه کرده‌ایم ولی نه به SNC و CS بلکه به SNS و CS همانطور که در شکل 7-3 نشان داده شده است. (مرابطه CS در هر دو تجزیه یکسان است ولی مرابطه SNS بجای ویژگی‌های SNO، SNAME و CITY دارای ویژگی‌های SNO، SNAME و STATUS است.) امیدوارم روشن باشد که اولا SNS و CS هر دو BCNF هستند، ثانیا تجزیه بی‌کم‌وکاست نیست و «کم‌وکاست‌دار» است. برای مثال نمی‌توان فهمید که محل توزیع‌کننده S2 پاریس است یا آتن. اطلاعات در اینجا از دست رفته‌اند.

SNS

SNO	SNAME	STATUS
S1	Smith	20
S2	Jones	30
S3	Blake	30
S4	Clark	20
S5	Adams	30

CS

CITY	STATUS
London	20
Paris	30
Athens	30

شکل 7-3. مرابطه‌های SNS و CS با مقدارهای نمونه

چه چیزی موجب می‌شود که برخی تجزیه‌ها بی‌کم‌وکاست و برخی دیگر کم‌وکاست دار شوند؟ توجه داشته باشید که *فرآیند تجزیه، همان فرآیند پرتو کردن است*. در همه مثال‌ها، مرابطه‌های «کوچکتر» از پرتو مرابطه اصلی بدست آمده‌اند. به بیان دیگر عملگر تجزیه همان عملگر پرتو جبر رابطه‌ای است.

توجه

دوباره شلختگی کردم. مانند همه عملگرهای جبر رابطه‌ای، پرتو هم به رابطه‌ها اعمال می‌شود نه به مرابطه‌ها. در حالی که معمولا چنین گفته می‌شود که مرابطه CS پرتوی از مرابطه RS است، یعنی *در هر لحظه از زمان رابطه‌ای که مساوی مقدار مرابطه RS است، پرتوی است از رابطه‌ای که مقدار آن مساوی مرابطه CS در همان لحظه از زمان*. امیدوارم فهمیده باشید!

وقتی می‌گوییم یک تجزیه بی‌کم‌وکاست است منظور این است که *اگر پرتوها را دوباره به هم پیوند دهیم، مجددا به همان مرابطه اصلی خواهیم رسید*.توجه داشته باشید که، با مراجعه به شکل 7-3، مرابطه RS با پیوند پرتوهای SNS و CS مساوی *نیست* و به همین دلیل است که این تجزیه کم‌وکاست‌دار است. در مقابل با مراجعه به شکل 7-2 مشاهده می‌شود که RS با پیوند SNC و CS مساوی *است* و این تجزیه بی‌کم‌وکاست است.

تکرار می‌کنم عملگر تجزیه، پرتو و عملگر ترکیب مجدد، پیوند است. و سوال رسمی که در هسته اصلی نرمال‌سازی وجود دارد این است:

فرض کنید که R یک مرابطه باشد و R_1، R_2،...R_n پرتوهایی از آن باشند، چه شرایطی بایستی برقرار باشد تا پیوند تمامی پرتوها مساوی R شود؟

پاسخ این سوال مهم را، یان هیث* هر چند ناقص، در سال 1971 و با ارائه قضیه زیر داد:

فرض کنید *A*، *B* و *C* زیرمجموعه‌هایی از عنوان مرابطه *R* هستند به گونه‌ای که اجتماع (مجموعه‌ای) *A*، *B* و *C* برابر کل عنوان *R* می‌شود. فرض کنید *AB* اجتماع (مجموعه‌ای) *A* و *B* است و همین امر در مورد *AC* صادق است. اگر وابستگی *A*→*B* در *R* برقرار باشد آنگاه *R* مساوی پیوند *AB* و *AC* با یکدیگر خواهد بود.

برای مثال مرابطه RS را یک‌بار دیگر در نظر بگیرید (شکل 7-2). این مرابطه وابستگی {CITY}→{STATUS} را دارد. پس می‌توان *A* را {CITY}، *B* را {STATUS} و *C* را {SNO,SNAME} در نظر گرفت. قضیه هیث می‌گوید RS ممکن است در یک تجزیه بی‌کم‌وکاست به پرتوهای {CITY,STATUS} و {CITY,SNO,SNAME} تبدیل شود، همانطور که قبلا هم می‌دانستیم.

توجه

ممکن است تعجب کرده باشید که چرا گفتم قضیه هیث یک پاسخ ناقص به این مساله بود، اجازه دهید تا این امر را با توجه به مثال قبل توضیح دهم. اساسا این نظریه به ما می‌گوید که تجزیه شکل 7-2 بی‌کم‌وکاست است ولی نمی‌گوید که تجزیه شکل 7-3 کم‌وکاست‌دار است. این شرط برای تجزیه بی‌کم‌وکاست *لازم* هست ولی *کافی* نیست. (شکل قوی‌تر قضیه هیث، هر دو شرط لازم و کافی را تامین می‌کند و توسط ران فاگین در سال 1977 ارائه شد، ولی جزئیات بیشتر از محدوده این بحث خارج است. تمرین 18 پایان فصل را ببینید.)

به عنوان یک حاشیه، در همان مقاله‌ای که در آن هیث قضیه خود را در آن منتشر کرد، تعریفی نیز از چیزی ارائه کرد که آن را فرم «سوم» نرمال نامید و در حقیقت همان BCNF بود. از آنجا که این تعریف سه سال قبل از تعریف بویس و کاد ارائه شد، به نظر من جا داشت که BCNF فرم نرمال هیث نامیده شود، در حالی که اینطور نشد.

نکته آخر: نتیجه این بحث آن است که قیدی که پیش از این برای مرابطه RS نشان دادم:

```
CONSTRAINT RSC COUNT ( RS { CITY } ) =
           COUNT ( RS { CITY, STATUS } ) ;
```

---

* بر وزن خیس. مترجم

می‌تواند به این صورت نیز نشان داده شود:

```
CONSTRAINT RSC
  RS = JOIN { RS { SNO, SNAME, CITY }, RS { CITY, STATUS } } ;
```

«در هر زمان، مرابطه RS با پیوند {CITY,STATUS} و {CITY,SNO,SNAME} برابر است. من اینجا از نوع پیشوندی پیوند استفاده کرده‌ام.

آیا این فقط یک احساس ذاتی است؟

همانطور که قبلا هم گفتم انتقادهایی که به نرمال‌سازی وارد می‌شوند، بر این مبنا که نرمال‌سازی فقط یک احساس ذاتی است. برای مثال مرابطه STP را دوباره در نظر بگیرید (شکل 1-7). این مرابطه آشکارا بد طراحی شده است، افزونگی در آن آشکار است و پی‌آمدهای آن هم بدیهی است، هر کس که با طراحی آشنایی داشته باشد بطور «طبیعی» این مرابطه را به دو پرتو SP و SS تجزیه می‌کند، حتی اگر هیچگونه آشنایی با BCNF نداشته باشد. ولی «طبیعی» در اینجا یعنی‌چه؟ برای طراحی «طبیعی» چه *اصولی* باید رعایت شوند؟

پاسخ این است: این اصول دقیقا همان قواعد نرمال‌سازی هستند. طراحان باتجربه آنها را در مغز خود دارند حتی اگر آنها را بطور رسمی نیاموخته باشند و نامشان را هم ندانند. بله. این اصول همان احساسات ذاتی هستند ولی بایستی بتوان آنها را به صورت رسمی بیان کرد. (احساس ذاتی را می‌توان به کار بست ولی توضیح دقیق آن آسان نیست!) کاری که نظریه نرمال‌سازی انجام می‌دهد این است که به روشی مختصر و مفید این احساس ذاتی را بیان می‌نماید. به نظر من کار بزرگی که نرمال‌سازی انجام می‌دهد این است: رسمیت بخشیدن به قواعد ذاتی، که راه را بر مکانیزه کردن اصول باز کرده است (با وارد شدن به ابزارهای طراحی ماشینی). مخالفان نرمال‌سازی معمولا این نکته‌ای را فراموش می‌کنند، آنها، به درستی ادعا می‌کنند که این ایده‌ها ذاتی هستند ولی درک نمی‌کنند توانایی تشریح احساسات ذاتی به صورت دقیق و رسمی پیروزی بزرگی است.

1NF، 2NF و 3NF

فرم‌های نرمال پایین‌تر از BCNF غالبا از نظر تاریخی مورد توجه هستند. همان‌طور که در مقدمه گفته شد، من قصد ندارم با تعریف آنها در اینجا وقت را هدر دهم فقط این را می‌گویم که تمام مرابطه‌ها دست‌کم 1NF هستند*، حتی آنهایی که دارای ویژگی‌هایی با مقدار رابطه‌اند(RVA). از نقطه نظر طراحی مرابطه‌های دارای RVA معمولا –اما نه همیشه– مقرون به صرفه نیستند. البته این بدین معنا نیست که هیچ وقت نباید از RVA استفاده کنیم (نتایج کوئری‌هایی که دارای RVA اند، هیچ مشکلی ندارند). این فقط به این معناست که معمولا نمی‌خواهیم RVAها وارد طراحی پایگاه داده‌ها شوند (و ما همیشه آنها را از بین می‌بریم و این را مدیون عملگر UNGROUP رابطه‌ای هستیم). من قصد ندارم در اینجا وارد جزئیات شوم و فقط می‌خواهم مرابطه‌های دارای RVA بسیار شبیه سیستم‌های قدیمی سلسله‌مراتبی مانند IMS به نظر می‌رسند. با استفاده از آنها مشکلات مربوط به سیستم‌های سلسله‌مراتبی دوباره سر بلند می‌کنند. در اینجا با هدف کامل بودن کتاب به عنوان مرجع، برخی از این مشکلات را فهرست می‌کنم:

- مشکل بنیادی آن است که سلسله‌مراتبی‌ها نامتقارن هستند. از این رو ممکن است برخی کارها به آسانی و برخی کارهای دیگر به سختی انجام گیرند.(تمرین‌های 27، 29 و 30 در انتهای فصل 5 این نکته را روشن می‌کنند.)
- در نتیجه کوئری‌ها نامتقارن هستند. برخی از آنها از هم‌ردیف‌های خود پیچیده‌تر هستند.
- این مشکل در مورد قیدهای جامعیت هم وجود دارد.
- این مشکل در مورد به‌روزرسانی‌ها هم وجود دارد، ولی حادتر.
- هیچ توصیه‌ای در این مورد که بهترین سلسله‌مراتب چگونه است، وجود ندارد.
- حتی ساختارهایی مانند چارت سازمانی که «طبیعتی» سلسله‌مراتبی دارند، با طراحی‌های غیرسلسله‌مراتبی بهتر پیاده سازی می‌شوند.

تکرار می‌کنم، استفاده از سلسله‌مراتبی‌ها بسته به موقعیت می‌تواند قابل قبول باشد، حتی در مرابطه‌های پایه. تمرین 14 را ببینید.

---

* در فصل‌های قبلی گفتم که رابطه‌ها –نه مرابطه‌ها– همیشه 1NF هستند. ولی اگر این امر را به مرابطه‌ها تعمیم دهم مشکلی پیش نخواهد آمد.

وابستگی پیوندی و فرم پنجم نرمال

فرم پنجم نرمال یا 5NF -که آن را در این بخش توضیح می‌دهم- «فرم نهایی نرمال» است. در واقع BCNF فرم *شناخته‌شده* نرمال با توجه به وابستگی تابعی و فرم پنجم نرمال، فرم *شناخته‌شده* نرمال با توجه به وابستگی *پیوندی* است:

تعریف: فرض کنید A، B،...، Z زیرمجموعه‌هایی از عنوان مرابطه R هستند. آنگاه R دارای وابستگی پیوندی یا JD زیر است:

☆{ $A, B,...Z$ }

اگر و تنها اگر هر رابطه‌ای که مقدار آن برای R مجاز محسوب می‌شود، مساوی پیوند پرتوهای A، B،...، Z آن باشد.

وابستگی پیوندی ☆$\{A,B,...,Z\}$ به صورت «ستاره (یا استار) A، B،...، Z» خوانده می‌شود. نکته‌هایی که از این تعریف حاصل می‌شوند:

- ضروری است R قابل تجزیه بی‌کم‌وکاست به A، B،...، Z باشد، اگر و تنها اگر بخواهد وابستگی پیوندی ☆$\{A,B,...,Z\}$ را داشته باشد.
- همچنین لازم است هر FD، JD هم باشد (همانطور که از قسمت قبل به یاد داریم) اگر R دارای یک FD خاص باشد، آنگاه می‌تواند به صورت بی‌کم‌وکاست به پرتوهای خاص تجزیه شود (به بیان دیگر دارای یک JD خاص است).

به عنوان یک مثال از نکته آخر، یک بار دیگر مرابطه RS را در نظر بگیرید (شکل 7-2). این مرابطه دارای وابستگی {CITY}→{STATUS} است و از این رو می‌تواند بی‌کم‌وکاست به پرتوهای SNC (SNO، SNAME و CITY) و CS (CITY و STATUS) تجزیه شود. در نتیجه وابستگی پیوندی ☆{SNC,CS} در مرابطه RS وجود دارد، البته اگر موقتا زیرمجموعه‌های مورد نظر از عنوان را SNC و CS بنامیم.

در فصل قبل دیدیم که همیشه «فلش‌ها از فراکلیدها خارج می‌شوند». این یک وابستگی تابعی است که هیچ گریزی از آنها وجود ندارد. بطور کلی‌تر از آن دسته وابستگی‌های پیوندی که بوسیله فراکلیدها اعمال می‌شوند هرگز نمی‌توان رهایی یافت. بطور خاص وابستگی پیوندی ☆$\{A,B,...,Z\}$ بوسیله فراکلیدها ایجاد شده است اگر و تنها اگر هریک از A، B،...، Z فراکلیدی برای مرابطه R باشند. برای مثال رابطه S خودمان را در نظر بگیرید. واقعیت این است

که {SNO} برای این رابطه یک فراکلید (در حقیقت کلید) است و این امر و موارد دیگری موجب می‌شود که وابستگی پیوندیJD زیر در آن برقرار باشد:

☆{ SN,SS,SC }

که SN، {SNO,SNAME} و SS، {SNO,STATUS} و SC، {SNO,CITY}

می‌باشد (توجه داشته باشید که هر کدام از اینها برای S فراکلید به حساب می‌آیند). و این گفته قطعا صحیح خواهد بود که، اگر بخواهیم، می‌توانیم S را به صورت بی‌کم‌وکاست به پرتوهای SN، SS و SC تجزیه کنیم. اما اینکه آیا این پرتوها همان چیزی است که مورد نظرمان است یا نه، مساله دیگری است.

همچنین در بخش قبل دیدیم که برخی FDها بی‌اهمیت هستند. همانطور که انتظار دارید برخی JDها هم بی‌اهمیت‌اند. وابستگی پیوندی ☆{*A*,*B*,...,*Z*} بی‌اهمیت است تنها اگر یکی از زیرمجموعه‌های *A*، *B*،...، *Z* مساوی کل عنوان مرابطه *R* باشد. برای نمونه این یکی از چندین JD بی‌اهمیتی است، که می‌توان برای S نوشت:

☆{ S,SN,SS,SC }

من موقتا از S برای اشاره به مجموعه تمام ویژگی‌ها (پرتو اصلی یا همانی) استفاده کرده‌ام. امیدوارم روشن شده باشد که هر رابطه می‌تواند بی‌کم‌وکاست، به مجموعه‌ای از پرتوها تجزیه شود، به شرط آنکه یکی از پرتوهای این مجموعه، پرتو اصلی باشد. (چنین گفتاری در مورد تجزیه خالی از اشکال نیست، چرا که یکی از پرتوهای حاصل تجزیه دقیقا همان رابطه اصلی است، منظورم این است که چنین تجزیه‌ای خیلی هم تجزیه نیست!)

تشریح وابستگی پیوندی در اینجا به پایان می‌رسد، هم اکنون من می‌توانم تعریف دقیقی از 5NF ارائه نمایم:

تعریف: مرابطه *R*، 5NF خواهد بود اگر و تنها اگر هر JD بااهمیت *R*، تنها از فراکلیدهای *R* تشکیل شده باشد.

به بیان دیگر، یک رابطه 5NF فقط دارای JDهایی است که از آنها خلاصی نداریم و اگر JD دیگری وجود داشت، رابطه 5NF نیست (و دچار افزونگی خواهد بود)، و از این رو احتمالا نیاز به تجزیه دارد.

اهمیت 5NF

حتما متوجه شده‌اید در بحث قبل مثالی نیاوردم که BCNF باشد ولی 5NF نباشد (و بنابراین همه در بهترین حالت تجزیه‌ای بی‌کم‌وکاست داشتند). دلیل عدم نمایش JDهایی که FDهای ساده نباشند: الف) احتمال وجود چنین JDها در عمل، بسیار کم است. ب) آنها موجب کمی پیچیدگی می‌شوند. به دلیل همین پیچیدگی من از آوردن مثال مستقیم خودداری کردم (به هر صورت در بخش بعد مثالی خواهم آورد) و به دلیل بعید بودن، آنها در هر حال از دیدگاه عملی اهمیت چندانی ندارند. صبر کنید تا توضیح دهم:

قبل از هر چیز، اگر شما یک طراح پایگاه هستید، حتما بایستی در مورد JDها و 5NF بدانید. اینها مانند ابزارهایی در جعبه ابزار شما هستند و (اگر همه شرایط مهیا باشد) بایستی مطمئن شوید که تمام مرابطه‌های پایگاه داده‌تان در سطح 5NF قرار دارند. لیکن در دنیای واقعی اکثریت (نه همه) BCNFها در سطح 5NF نیز قرار دارند و در عمل مرابطه‌هایی که BCNF هستند ولی 5NF نیستند بسیار نادرند. قضیه‌ای هست که به این موضوع اشاره دارد:

اگر R یک رابطه BCNF و فاقد کلید مرکب (کلیدی شامل دو ویژگی یا بیشتر) باشد، آنگاه R، 5NF است.

این قضیه بسیار کاربردی است و چیزی که می‌گوید این است که اگر به سطح BCNF برسید (که به اندازه کافی آسان است) و کلید مرکبی هم نداشته باشید (که بیشتر اوقات و نه همیشه این طور است) دیگر لازم نیست نگران مشکلات وابستگی پیوندی و 5NF باشید. در این شرایط بدون درگیری بیشتر فرض را بر این می‌گذارید که رابطه 5NF نیز هست.

به عنوان یک حاشیه، دقیق‌تر آن است که بگوییم این قضیه در مورد 3NF است، نه BCNF. در واقع رابطه 3NFای که کلید مرکب نداشته باشد، 5NF است. با این حال هر BCNFای، 3NF هم هست و BCNF از 3NF بااهمیت‌تر است (گفتار واقع‌بینانه).

5NF مبحثی است که از دیدگاه عملی اهمیت چندانی ندارد، ولی از نظر تئوری اهمیت آن *بسیار زیاد* است چرا که (در ابتدای بخش هم گفتم) «فرم نهایی نرمال» و یا فرم *شناخته‌شده* نرمال با توجه به وابستگی پیوندی است. اگر مرابطه R در سطح 5NF باشد فقط JDهای بی‌اهمیت و JDهایی که با فراکلیدها بوجود آمده‌اند در آن وجود دارند. به همین علت در تجزیه‌های بی‌کم‌وکاست چنین رابطه‌ای، هر پرتو بر روی ویژگی‌هایی که فراکلید هستند، ایجاد

شده است. به بیان دیگر هر پرتو دارای کلیدی از R است. «ترکیب» آنها را دو به دو پیوند می‌دهد، و افزونگی‌ای وجود ندارد که با تجزیه از بین برود.

اجازه دهید تا این نکته را طور دیگری شرح دهم. این که بگوییم R، 5NF است مثل این است که بگوییم تجزیه بیشتر R، در صورتی که ممکن باشد، مطمئنا افزونگی را کاهش نخواهد داد. *حتما توجه داشته باشید که 5NF بودن R نمی‌گوید که R بدون افزونگی است.* انواع گوناگونی از افزونگی وجود دارد که چنین پرتوهایی توان از بین بردن آنها را ندارند. همانطور که در مثال بخش «جایگاه نظریه طراحی» گفتم، سوالات بی‌شماری هست که نظریه طراحی پاسخ صریحی برای آنها ندارد. برای مثال شکل 7–4 را در نظر بگیرید که در آن، مرابطه SPJ با وجود آنکه 5NF است، دچار افزونگی است. مثلا توزیع‌کننده S2 قطعه P3 را بارها توزیع کرده است و همچنین P3 در پروژه J4 به کار رفته است. – JNO اینجا نشان دهنده *شماره پروژه* است– و نیز نیاز J1 توسط توزیع‌کننده S2 تامین شده است. (گزاره‌نمای مرابطه چنین است: *توزیع‌کننده SNO قطعه PNO را جهت پروژه JNO به تعداد QTY تامین کرده است*، و تنها کلید {SNO,PNO,JNO} است.) تنها وابستگی پیوندی بااهمیتی که در این رابطه وجود دارد، این وابستگی *تابعی* است:*

{SNO,PNO,JNO} → {QTY}

که یک «فلش از فراکلید خارج شده است». به بیان دیگر QTY به هر سه SNO، PNO و JNO بستگی دارد و نمی‌تواند در مرابطه‌ای که هر سه آنها در آن حضور ندارند، ظاهر شود. بنابراین هیچ تجزیه بی‌کم‌وکاستی به منظور کاهش افزونگی در اینجا امکان‌پذیر نیست.

حال چند نکته جدید پدید می‌آید که بایستی آنها را شرح دهم. اول، تاکنون نگفته‌ام ولی احتمالا حدس زده‌اید که 5NF همیشه دست یافتنی است. همواره می‌توان مرابطه‌ای که 5NF نیست [و قابل تبدیل به BCNF هست] را به پرتوهایی که 5NF هستند، تجزیه کرد.

* اگر مجموعه تمامی ویژگی‌های عنوان SPJ را SPJQ بنامیم مرابطه دارای وابستگی پیوندی { S,SN,SS,SC }☆ بی‌اهمیت خواهد بود. (یادآوری از فصل پنجم حاصل ÷یوند تنها یک رابطه مانند r خود r خواهد بود.

SPJ

SNO	PNO	JNO	QTY
S1	P1	J1	200
S1	P3	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J1	100

شکل 7–4. مرابطه 5NF به نام SPJ با مقدارهای نمونه

دوم، هر رابطه 5NFای، BCNF است و BCNF بودن یک رابطه احتمال 5NF بودن آن را رد نمی‌کند. بطور غیر رسمی متداول است که وقتی می‌گویند فلان رابطه BCNF است چنین برداشت می‌شود که BCNF هست *و در سطح نرمال‌تر نیست*. من در این فصل از این روش پیروی نکرده‌ام (و نخواهم کرد).

سوم، از آنجا که 5NF فرم نهایی نرمال است، گاهی فرم نرمال پرتو پیوند PJNF خوانده می‌شود. با تکیه بر فرم شناخته‌شده نرمال، پرتو برای تجزیه و پیوند برای ترکیب جهت همه کارها کافی خواهند بود. باید این نکته را اضافه کنم که امکان استفاده از سایر عملگرها و در نتیجه احتمال ساخت فرم‌های دیگر نرمال وجود دارد. این کار امکان‌پذیر و مطلوب است: الف) نگارش‌های تعمیم یافته پرتو و پیوند و در نتیجه ب)نگارش تعمیم یافته وابستگی پیوندی و در نتیجه ج)فرم جدید «ششم» نرمال 6NF. اهمیت چنین توسعه‌ای به‌خصوص در ارتباط با پشتیبانی از داده‌های زمان‌مند است که در کتاب *داده‌های زمان‌مند و مدل رابطه‌ای* 2003 نوشته هیو داروین، نیکوس لورنتزوس و من شرح داده شده‌اند. در هر صورت تنها کاری که می‌خواهم در اینجا انجام دهم ارائه یک تعریف از 6NF جهت مرابطه‌های «معمولی» (غیر زمان‌مند) است:

تعریف: مرابطه R در سطح 6NF است اگر و تنها اگر هیچ JD بااهمیتی نداشته باشد.

مرابطه‌های «معمولی» در صورتی 6NF هستند که دارای تنها یک کلید، همراه با حداکثر یک ویژگی دیگر باشند. مرابطه سفارش‌های خودمان SP در سطح 6NF است. همینطور مرابطه SPJ (شکل 7–4). در مقابل مرابطه‌های توزیع‌کنندگان و قطعات خودمان یعنی S و P، 5NF هستند ولی 6NF نیستند.

توجه

به 6NF گاهی اوقات *ساده‌نشدنی* گفته می‌شود چرا که تحت هیچ شرایطی نمی‌تواند بوسیله پرتو تجزیه شود. هر 6NFای لزوما 5NF است.

برای پایان این بحث، از مطالب فوق نتیجه‌گیری می‌شود که هر مرابطه «تمام کلید» و یا محتوی کلید همراه با یک ویژگی 6NF است و از همین رو حتما BCNF هم هست. ولی نتیجه گیری نمی‌شود که چنین مرابطه‌ای لزوما خوب طراحی شده است! مثلا اگر مرابطه RS (شکل 2-7) دارای FD به صورت {CITY}→{STATUS} است. پرتو از {SNO,STATUS} آن BCNF است، ولی این قطعا یک طراحی خوب نیست. (برای جزئیات بیشتر، بحث *محافظت از وابستگی‌ها* در بخش «نود و نه آفرین بر نرمال‌سازی» را ببینید.)

در مورد 5NF بیشتر بدانیم

شکل 7-5 که ساده شده مرابطه SPJ است، را در نظر بگیرید. فرض کنید در این مرابطه ساده شده، وابستگی پیوندی {SP,PJ,SJ}☆ برقرار است که SP به‌جای {SNO,PNO}، PJ به‌جای {PNO,JNO} و SJ به‌جای {SNO,JNO} می‌باشد. از دیدگاه حسی این JD به چه معناست؟ پاسخ:

SPJ

SNO	PNO	JNO
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

شکل 7-5. ساده‌شده مرابطه SPJ با مقدارهای نمونه

اول، JD به این معناست که مرابطه حاصل پیوند-و قابل تجزیه بی‌کم‌وکاست به- پرتوهای SP، PJ و SJ است. (توجه دارید که من از اسامی SP، PJ و SJ برای پرتوها و بجای زیرمجموعه‌های عنوان استفاده کرده‌ام و امیدوارم این نام‌گذاری شما را گیج نکرده باشد.)

دوم، بنابراین قید زیر برقرار می‌شود:

IF <*s*,*p*> ∈ SP AND <*p*,*j*> ∈ PJ AND <*s*,*j*> ∈ SJ THEN <*s*,*p*,*j*> ∈ SPJ

به همین دلیل اگر <*s,p*>، <*p,j*> و <*s,j*> به ترتیب در SP، PJ و SJ دیده شوند، آنگاه مطمئنا <*s,p,j*> در پیوند این سه مرابطه دیده خواهد شد و انتظار می‌رود که حاصل این پیوند برابر SPJ باشد (چیزی که JD می گوید). با مقدارهای نمونه در شکل 7-5، مثلا تاپل‌های <S1,P1>، <P1,J1> و <S1,J1> به ترتیب در SP، PJ و SJ وجود دارند و تاپل <S1,P1,J1> در SPJ مشاهده می‌شود. (من از یک اختصار بدون شرح برای تاپل استفاده کرده‌ام و یادآوری می کنم که نماد ∈ می‌تواند به صورت «دیده می‌شود در» خوانده شود.)

سوم، بدیهی است تاپل <*s,p*> در SP دیده می‌شود تنها اگر برای برخی *z*ها تاپل <*s,p,z*> در SPJ موجود باشد. به همین طریق، تاپل <*p,j*> در PJ دیده می‌شود تنها اگر برای برخی *x*ها تاپل <*x,p,j*> در SPJ موجود باشد و تاپل <*s,j*> در SJ دیده می‌شود تنها اگر برای برخی *y*ها تاپل <*s,y,j*> در SPJ موجود باشد. این قید از نظر منطقی معادل عبارت زیر است:

IF for some *x* , *y* , *z* <*s,p,z*> ∈ SPJ AND
<*x,p,j*> ∈ SPJ AND
<*s,y,j*> ∈ SPJ
THEN <*s,p,j*> ∈ SPJ

با مراجعه به شکل 7-5 مشاهده می‌شود که تاپل‌های <S1,P1,J2>، <S2,P1,J1> و <S1,P2,J1> در SPJ دیده می‌شوند و از همین رو <S1,P1,J1> نیز در SPJ دیده خواهد شد.

وابستگی پیوندی موجود معادل این قید است ولی این قید در دنیای واقعی به چه معناست؟ به این مثال توجه کنید. فرض کنید مرابطه SPJ دارای تاپل‌هایی باشد که این سه گزاره را می‌رسانند:

1- اسمیت، آچار شلاقی برخی پروژه‌ها را تامین می کند.
2- کسانی هستند که آچار شلاقی پروژه منهتن را تامین می کنند.
3- اسمیت قطعاتی را برای پروژه منهتن تامین می کند.

این JD می گوید که مرابطه بایستی دارای تاپلی باشد که مبتنی بر این گزاره است:

4- اسمیت برای پروژه منهتن، آچار شلاقی تهیه می کند.

گزاره‌های 1، 2 و 3 در حالت عادی گزاره 4 را نتیجه *نمی‌دهند*. اگر مطمئن باشیم که 1، 2 و 3 درست‌اند فقط می‌دانیم که اسمیت آچار شلاقی برخی پروژه‌ها را تامین می کند (مثلا پروژه *z*)، برخی توزیع کنندگان آچار شلاقی پروژه منهتن را تامین می کنند (مثلا توزیع کننده *x*) و

اسمیت برخی قطعات را برای پروژه منهتن تامین می‌کند (مثلا قطعه y)، ولی نمی‌توان چنین استنتاج کرد که x اسمیت، y آچار شلاقی و z پروژه منهتن است. چنین استنتاج‌های غلطی دام‌های ارتباطی نامیده می‌شوند. در این مورد وجود JD به ما می‌گوید که *دامی در کار نیست*. در این مورد خاص می‌توان درستی گزاره 4 را از گزاره‌های 1، 2 و 3 نتیجه گرفت.

در اینجا بایستی به *طبیعت حلقه‌ای* این قید توجه کرد. «اگر s به p و p به j و j مجددا به s متصل باشد آنگاه s، p و j با هم در ارتباط‌اند و بایستی در یک تاپل ظاهر شوند». مختصر و مفید، اگر چنین حلقه‌ای به وجود آید ممکن است با رابطه‌ای سروکار داشته باشیم که BCNF است ولی 5NF نیست. بنابر تجربه شخصی من، چنین حلقه‌هایی در عمل بسیار نادرند. به همین دلیل همانطور که در بخش قبل هم گفتم فکر نمی‌کنم از نقطه نظر عملی اهمیت زیادی داشته باشند.

این بخش را با ذکر مختصری از فرم *چهارم* نرمال 4NF به پایان می‌برم. در بخش «مفهوم 5NF» گفتم که اگر شما یک طراح پایگاه باشید، بایستی در مورد JDها و 5NF اطلاعاتی داشته باشید. واقعیت این است که در مورد وابستگی‌های *چندمقداری* (MVDها) و فرم چهارم نرمال نیز لازم است چیزهایی بدانید. خاطر نشان می‌کنم که این فقط یک بحث تکمیلی است و مانند 2NF و 3NF بحثی عمدتا تاریخی به شمار می‌رود. فقط جهت ثبت در کتاب می‌گویم:

- هر MVD، JDای است که بیش از دو پرتو در آن حضور ندارند (در عمل دقیقا دو).
- یک مرابطه 4NF است اگر و تنها اگر هر MVD بااهمیت آن، بوسیله فراکلید اعمال شده باشد.

جزئیات در این مورد که چه MVDهایی بی‌اهمیت هستند و یا بوسیله فراکلید اعمال شده‌اند، از محدوده این بحث خارج است (تمرین 19 را ببینید). ولی فقط تاکید می‌کنم در پی این تعریف، برای تجزیه بی‌کم‌وکاست به دقیقا دو پرتو، بایستی حداقل در سطح 4NF قرار داشته باشیم. در مقابل JD بخش قبل دارای سه پرتو بود و مطمئنم که متوجه این موضوع شده‌اید. در واقع می‌توان گفت برای تجزیه به n پرتو (n>2) رسیدن به سطح 5NF در صورتی ضروری است، که مرابطه دارای یک قید حلقه‌ای nتایی باشد؛ یعنی، یک JD با n پرتو (و نه کمتر) در آن برقرار باشد.

نود و نه آفرین بر نرمال‌سازی

نرمال‌سازی را نمی‌توان نوشداروی همه دردها دانست، این را با در نظر گرفتن هدف‌های نرمال‌سازی در کنار آنچه با این اهداف در تضاد است، می‌توان مشاهده کرد. این اهداف عبارتند از:

- دستیابی به طراحی‌ای که نمایشی «خوب» از دنیای واقعی باشد. یادگیری آن به دلیل ذاتی بودن آسان بوده و می‌تواند سکویی برای پیشرفت‌های آتی در هنر طراحی باشد.
- کاهش افزونگی
- در نتیجه، جلوگیری از آنومالی به هنگام به‌روزرسانی
- ساده‌سازی در عبارت‌ها و الزام به رعایت قیدهای جامعیت

من به نوبت به هر یک از آنها می‌پردازم:

یک نمایش خوب از دنیای واقعی: نرمال‌سازی به خوبی از پس این کار برمی آید و من هیچ انتقادی به آن ندارم.

کاهش افزونگی: نرمال‌سازی برای مقابله با افزونگی شروع خوبی است ولی فقط یک شروع است. یک دلیل این امر آن است که در نرمال‌سازی تجزیه‌ای بی‌کم‌وکاست انجام می‌شود و (همانطور که دیدیم) تجزیه بی‌کم‌وکاست نمی‌تواند تمام افزونگی‌ها را از بین ببرد. در واقع برخی افزونگی‌ها هستند –در این قسمت درباره آنها بحث زیادی نشده است– که با نرمال‌سازی هیچ نسبتی ندارند. دلیل دیگر آن که، هدف کاهش افزونگی ممکن است با هدفی دیگر تضاد داشته باشد، در این مورد هم تاکنون صحبتی نشده بود، این هدف *محافظت از وابستگی‌ها* است. صبر کنید تا توضیح دهم. مرابطه زیر را در نظر بگیرید. (ویژگی ZIP نشان‌دهنده کدپستی(5رقمی)، STATE استان، CITY شهر و STREET خیابان است):

ADDR { STREET, CITY, STATE, ZIP }

فرض کنید FD زیر در مرابطه وجود دارند:

{ STREET, CITY, STATE } → { ZIP }
{ ZIP } → { CITY, STATE }

FD دوم می‌رساند که این رابطه BCNF نیست. ولی با اعمال قضیه هیث و تجزیه آن به پرتوهای BCNF زیر:

```
ZCS { ZIP, CITY, STATE }
   KEY { ZIP }

ZS { ZIP, STREET }
  KEY { ZIP, STREET }
```

وابستگی {STREET, CITY, STATE}→{ZIP} که در مرابطه اصلی وجود دارد، «ناپدید» می‌شود! (این وابستگی در پیوند ZS و ZCS وجود دارد، ولی این کافی نیست و وابستگی باید در هر یک از پرتوها به تنهایی موجود باشد.) در نتیجه ZS و ZCS نمی‌توانند بطور مستقل به‌روزرسانی شوند. مثلا فرض کنید در حال حاضر این پرتوها دارای مقدارهای موجود در شکل 7-6 هستند. در صورت درج تاپل <10111,Broadway> در ZS، وابستگی «ناپدید شده» زیرپا گذاشته خواهد شد. برای تصمیم‌گیری بایستی هم ZCS و هم ZS مورد بررسی قرار گیرند. به همین دلیل قاعده محافظت از وابستگی‌ها می‌گوید: *بوسیله پرتو وابستگی‌ها را گسسته نکنید*. مثال ADDR نشان داد که، متاسفانه، این هدف و هدف تجزیه تا سطح BCNF گاهی با هم در تضاد قرار می‌گیرند.

ZCS

ZIP	CITY	STATE
10003	New York	NY
10111	New York	NY

ZS

ZIP	STREET
10003	Broadway

شکل 7-6. پرتوهای ZCS و ZS با مقدارهای نمونه

جلوگیری از آنومالی: این هدف عملا همان هدف قبلی (کاهش افزونگی)، با یک نام دیگر است. همه می‌دانند که طراحی‌هایی که کاملا نرمال نباشند، به دلیل افزونگی احتمالی در معرض آنومالی هنگام به‌روزرسانی قرار دارند. برای مثال در مرابطه STP (یک‌بار دیگر شکل 7-1)، توزیع‌کننده S1 ممکن است در یک تاپل دارای رتبه 20 و در تاپل دیگر دارای رتبه 25 باشد. (البته این «آنومالی به‌روزرسانی» فقط در صورتی امکان بروز دارد که، جامعیت بطور کامل برقرار نشده باشد. شاید راه توصیف آنومالی به‌روزرسانی به این روش بهتر باشد: اگر طراحی

کاملا نرمال باشد، قیدهایی که از چنین آنومالی‌هایی جلوگیری می‌کنند به آسانی قابل شرح و احتمالا به آسانی قابل اعمال خواهند بود. پاراگراف بعد را ببینید.)

ساده‌سازی عبارات و الزام به رعایت قیدهای جامعیت: بدیهی است که برقراری برخی قیدها، تضمین کننده برقراری برخی قیدهای دیگر هستند. به عنوان یک مثال ساده، اگر مقدارها بایستی کوچکتر یا مساوی 5000 باشند، آنها قطعا بایستی کوچکتر یا مساوی 6000 باشند (صحبت تقریبی). حال اگر قید A تضمین کننده قید B باشد، اعلان برقراری قید A، قید B را بطور «اتوماتیک» برقرار خواهد کرد (B دیگر نیازی به اعلان نخواهد داشت). نرمال‌سازی تا سطح 5NF راه ساده‌ای است برای اعلان و برقراری برخی قیدهای مهم: تنها کاری که پس از آن باید انجام دهیم تعریف کلیدها و تضمین یکتایی آنهاست –کاری که در هر حال انجام می‌دادیم– تا از آن پس تمامی JDها (و MVDها و FDها) به صورت اتوماتیک برقرار شوند چرا که اعمال همه آنها بوسیله کلیدها تضمین می‌شوند. نرمال‌سازی در اینجا نقش بزرگی ایفا می‌کند.

از سوی دیگر چند دلیل برای اینکه چرا نرمال‌سازی نوشدارو نیست. اول، JDها تنها یکی از انواع قیدها هستند و نرمال‌سازی به نوع‌های دیگر هیچ کمکی نمی‌کند. دوم، یک مرابطه خاص بسیاری مواقع می‌تواند به روش‌های گوناگونی به پرتوهای 5NF تجزیه شود و هیچ رهنمود رسمی برای انتخاب یکی از آنها وجود ندارد. سوم، در طراحی مباحث بسیاری وجود دارد که نرمال‌سازی سخنی از آنها به زبان نمی‌آورد. برای مثال از کجا بفهمیم که بایستی یک مرابطه برای توزیع‌کنندگان داشته باشیم و نه چند مرابطه برای توزیع‌کنندگان لندنی، پاریسی و غیره؟ این امر از بدیهیات نرمال‌سازی نیست.

بایستی روشن کنم که گفتار من در این بخش نبایستی به منزله حمله به نرمال‌سازی تلقی شود. من به این نظر پایبندم که هر چیز کمتر از کاملا نرمال، *شدیدا دور از صرفه* است. من قصد دارم این مطلب را با یک بحث –*منطقی*– ختم کنم و مسلما برخی از شما تاکنون چیزی در مورد به آن نشنیده‌اید و آن در حمایت از این نظر است که *«کاهش درجه نرمال» تنها بایستی به عنوان آخرین راه حل مورد استفاده قرار گیرد.* فقط در صورتی که تمامی راه‌ها برای افزایش کارایی به نحوی شکست خورده باشند می‌توان از طراحی با نرمال‌سازی کامل عقب‌نشینی نمود. به خاطر داشته باشید من با این عقیده که نرمال‌سازی کارایی را بالا می‌برد، موافقم. بلی؛ اما این امر در

محصولات SQL به گونه دیگری است که بعدا به آن خواهم پرداخت (بخش «توصیه‌هایی در مورد طراحی فیزیکی» را ببینید). در هر حال به این استدلال توجه کنید.

همه ما می‌دانیم که کاهش درجه نرمال برای به‌روزرسانی‌ها مضر است (از نظر منطقی بد است یعنی عبارت‌نویسی آنها را سخت‌تر می‌کند و می‌تواند جامعیت پایگاه را هم به خطر اندازد). همچنین به نظر می‌رسد که این امر برای بازیابی هم مضر باشد و نوشتن کوئری‌های خاص را مشکل‌تر نماید (نوشتن عبارت‌های غلط آسان‌تر می‌شود یعنی اگر عبارات قابل اجرا باشند ممکن است پاسخی «صحیح» برگردانند ولی پاسخ‌هایی هستند که به یک کوئری غلط داده شده‌اند). توضیح می‌دهم. دوباره مرابطه LS (شکل 7–2) را نگاه کنید که در آن FD، {CITY}→{STATUS} برقرار است. کوئری «متوسط رتبه شهرها را بده» را در نظر بگیرید. با در نظر گرفتن مقدارهای نمونه در شکل، شهرهای لندن، آتن و پاریس دارای رتبه‌های 30، 20 و 30 هستند و در نتیجه معدل (تا سه رقم اعشار) 26/667 خواهد شد. حال می‌خواهیم عبارت SQL این کوئری را بنویسیم:

```
1  SELECT AVG ( RS.STATUS ) AS RESULT
   FROM  RS
```

نتیجه (غلط): 26. مشکل این است که رتبه پاریس و لندن هر یک دوبار به حساب آمده‌اند. گویا به DISTINCT درون AVG نیاز داریم. امتحان می‌کنیم:

```
2  SELECT AVG ( DISTINCT RS.STATUS ) AS RESULT
   FROM  RS
```

نتیجه (غلط): 25. ما به متمایز کردن *شهرها* نیاز داشتیم، نه رتبه‌ها. با گروه‌بندی چه کار می‌توان کرد:

```
3  SELECT RS.CITY, AVG ( RS.STATUS ) AS RESULT
   FROM   RS
   GROUP  BY RS.CITY
```

نتیجه (غلط): <Athens,30>, <London,20>, <Paris,30>. این عبارت معدل رتبه هر *شهر* را می‌دهد نه معدل کلی شهرها را. شاید چیزی که می‌خواهیم معدلِ معدل‌هاست.

```
4  SELECT RS.CITY, AVG ( AVG ( RS.STATUS ) ) AS RESULT
   FROM   RS
   GROUP  BY RS.CITY
```

نتیجه: خطا. SQL استاندارد به درستی* اجازه اجرای «توابع مجموعه‌ای» را نمی‌دهد و AVG تودرتو در این گروه قرار دارد. یک تلاش دیگر:

```
5  SELECT AVG ( TEMP.STATUS ) AS RESULT
   FROM ( SELECT DISTINCT RS.CITY, RS.STATUS
          FROM   RS ) AS TEMP
```

نتیجه (بالاخره درست): 26/667 همانطور که در فصل 5 گفتم تمامی محصولات SQL از زیرکوئری‌های تودرتو در بند FROM می‌شود پشتیبانی نمی‌کنند.

این انتهای نرمال‌سازی است (بهتر است بگوییم فعلا)، حال می‌خواهم سراغ موضوعی بروم که کمتر با آن آشنایی دارید، *تفکیک(تعامد)*، که بخش کوچک دیگری است از دانش طراحی پایگاه داده‌ها.

تعامد (تفکیک)

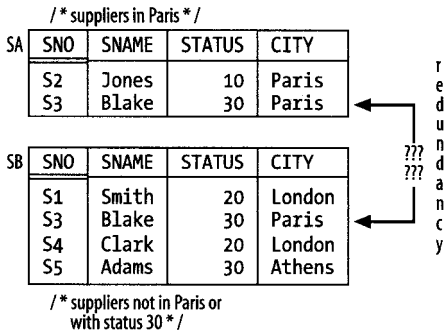
شکل 7-7 یک طراحی ممکن ولی بطور واضح بد را برای توزیع‌کنندگان نشان می‌دهد. مرابطه SA متعلق به توزیع‌کنندگان مستقر در پاریس است و رابطه SB مربوط به توزیع‌کنندگانی است که در پاریس نیستند و یا دارای رتبه 30 هستند. همانطور که می‌بینید این طراحی موجب افزونگی شده است و بطور مشخص تاپل S3 در هر دو مرابطه آمده است. مثل همه افزونگی‌ها، این مورد هم موجب آنومالی در به‌روزرسانی می‌گردد. (افزونگی از هر نوعی که باشد، ظرفیت ایجاد آنومالی در به‌روزرسانی را دارد.)

---

* به این دلیل گفتم به درستی چون دقیقا در مورد SQL صحبت می‌کنیم. زبان‌های کامل‌تر بایستی اجازه اجراهای تودرتو را بدهند. اجازه بدهید تا توضیح بدهم. عبارت زیر را در SQL در نظر بگیرید:

```
SELECT SUM(SP.QTY) AS SQ FROM SP WHERE SP.QTY > 100
```

(عمدا مثال دیگری آوردم). در اینجا SUM به SP.QTY FROM SP WHERE SP.QTY > 100 برمی‌گردد و زبان‌های کامل‌تر کل عبارت بایستی درون پرانتز قرار گیرد. ولی SQL چنین نیست در نتیجه عبارتی به شکل AVG(SUM(QTY)) غیرمجاز است چرا که SQL نمی‌تواند بفهمد کدام بخش از عبارت محصور شده مربوط به SUM و کدام بخش مربوط به AVG است.



شکل 7-7. مرابطه‌های SA و SB با مقدارهای نمونه

به یاد داشته باشید که S3 *باید* در هر دو مرابطه حضور داشته باشد. اگر فرض کنیم که این تاپل در SB هست ولی در SA وجود ندارد، از عدم حضور در SA با فرض بسته بودن جهان می‌توان چنین انگاشت که محل استقرار S3 پاریس نیست ولی SB می‌گوید که S3 در پاریس *است*. بنابراین یک تناقض روی دست‌مان مانده است و پایگاه داده ثبات خود را از دست داده است.

مشکل طراحی 7-7 بدیهی است. بطور خلاصه، دو تاپل یکسان می‌توانند در دو مرابطه مجزا ظاهر شوند و دقیق‌تر یک تاپل در دو مرابطه ظاهر شده است بدون اینکه هیچ قیدی زیرپا گذاشته شود. یک قانون بدیهی:

اصل طراحی متعامد (تفکیکی) (نگارش اول): قیدها نبایستی اجازه دهند که یک تاپل در دو مرابطه یک پایگاه داده ظاهر شود.

کلمه *متعامد(تفکیکی)* در اینجا به این واقعیت اشاره دارد که طبق این اصل مرابطه‌ها بایستی از هم مجزا و مستقل باشند -که در اینجا نیستند- حال اگر قیدهای آنها «منع همپوشانی» داشته باشند چنین اتفاقی نمی‌افتد.

بدیهی است که اگر دو رابطه هم‌نوع نباشند نمی‌توانند این اصل را نقض کنند و در نتیجه می‌توانید چنین تصور کنید که این اصل در چنین مواردی ارزش چندانی ندارد و در نهایت، وجود دو یا چند مرابطه هم‌نوع در پایگاه چندان متداول نیست. ولی شکل 7-8 را در نظر بگیرید که یک طراحی بد دیگر از توزیع‌کنندگان را نشان می‌دهد. با این وجود که یک تاپل نمی‌تواند در دو مرابطه ظاهر شود، ممکن *است* یک تاپل از SX و یک تاپل از SY دارای پرتوی یکسان در

{SNO,SNAME}* باشند و این حقیقت نیز سبب افزونگی و آنومالی هنگام به‌روزرسانی گردد. از این رو بایستی اصل طراحی را توسعه دهیم:

اصل طراحی متعامد (تفکیکی) (نگارش دوم و نهایی): فرض کنید A و B دو مرابطه مجزا در یک پایگاه داده هستند. اگر A و B به ترتیب قابل تجزیه بی‌کم‌وکاست به A_1،A_2،...A_m و B_1،B_2،...B_m باشند، نبایستی هیچ A_i و B_jای وجود داشته باشند، به گونه‌ای که یک تاپل بتواند در هر دو ظاهر شود.

تجزیه بی‌کم‌وکاست همان چیزی است که در نرمال‌سازی نیز با آن سروکار داشتیم.

SX

SNO	SNAME	STATUS
S1	Smith	20
S2	Jones	10
S3	Blake	30
S4	Clark	20
S5	Adams	30

SY

SNO	SNAME	CITY
S1	Smith	London
S2	Jones	Paris
S3	Blake	Paris
S4	Clark	London
S5	Adams	Athens

شکل 7-8. مرابطه‌های SY و SX با مقدارهای نمونه

پس از این بحث و تعریف، چند نکته جدید را مطرح می‌کنم:

- نگارش دوم این اصل، نگارش اول را هم شامل می‌شود چرا که یک تجزیه بی‌کم‌وکاست هست که همیشه و برای هر مرابطه R وجود دارد و آن پرتو اصلی R است (پرتوی که شامل تمام ویژگی‌های R است).
- *اصل طراحی متعامد* نیز مانند نرمال‌سازی موضوعی ذاتی است ولی این اصل (باز هم مانند نرمال‌سازی) به آن رسمیت می‌بخشد.
- هدف از طراحی متعامد کاهش افزونگی و در نتیجه جلوگیری از آنومالی‌های به‌روزرسانی است. (باز هم مانند نرمال‌سازی). در واقع تعامد، مکمل نرمال‌سازی است و به صورت تقریبی می‌توان گفت نرمال‌سازی افزونگی داخلی مرابطه‌ها و تفکیک (تعامد) افزونگی سراسری مرابطه‌ها را کاهش می‌دهد.

* پرتو یک عملگر رابطه‌ای است ولی کاملا روشن است که می‌توان نگارشی از این عملگر تعریف کرد که بجای رابطه‌ها، با تاپل‌ها کار کند و با آن پرتوی از یک تاپل را پیدا کرد. این امر در مورد برخی دیگر از عملگرهای رابطه‌ای نیز صادق است.

- تفکیک (تعامد) به نوع دیگری نیز مکمل نرمال‌سازی است. مجددا تجزیه مرابطه S به SX و SY در شکل 7-8 را در نظر بگیرید. می‌بینیم که *تمامی* اصول نرمال‌سازی در این طراحی رعایت شده‌اند! هر دو پرتو در سطح 5NF هستند، تجزیه بی‌کم‌وکاست است، وابستگی‌ها محافظت شده‌اند و برای بازسازی مرابطه S هر دو پرتو مورد نیازاند.* این تعامد –و نه نرمال‌سازی– است که به ما می‌گوید، طراحی به درستی انجام نگرفته است.
- تعامد گرچه امری ذاتی است، ولی بسیاری اوقات در عمل مورد بی‌اعتنایی قرار می‌گیرد. چنین نحوه طراحی را بارها، در پایگاه‌های داده تجاری، دیده‌ایم:

ACTIVITIES_2001 { ENTRYNO, DESCRIPTION, AMOUNT, NEW_BAL }
ACTIVITIES_2002 { ENTRYNO, DESCRIPTION, AMOUNT, NEW_BAL }
ACTIVITIES_2003 { ENTRYNO, DESCRIPTION, AMOUNT, NEW_BAL }
ACTIVITIES_2004 { ENTRYNO, DESCRIPTION, AMOUNT, NEW_BAL }
ACTIVITIES_2005 { ENTRYNO, DESCRIPTION, AMOUNT, NEW_BAL }
...

با استفاده از فقط یک مرابطه، طراحی بهتر خواهد شد:

ACTIVITIES { ENTRYNO, DESCRIPTION, AMOUNT, NEW_BAL, YEAR }

توجه

یکی از دلایل اینکه چنین طراحی‌هایی –با چند مرابطه– در عمل انجام می‌شوند این است که معمولا دلایلی برای جداکردن داده‌ها در سطح فیزیکی وجود دارد و با انجام این کار بایستی داده‌ها هنوز هم به همان خوبی در سطح منطقی قابل نمایش باشند. ولی دلایلی که با توجه به طراحی فیزیکی عنوان می‌شوند، نمی‌توانند طراحی بد منطقی را توجیه نمایند.

- اگر A و B دو مرابطه هم‌نوع باشند، در تبعیت از اصل طراحی متعامد به صورت زیر پیاده‌سازی می‌شوند و بایستی کاملا مجزا باشند، بدین معنا که هیچ تاپلی

* پیش از این گفتم تجزیه‌ای بی‌کم‌وکاست است که با پیوند پرتوهای آن بتوانیم به مرابطه اصلی بازگردیم. این درست است، ولی کافی نیست. در عمل مطمئنا ما این شرط را اضافه می‌کنیم که تمامی پرتوها بایستی مورد نیاز باشند. برای مثال قاعدتا مرابطه S را به پرتوهای {SNO}، {SNO,STATUS} و {SNO,SNAME,CITY} تجزیه نمی‌کنیم گر چه S از پیوند آنها بدست می‌آید.

نباید در هر دو آنها ظاهر شود. (اجتماع آنها ما را به رابطه اولیه بازخواهد گردانید.) به چنین تجزیه‌ای، *تجزیه‌متعامد* گفته می‌شود.

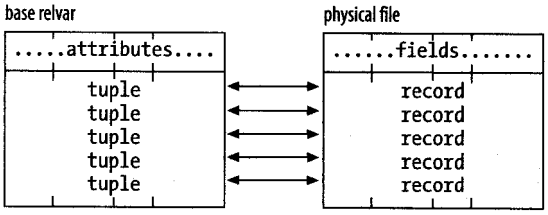
A INTERSECT *B* : *is always empty*
A UNION *B* : *is always a disjoint union*
A MINUS *B* : *is always equal to A*

توصیه‌هایی درباره طراحی فیزیکی

مدل رابطه‌ای چیزی برای گفتن درباره طراحی فیزیکی ندارد ولی چیزهایی هستند که می‌توانند در مورد طراحی فیزیکی سیستم‌های رابطه‌ای گفته شوند، چیزهایی که دست‌کم توسط مدل پیاده‌سازی شده‌اند ولی هرگز بطور روشن تشریح نشده‌اند (جزئیات طراحی فیزیکی از ویژگی‌های DBMS است و در مورد آنها بین سیستم‌های مختلف تفاوت وجود دارد).

اولین نکته این است که *طراحی فیزیکی باید دنباله‌رو طراحی منطقی باشد.* رویکرد «درست» آن است که ابتدا طراحی منطقی را تمیز انجام دهیم و سپس با پیگیری گام به گام این طرح، ساختارهای فیزیکی DBMS مورد نظر طرح شوند. طراحی فیزیکی بایستی از طراحی منطقی ناشی شده باشد و راه‌های دیگر برای بدست آوردن آن قابل قبول نیست. در حالت ایده‌آل سیستم بایستی قادر باشد بدون دخالت افراد، خودش طراحی فیزیکی بهینه را پیدا کند (این هدف آنقدرها هم که فکر می‌کنید دور از دسترس نیست. در این فصل کمی در مورد آن توضیح می‌دهم.)

نکته دوم: در فصل 1 دیدیم، یکی از دلایل این که هیچ کدام از مباحث طراحی فیزیکی در نظریه رابطه‌ای جایی ندارند این است که دست پیاده‌سازی کنندگان در اجرای مدل کاملا باز باشد. من تصور می‌کنم که عدم درک مدل همه‌گیر شده و این امر به ما لطمه وارد کرده است، همانا بیشتر محصولات SQL در بهره‌برداری کامل از قابلیت‌های مدل ناموفق بوده‌اند. در این محصولات بین چیزی که کاربر می‌بیند و چیزی که به صورت فیزیکی ذخیره می‌شود، تفاوتی وجود ندارد. به بیان دیگر در عمل انباره فیزیکی، تصویر مستقیم چیزی است که کاربر در سطح منطقی می‌بیند، همانطور که شکل 7–9 به آن اشاره دارد. (من می‌دانم که این مفاهیم بیش از حد ساده گرفته شده‌اند ولی هنوز آنقدر درست هستند که مقصود را منتقل کنند.)



شکل 7-9. پیاده‌سازی با تصویر مستقیم (بد)

پیاده‌سازی به روش تصویر مستقیم اشکالات زیادی دارد که برای دوری از طولانی شدن بحث وارد جزئیات نمی‌شوم، ولی از همه آنها برجسته‌تر آن است که این کار استقلال داده‌ای را تامین نمی‌کند. اگر بخواهیم طراحی فیزیکی را تغییر دهیم (به دلیل افزایش کارایی)، مجبور خواهیم بود که طراحی منطقی را هم عوض کنیم. بارها شنیده‌ایم که بایستی «کاهش درجه نرمال به منظور افزایش کارایی» انجام شود ولی در اصل نحوه طراحی منطقی مطلقا ربطی به کارایی ندارد اما اگر طراحی منطقی با طراحی فیزیکی دارای رابطه یک به یک باشد آنوقت.... معلوم است چه اتفاقی خواهد افتاد. مطمئنا ما می‌توانیم وضعی بهتر از این داشته باشیم. هواداران مدل رابطه‌ای سالهاست که می‌گویند مدل رابطه‌ای نباید به این شکل پیاده‌سازی شود. یکی از محصولات کاملا جدید که بزودی به بازار عرضه خواهد شد، مشکلات مربوط به تصویر مستقیم را در دستور کار خود قرار داده است. این تکنولوژی *ترارابطه‌ای* (*TransRelational*) نام دارد. از آنجا که این تکنولوژی درباره پیاده‌سازی است ذکر جزئیات آن از موضوع این کتاب خارج است و می‌توانید کلیات آن را در کتاب *آشنایی با سیستم‌های پایگاه داده‌ها* ویرایش هشتم نگارش اینجانب، پیدا کنید. تنها کاری که من می‌خواهم در اینجا انجام دهم این است که تا حدودی ویژگی‌های پیاده‌سازی‌هایی که با دقت و سخت‌گیری سطوح منطقی و فیزیکی را از هم جدا کرده‌اند، را برایتان شرح دهم.

اول، هرگز مجبور به «کاهش درجه نرمال برای کارایی» نخواهیم بود (منظورم در سطح منطقی است). بدون پرداخت هیچ تاوانی، تمام مرابطه‌ها می‌توانند در سطح 5NF و حتی 6NF باشند. طراحی فیزیکی فارغ از مساله کارایی انجام می‌شود.

دوم، 6NF در برخورد با مشکل گم‌شدگی اطلاعات راه واقع‌گرایانه‌ای را در پیش گرفته است (یعنی وارد حوزه تهی و منطق سه‌حالته نشده است). با استفاده از تهی، در واقع پایگاه داده را

مجبور می‌کنید به صراحت اعلام کند که چه چیزهایی را نمی‌داند. ولی اگر کسی چیزی را نمی‌داند بهتر است اصلا چیزی نگوید که چنین گفته‌اند:

به آیینه گر خویشتن دیدمی — به بی‌دانشی پرده ندریدمی
چو مردم سخن گفت باید به‌هوش — وگرنه شدن چون بهایم خموش
بهایم خموشند و گویا بشر — زبان بسته بهتر که گویا به شر!

برای مثال فرض کنید فقط دو توزیع‌کننده وجود دارند S1 و S2. رتبه S1 را می‌دانیم ولی رتبه S2 را نمی‌دانیم. یک طراحی 6NF برای این وضعیت می‌تواند به صورت شکل 7–10 نشان داده شود.

SN

SNO	SNAME
S1	Smith
S2	Jones

SS

SNO	STATUS
S1	20

SC

SNO	CITY
S1	London
S2	Paris

شکل 7–10. وضعیتی که در آن رتبه توزیع‌کننده S2 نامعلوم است.

گفتنی درباره گم‌شدگی اطلاعات بسیار است ولی اینجا جای آنها نیست. چیزی که در اینجا می‌خواهم بر آن تاکید کنم این است که در این نوع طراحی مجبور نیستیم یک «تاپل» داشته باشیم که در آن رتبه S2 «تهی» است، بلکه *اصلا* تاپلی نداریم که رتبه S2 را نشان دهد.

در نهایت، سیستمی که مد نظر من است بایستی واقعا بطور خودکار بتواند طراحی فیزیکی را از روی طراحی منطقی بدست آورد و دخالت فرد طراح در این امر بایستی بسیار ناچیز و یا هیچ باشد. با در نظر گرفتن همه جوانب من در اینجا نمی‌توانم دلیلی برای این امر بیاورم ولیکن به آن سخت معتقدم.

خلاصه

مرکز توجه این فصل بر نظریه طراحی پایگاه در سطح منطقی بود یعنی نرمال‌سازی و تعامد (تفکیک) (بخش علمی نظام و قواعد طراحی پایگاه). نکته اصلی این است که طراحی منطقی، بر خلاف طراحی فیزیکی مستقل از DBMS است و یا باید باشد. و دیدیم که اصول تغییر ناپذیری وجود دارند که معمولا می‌توانند در حل مسائل به کار روند.

نکته دیگری که در متن به روشنی به آن اشاره نکردم این است که طراحی منطقی همانطور که از DBMS مستقل است، بایستی از برنامه‌های کاربردی هم مستقل باشد. هدف ایجاد نوعی طراحی است که تمرکز آن بیشتر بر *معنای* داده‌ها باشد تا چگونگی کاربرد آنها، و به همین دلیل هم بر مفهوم قیدها و گزاره‌نماها تاکید کرده‌ام. پایگاه داده‌ها به چشم یک نمایش قابل اعتماد از معنای وضعیتی خاص، نگریسته می‌شود و قیدها به خود معنا اشاره دارند. در کل روند طراحی منطقی چیزی شبیه این است:

1- موشکافی گزاره‌نماهای مرابطه‌ها با دقت هرچه بیشتر.

2- انتساب خروجی مرحله 1 به مرابطه‌ها و قیدها. البته تعدادی از این قیدها به صورت FDها، MVDها و یا JDها خواهند بود.

دلیل دیگر مطلوب بودن استقلال از برنامه‌های کاربردی آن است که هیچ‌گاه نمی‌توان تمامی کاربردهای داده‌ها را پیش‌بینی کرد. بنابراین اگر نمی‌خواهیم بعدها پایگاه در برابر نیازهای پردازشی جدید ناتوان شود، بایستی یک طراحی قوی داشته باشیم.

مهمترین کاری که نظریه طراحی انجام می‌دهد *کاهش افزونگی* است. نرمال‌سازی افزونگی داخلی مرابطه‌ها و تفکیک افزونگی سراسری مرابطه‌ها را کاهش می‌دهد. بحث من در مورد نرمال سازی بر روی BCNF و 5NF -که فرم‌های شناخته‌شده نرمال با توجه به وابستگی‌های تابعی و پیوندی هستند- بود. (با این حال دست کم اشاره‌ای به دیگر فرم‌های نرمال بخصوص 6NF داشتم.) گفتم که نرمال‌سازی برقراری قیدها را آسانتر (و شاید اجبار) می‌کند. در غیر این صورت -تا به حال این را نگفته‌ام- به‌روزرسانی بیش از یک تاپل در یک لحظه، بایستی از لحاظ منطقی قابل قبول باشد (چرا که طراحی‌های غیر نرمال دارای افزونگی هستند و افزونگی بر این امر دلالت دارد که گاهی اوقات بایستی چند چیز به طور یک‌جا به‌روزرسانی شوند). روشن کردم که نرمال سازی یک احساس ذاتی را به صورت رسمی درمی‌آورد. همچنین برای عدم کاهش درجه نرمال یک استدلال منطقی و شاید نامانوس آوردم که بجای به‌روزرسانی، در مورد بازیابی بود؛ در اینجا می‌خواهم اضافه کنم که بر خلاف شعار «کاهش درجه نرمال برای کارایی»، همانطور که احتمالا می‌دانید،کم کردن درجه نرمال در عمل به کارایی لطمه وارد می‌نماید (هم در به‌روزرسانی و هم در بازیابی). در واقع «کاهش درجه نرمال برای کارایی» معمولا به معنای افزایش برای کارایی یک کاربرد خاص به بهای قربانی کردن کارایی در بقیه موارد است.

اصل طراحی متعامد (تفکیکی) را نیز توضیح دادم (بازهم رسمی کردن احساس)، و چند توصیه در مورد طراحی فیزیکی مطرح کردم. اولا طراحی فیزیکی بایستی از روی طراحی منطقی بدست آید و هیچ راه دیگر قابل قبول نیست. ثانیا اگر بتوانیم به نحوی از طراحی‌های امروزی که به صورت تصویر مستقیم هستند رهایی پیدا کنیم، بسیار خوب خواهد شد. ثالثا اگر انجام طراحی فیزیکی هم کاملا خودکار شود، خیلی خوب است و من در این مورد ابراز امیدواری کردم.

آخرین نکته، تاکید می‌کنم که قوانین نرمال‌سازی و تفکیک همواره اختیاری‌اند. نبایستی این قانون‌ها را به شکل وحی‌منزل دید که هرگز نباید زیرپا گذاشته شوند. همانطور که می‌دانیم برای عدم نرمال‌سازی «کامل» دلایلی وجود دارد (منظورم دلایل *منطقی* است، در مورد «کاهش درجه نرمال برای کارایی» صحبت نمی‌کنم). این امر در مورد تفکیک هم صادق است. عدم انجام نرمال‌سازی کامل موجب افزونگی می‌شود و ممکن است آنومالی‌های خاصی را موجب شود، رعایت نکردن تفکیک هم چنین است. حتی با وجود نظریه طراحی که در این فصل مطرح شد، طراحی پایگاه داده‌ها هنوز نیازمند نوعی مصالحه در برقراری تعادل میان نیازهاست.

تمرین‌ها

1- وابستگی تابعی و وابستگی پیوندی را به دقیق‌ترین شکل ممکن تعریف کنید.

2- تمام FDهای برقرار –بی‌اهمیت و بااهمیت– در مرابطه سفارش‌های خودمان SP را ذکر کنید.

3- موضوع FD به موضوع تساوی تاپل‌ها وابسته است: درست یا غلط؟

4- قضیه هیث را ثابت کنید. همچنین ثابت کنید که عکس این قضیه درست نیست.

5- تجزیه بی‌کم‌وکاست بدین معناست که یک مرابطه به چند پرتو تجزیه شود به گونه‌ای که با پیوند آنها بتوان دوباره مرابطه اولیه را بدست آورد. در واقع اگر پرتوهای *r1* و *r2* از مرابطه *r* به گونه‌ای باشند که هر ویژگی *r* حداقل در یکی از آنها دیده شود آنگاه پیوند *r1* و *r2* همیشه تمامی تاپل‌های *r* را تولید می‌کند. این ادعا را ثابت کنید. (بنابراین مشکل تجزیه از کم و کاست دار این است که تاپل‌های اضافه «قلابی» تولید می‌کند. از آنجا که نمی‌توان تاپل‌های اصلی و قلابی را از هم تشخیص داد، اطلاعات از بین می‌روند.)

6- فراکلید چیست؟ این که می‌گویند FD بوسیله فراکلید اعمال می‌شود به چه معناست؟ این که می‌گویند JD بوسیله فراکلید اعمال می‌شود به چه معناست؟

7- کلیدها یکتا و کاهش‌ناپذیر فرض می‌شوند. حال، آشکار است که سیستم قابلیت اجبار به یکتایی را دارد؛ ولی در مورد کاهش‌ناپذیری چه می‌توان گفت (آیا می‌توان آن را هم توسط سیستم اجبار کرد)؟

8- منظور چیست از: الف) یک FD بی‌اهمیت ب) یک JD بی‌اهمیت. آیا اولی حالت خاصی از دومی است؟

9- فرض کنید *R* مرابطه‌ای از درجه *n* باشد. بیشترین تعداد *FDهایی* که *R* میتواند داشته باشد چندتاست (با اهمیت و بی‌اهمیت)؟

10- فرض کنید *A* و *B* که هر دو *مجموعه‌ای* از ویژگی‌ها هستند و FD، *A→B* در آنها برقرار است. اگر یکی از آنها تهی باشد چه اتفاقی خواهد افتاد؟

11- یک گزاره: در روز *d* زنگ *p* دانش‌آموز *s* درس *l* که توسط معلم *t* در کلاس *c* برگزار می‌شود را دارد (*d* یک روز هفته بین شنبه تا پنجشنبه و *p* یک زنگ بین 1 تا 8 است). هر درس در یک زنگ برگزار می‌شود که در میان تمام درس‌های طول هفته، نامی یکتا دارد. مجموعه‌ای

از مرابطه‌های BCNF برای این پایگاه داده طراحی کنید. آیا این مرابطه‌ها 5NF هم هستند؟ 6NF چطور؟ کلیدها کدامند؟

12- بیشتر مثال‌های تجزیه بی‌کم‌وکاست در این فصل نشان می‌دادند که یک رابطه به دقیقا دو پرتو تجزیه می‌شود. آیا ممکن است ضرورتی برای تجزیه به سه پرتو یا بیشتر پیش آید؟

13- بسیاری طراحان پایگاه توصیه می‌کنند که از کلیدهای *ساختگی* و یا *جایگزین* بجای کلید اصلی «طبیعی» استفاده شود. برای مثال می‌توانیم ویژگی SPNO را به مرابطه همیشگی سفارش‌ها اضافه کنیم (بعد از اینکه مطمئن شدیم نام این ویژگی یکتاست) و سپس {SPNO} کلید جایگزین مرابطه کنیم. (توجه، هنوز هم {SNO,PNO} کلید است ولی دیگر کلید اصلی نیست.) بنابراین کلیدهای جایگزین با کلیدهای دیگر رابطه‌ای تفاوتی ندارند مگر الف) فقط دارای یک ویژگی هستند ب) مقدار موجود در آنها فقط به درد جانشینی چیزهایی می‌خورد که نماد آنها هستند (یعنی فقط برای نمایش فلان موجودیت به کار می‌رود و هیچ معنای دیگری ندارد). در حالت ایده‌آل این مقدارها بایستی توسط سیستم ایجاد شوند ولی این موضوع که آنها توسط کاربر و یا سیستم ایجاد شوند هیچ تفاوتی در اساس ایده کلیدهای جانشین بوجود نمی‌آورد. آیا کلیدهای جانشین همان شناسه‌های تاپل‌ها هستند؟ آیا فکر می‌کنید که کلید جانشین ایده خوبی است؟

14- (با تشکر از هیو داروین) من قصد داشتم یک مهمانی بدهم و از این روی، فهرستی اولیه از میهمانان جهت شرکت در میهمانی تهیه کردم. استقبال خوب بود ولی بعضی از میهمانان آمدن‌شان را به آمدن دیگران مشروط کردند. مثلا باب و کال گفتند اگر امی بیاید آنها هم خواهند آمد، هال گفت اگر دان و ایوه هر دو و یا فری بیایند او هم می‌آید، گای گفت در هر صورت می‌آید، جو گفت اگر باب و امی بیایند، می‌آید و به همین ترتیب. پایگاه داده‌ای طراحی کنید که نشان دهد آمدن چه کسی وابسته به چه کسی است؟

15- برای این منظور یک پایگاه داده طراحی کنید. موجودیت‌های مورد نظر کارمندان و برنامه‌نویسان هستند. هر برنامه‌نویسی کارمند است ولی برخی کارمندان برنامه‌نویس نیستند. کارمندان دارای شماره پرسنلی، نام و دستمزد هستند و برنامه‌نویسان دارای (یک) زبان برنامه‌نویسی که در آن ماهرند، هستند. حال اگر هر برنامه‌نویس تعداد دلخواهی زبان برنامه‌نویسی داشته باشد طراحی چه تغییری می‌کند؟

16- فرض کنید که A، B، و C زیرمجموعه‌هایی از عنوان مرابطه R هستند و اجتماع (مجموعه‌ای) این سه مساوی عنوان مرابطه اصلی است. فرض کنید AB نشانه اجتماع (مجموعه‌ای) A و B و همینطور در مورد AC در R وابستگی‌های چندمقداری MVD زیر برقرار است:

$A \twoheadrightarrow B$

$A \twoheadrightarrow C$

(که $A \twoheadrightarrow B$ به صورت «A دو فلش B» یا «B دارای وابستگی چندگانه به A است» خوانده می‌شود اگر و تنها اگر وابستگی پیوندی $\{AB, AC\}$☆ در آن برقرار باشد.) نشان دهید اگر R دارای MVDهای فوق باشد، دارای این خاصیت خواهد بود که اگر دو تاپل $<a,b1,c1>$ و $<a,b2,c2>$ در آن حضور داشته باشند، حتما دارای دو تاپل $<a,b1,c2>$ و $<a,b2,c1>$ نیز خواهد بود.

17- نشان دهید اگر R دارای FD، $A \rightarrow B$ باشد، دارای MVD، $A \twoheadrightarrow B$ نیز خواهد بود.

18- (قضیه فاگین) فرض کنید R مانند تمرین 16 است. ثابت کنید R می‌تواند به صورت بی‌کم‌وکاست به AB و AC تجزیه شود اگر و تنها اگر MVDهای $A \twoheadrightarrow B$ و $A \twoheadrightarrow C$ در آن برقرار باشند.

19- ثابت کنید اگر K کلید R باشد آنگاه $K \twoheadrightarrow A$ برای تمام ویژگی‌های رابطه همانند A برقرار است.

توجه:اینجا برای ارائه تعریفی دیگر مکان مناسبی به نظر می‌رسد. گفته‌شد R در سطح 4NF است اگر و تنها اگر هر MVD با اهمیت بوسیله فراکلید اعمال شود. MVD، $A \rightarrow\rightarrow B$ بی‌اهمیت است اگر و تنها اگر AB مساوی عنوان R و یا A مجموعه‌مادر B باشد. اعمال بوسیله فراکلید در صورتی اتفاق می‌افتد که A فراکلید باشد.

20- مثالی بیاورید که یک رابطه BCNF باشد و 4NF نباشد.

21- پایگاهی برای این منظور طراحی کنید: موجودیت‌ها عبارت‌اند از بازاریاب‌ها، مناطق فروش و محصولات. هر بازاریاب می‌تواند در یک یا چند منطقه فعالیت داشته باشد و هر منطقه یک یا چند بازاریاب دارد، هر بازاریاب یک یا چند محصول برای فروش دارد و هر محصول توسط یک یا چند بازاریاب فروخته می‌شود. هر کالا حتما در هر منطقه فروخته می‌شود ولی دو بازاریاب یک کالا را در یک منطقه نمی‌فروشند. هر بازاریاب در تمامی مناطق زیرپوشش خود کالاهای یکسانی را می‌فروشد.

22- به زبان *توتوریال‌دی* قیدی بنویسید تا JD مرابطه SPJ در شکل 7-5 را برقرار نماید.

23- (تغییر یافته تمرین 21) پایگاهی برای این منظور طراحی کنید: موجودیت‌ها عبارت‌اند از بازاریاب‌ها، مناطق فروش و محصولات. هر بازاریاب می‌تواند در یک یا چند منطقه فعالیت داشته باشد و هر منطقه یک یا چند بازاریاب دارد، هر بازاریاب یک یا چند محصول برای فروش دارد و هر محصول توسط یک یا چند بازاریاب فروخته می‌شود، هر کالا در یک یا چند منطقه فروخته می‌شود و در هر منطقه یک یا چند کالا فروخته می‌شود. در نهایت، اگر بازاریاب r مسئول منطقه a باشد و کالای p در منطقه a فروخته شود و بازاریاب r کالای p را داشته باشد آنگاه، r، p را در a می‌فروشد.

24- کدامیک از عبارت‌های زیر صحیح‌اند؟

الف) هر مرابطه «تمام کلید»، BCNF است.

ب) هر مرابطه «تمام کلید»، 5NF است.
ج) هر مرابطه باینری، BCNF است.

25- در زمان نوشتن این متن، در صنعت بحث‌هایی در مورد ایجاد *پایگاه‌های داده XML* وجود دارد. حال آنکه متن‌های XML ذاتاً سلسله مراتبی‌اند. آیا تصور می‌کنید اشکالاتی که در این فصل در مورد سیستم‌های سلسله‌مراتبی برشمرده شدند، به متن‌های XML هم سرایت می‌کنند؟ شرح دهید؟

26- نمودار ERD تمرین‌های 11، 14، 21 و 23 را رسم کنید. از این تمرین چه نتیجه‌ای می‌گیرید؟

27- یک پایگاه داده دارای دو مرابطه زیر است:

FATHER_OF { X NAME, Y NAME } KEY { X }
MOTHER_OF { X NAME, Y NAME } KEY { X }

گزاره‌نماهای این دو به ترتیب عبارتند از: *X پدر Y است* و *X مادر Y است*. برای سادگی هیچ قیدی غیر از دو قید کلید تعریف نشده است. این طراحی را نقد کنید.

28- این فصل درباره طراحی*داده‌ای* بود و من به مباحثی که در مورد طراحی رابطه‌ای پرداختم. دلایل محکمی وجود دارد مبنی بر این که شما اول بایستی طراحی رابطه‌ای را به خوبی انجام دهید، حتی اگر DBMS مورد نظر شما اصلا رابطه‌ای نباشد. به همین طریق گاهی توصیه می‌شود که برای ایجاد تراکنش‌ها و کوئری‌ها ابتدا بایستی طراحی بوسیله جبر رابطه‌ای انجام شود و سپس به SQL (یا هر زبان دیگری) تبدیل گردد. شما چه نظری دارید؟

فصل هشتم

مدل رابطه‌ای چیست؟

ممکن است تعجب کنید که چرا آخرین فصل کتاب به تشریح مدل رابطه‌ای اختصاص داده شده و «*مدل رابطه‌ای چیست؟*» نامیده شده است. آیا تا اینجا جواب این پرسش را نگرفته‌اید؟

چرا گرفته‌اید. ولی این فصل همه چیز را از اول تاکنون پوشش می‌دهد، یعنی –به منظور استفاده از این کتاب به عنوان مرجع و برای آینده– تعریف دقیقی از مدل رابطه‌ای ارائه می‌کند. مشکل اینجاست که تعریفی که ارائه می‌دهم به راستی دقیق است: بسیار دقیق، آنقدر که تصور می‌کنم درک آن کمی مشکل باشد و اگر چنین نبود آن را در فصل 1 می‌آوردم. (همانطور که برتراند راسل گفته: *نوشته می‌تواند جذاب یا دقیق باشد ولی نمی‌تواند در آن واحد هر دو ویژگی را داشته باشد.*) البته در فصل 1 تعریفی ارائه کردم که «مدل اصلی» نامیده شد ولی بدون رودربایستی تصور نمی‌کنم که این تعریف به اندازه کافی خوب باشد. دلایلی برای این گفته وجود دارد از جمله:

- برای تازه‌کارها بیش از حد دور و دراز بود. (برای فصل مقدمه مناسب بود ولی اکنون می‌خواهم تعریفی بیاورم که هم مختصر و هم دقیق است.)
- من واقعا پیرو این عقیده نیستم که مدل بایستی حتما «ساختار بعلاوه جامعیت بعلاوه دستکاری» داده‌ها را شامل شود. من فکر می‌کنم که چنین تصوری کاملا گمراه‌کننده است.
- «مدل اصلی» چیزهایی داشت که من زیاد با آنها موافق نیستم. بعنوان مثال تهی، قاعده جامعیت وجودی، اینکه لازم است یک کلید اصلی داشته باشیم، و اینکه –من به دلیل مخالفت با آن در این کتاب بدان نپرداختم– دامنه‌ها و نوع‌ها دو چیز متفاوت‌اند. در مورد تهی، کاد مدل رابطه‌ای را در سال 1969 تعریف کرد و تا سال 1979 تهی مطرح نشده بود، به عبارت دیگر مدل به مدت ده سال بدون تهی به درستی (و به نظر من بهتر از بعدا) کار می‌کرد. چگونه است که

بیشتر پیاده‌سازی‌های اولیه بدون هیچگونه تصوری از تهی، به درستی انجام وظیفه می‌کردند.

- نگارش اصلی مدل اندک مواردی را از قلم انداخته است که من تصور می‌کنم واجب و ضروری‌اند. برای مثال مدل اصلی هیچ اشاره‌ای (لااقل هیچ اشاره صریحی) به این موارد نداشته است: گزاره‌نماها، قیدها (بجز قیدهای کلید کاندید و خارجی)، متغیرهای رابطه‌ای، مقایسه‌های رابطه‌ای، استنتاج نوع و مباحث وابسته به آن، عملگرهای رابطه‌ای خاص (خصوصا تغییرنام، گسترش، خلاصه‌سازی، نیم‌پیوند و نیم تفاضل) و رابطه‌های مهم DUM و DEE. (از طرف دیگر، اگر منصف باشیم می‌بینیم که مدل اصلی سنگی جلوی پای این امکانات نیانداخته است و حتی می‌توان گفت که بعضی از این موارد بصورت نارس در مدل اصلی وجود داشته‌اند. به عنوان مثال این موضوع که پیاده‌سازی‌ها بایستی بتوانند از قیدهایی به غیر از کلید کاندید و خارجی پشتیبانی کنند، همواره مورد تاکید بوده است.)

حال بیایید تا بدون معطلی تعریف را بگویم.

تعریف مدل رابطه‌ای

مدل رابطه‌ای از پنج بخش تشکیل شده است:

1- یک مجموعه نامتناهی از *نوع‌های اسکالر*، به ویژه نوع بولی (مقدار درستی).

2- یک نوع*ساز رابطه* و یک تفسیر انتخابی برای نوع‌های ساخته شده بوسیله آن.

3- امکاناتی برای ساخت *متغیرهای رابطه‌ای* که از نوع رابطه ساخته می‌شوند.

4- یک عملگر *انتساب رابطه‌ای* برای انتساب مقادیر به متغیرهای رابطه‌ای.

5- مجموعه‌ای نامتناهی از عملگرهای عمومی رابطه‌ای برای بدست آوردن مقدارهای رابطه‌ای از سایر مقدارهای رابطه‌ای.

نوع‌های اسکالر

بطور کلی نوع‌های اسکالر می‌توانند تعریف شده در سیستم (توکار) و یا تعریف شده بوسیله کاربر باشند؛ بنابراین امکانات تعریف نوع‌های اسکالر شخصی، باید در اختیار کاربر باشند (این نیاز تا حدودی به دلیل نامتناهی بودن نوع‌های اسکالر است). چنین امکانی برای ایجاد عملگرهای اسکالر شخصی نیز بایستی وجود داشته باشد چرا که نوع‌ها، بدون عملگرها به هیچ دردی نمی‌خورند. تنها نوع *ضروری* تعریف شده در سیستم –و بنیادی‌ترین نوع– BOOLEAN است ولی مسلما سیستم‌های واقعی از نوع‌های دیگری هم پشتیبانی می‌کنند (INTEGER، CHAR و غیره).

پشتیبانی از نوع بولی موجب پشتیبانی از عملگرهای منطقی مانند NOT، AND، OR و غیره می‌شود که مقدارهای بولی برمی‌گردانند. عملگر مقایسه‌ای تساوی «=» *باید* برای تمام نوع‌ها، اسکالر و غیراسکالر، وجود داشته باشند چرا که بدون آن حتی نمی‌توان مقدارهای تشکیل دهنده نوع مورد نظر را تعریف کرد. دیگر اینکه مدل معنای این عملگر را نیز تعیین می‌کند. اگر *v1* و *v2* دو مقدار هم‌نوع باشند، *v1=v2* صحیح برمی‌گرداند اگر *v1* و *v2* مقدارهایی کاملا مشابه باشند، در غیر اینصورت غلط باز گردانده می‌شود.

در هر صورت SQL در این مورد رسما شکست می‌خورد (پشتیبانی از «=» برای همه نوع‌ها با معنای تعیین شده در مدل). برای مثال:

- برای نوع‌های توکار CHAR و VARCHAR می‌تواند –در واقع معمول است– که عملگر «=» مقدار صحیح برگرداند، حتی اگر طرفین مقایسه یکسان نباشند.
- برای نوع توکار XML (و نوع‌های توکار BLOB و CLOB در محصولات خاص)، «=» اصلا تعریف نشده است.
- برای نوع‌های تعریف شده توسط کاربر، «=» فقط در صورتی می‌تواند تعریف شود که «<» هم تعریف شده باشد. در مورد برخی نوع‌ها «<» بی‌معناست حتی اگر «=» در آنها تعریف *شده* باشد. پس معنای عملگر اختیاری است و در واقع به تعریف کننده نوع واگذار شده است.
- در مورد جدول‌ها اساسا هیچ عملگر مقایسه‌ای تعریف نشده است.

- آخری، مواردی که طرفین یکسان هستند و «=» صحیح بر نمی‌گرداند،کاملا متداول هستند. این مورد خصوصا در زمانی که طرفین تهی باشند دیده می‌شود.

یکی از عواقب چنین رفتارهایی در SQL، زیرپا گذاشتن *اصل انتساب* است (بخش «چند اصل پایگاه داده» را ببینید).

نوع رابطه

نوع‌ساز رابطه به کاربران این توانایی را می‌دهد که نوع‌های رابطه‌ای منحصر به فرد و مورد نظر خود را بسازند. مانند متغیرهای رابطه‌ای با ویژگی‌هایی که خودشان رابطه‌اند (توضیح بیشتر در فصل 2). یک رابطه خاص از یک نوع خاص و در یک زمینه خاص را می‌توان به صورت مجموعه‌ای از گزاره‌ها تفسیر کرد. تمامی این گزاره‌ها بر پایه یک گزاره‌نما ایجاد شده‌اند و الف)متناظر با عنوان رابطه هستند. ب)با یک تاپل در بدنه رابطه نمایش داده می‌شوند. اگر بحث در مورد مرابطه باشد –وقتی از رابطه صحبت می‌کنیم مانند این است که مقدار فعلی یک مرابطه مورد نظرمان است– گزاره‌نمای مورد نظر گزاره‌نمای مرابطه نامیده می‌شود. اگر یک تاپل بتواند در یک زمان درون یک مرابطه ظاهر شود ولی این اتفاق نیفتد، چنین فرض می‌شود که گزاره متناظر با آن در آن زمان غلط است.

از آنجا که عملگر مقایسه‌ای تساوی «=» برای هر نوعی وجود دارد، می‌توان نتیجه گرفت که این عملگر برای هر نوع خاصی از رابطه نیز موجود است.

متغیرهای رابطه‌ای

همان‌طور که در قسمت قبل دیدید یک کاربرد مهم نوع‌ساز رابطه تعیین نوع متغیرهای رابطه‌ای یا مرابطه‌ها، به هنگام تعریف آنهاست. *تنها* نوعی که در پایگاه‌های داده رابطه‌ای مجاز شمرده می‌شود، مرابطه است (به خصوص متغیرهای اسکالر و تاپلی ممنوع هستند، حتی اگر در برنامه‌هایی که با پایگاه سروکار دارند این نوع‌ها ممنوع نبوده احتمالا مورد نیاز باشند).

پایگاه داده به غیر از مرابطه‌ها هیچ چیز دیگری ندارد. این یکی از برداشت‌هایی است از چیزی که کاد آن را *اصل اطلاعات* نامید، گرچه من تصور نمی‌کنم که او هرگز چنین چیزی صراحتا را بیان کرده باشد ولی در عوض همواره بر چنین اصلی تکیه می‌کرد:

برای نشان دادن اطلاعاتی که (در هر لحظه از زمان) پایگاه داده محتوی آنهاست، فقط یک راه هست: نشان دادن به شکل مقدارهایی که به روشنی در ویژگی‌های تاپل‌های رابطه‌ها وجود دارند. من بارها دیدم که کاد این اصل را به عنوان اصل بنیادی مدل رابطه‌ای مورد استناد قرار داد. ولی چرا این اصل مهم است؟ در فصل 4 گفتم که نوع‌ها و رابطه‌ها برای نمایش داده‌ها در سطح منطقی ضروری هستند. پاسخ این سوال با این موضوع ارتباطی تنگاتنگ دارد. به بیان دیگر مدل رابطه دقیقا همان چیزهایی که به آنها نیاز داریم را در اختیار ما می‌گذارد و هیچ چیز غیر ضروری در آن وجود ندارد.

من کمی بعدتر این بحث را ادامه خواهم داد. بدیهی است که بطور کلی اگر ما n روش مختلف برای نمایش اطلاعات داشته باشیم آنگاه به n دسته عملگر مختلف نیاز خواهیم داشت. مثلا اگر ما آرایه‌ها را در کنار رابطه‌ها داشته باشیم به یک سری کامل عملگرهای آرایه‌ای (و یک سری کامل عملگرهای رابطه‌ای) نیازمند خواهیم بود. اگر n از یک بزرگتر باشد آنگاه تعداد عملگرهایی که باید پیاده‌سازی شوند، حجم مستندات، مطالب آموزشی، زمان یادگیری همگی بیشتر خواهند شد. ولی افزایش عملگرها فقط پیچیدگی را بیشتر می‌کند و قدرت بیشتری را با خود به ارمغان نمی‌آورد! هیچ کار مفیدی نیست که بتوان آن را با n بزرگتر از یک انجام داد و با n مساوی یک امکان‌پذیر نباشد. (و البته در مدل رابطه‌ای n مساوی یک است).

دیگر اینکه مدل رابطه‌ای فقط زیربنایی به شکل رابطه برای نمایش داده‌ها نیست. به قول کاد (بخش بعد «اهداف مدل رابطه‌ای» را ببینید)، این مدل دارای یک *سادگی درویشی* است. نه ترتیبی برای تاپل‌ها، نه ترتیبی برای ویژگی‌ها، نه تاپل‌های تکراری، نه اشاره‌گری و (لااقل به نظر من) نه تهی‌ای. هرگونه بی‌اعتنایی به این خصوصیات در حکم ارائه یک روش دیگر برای نمایش داده‌ها و در نتیجه تعریف عملگرهای جدید خواهد بود. در واقع SQL دلیل زنده این نظر است. برای مثال SQL دارای هشت نوع اجتماع است* در حالی که مدل رابطه‌ای فقط یکی دارد.

---

* در حقیقت نه هشت‌تا که دوازده‌تا. SQL دارای «اجتماع مجموعه چندتایی» است که بر «مجموعه‌ای چندتایی از سطرها» اعمال می‌شود. در این مورد از شش حالت ممکن تنها دو حالت پشتیبانی می‌شوند. چیزی که وضع را بدتر می‌کند این است که SQL اساسا پشتیبانی درستی از اجتماع مجموعه‌های چندتایی ندارد و چیزی که به عنوان اجتماع چند مجموعه‌ای شناخته می‌شود در واقع همان اجتماع پیشرفته است. برای اطلاع بیشتر به کتاب هنر *برنامه نویسی، جلد2: الگوریتم‌های شبه‌عددی* نوشته دونالد نوث مراجعه کنید.

همانطور که دیدید *اصل اطلاعات* واقعا مهم است ولی باید گفت که نام این اصل به سختی مقصود را می‌رساند. نام دیگری که توسط من و هیو داروین پیشنهاد شده است عبارت است از: *اصل نمایش متحدالشکل*.

انتساب رابطه‌ای

همانند عملگر مقایسه «=»، عملگر انتساب «:=» نیز بایستی در همه نوع‌ها وجود داشته باشد. بدون داشتن آن ما قادر نخواهیم بود یک مقدار دلخواه را به متغیری از نوع مورد نظر منتسب کنیم و این امر درمورد نوع رابطه‌ای هم صادق است. خلاصه‌نویسی‌های INSERT، DELETE و UPDATE مجاز و سودمندند ولی آنها فقط خلاصه هستند و جای اصل را نمی‌گیرند. دیگر اینکه انتساب رابطه‌ای بایستی شامل انتساب رابطه‌ای چندتایی نیز باشد.

عملگرهای رابطه‌ای

«عملگرهای عمومی رابطه‌ای» عملگرهایی هستند که در جبر رابطه‌ای وجود دارند (هیچ دلیلی وجود ندارد که کاربران نتوانند در صورت تمایل عملگرهای مورد نظر خود را بسازند).

چنین به نظر می‌رسد که تصور غلطی در مورد هدف جبر رابطه‌ای وجود دارد و بسیاری مردم فکر می‌کنند که تنها می‌توان با آن کوئری نوشت، ولی این واقعیت ندارد. با جبر رابطه‌ای می‌توان برای اهدافی گوناگون *عبارت‌های رابطه‌ای* نوشت که کوئری نویسی فقط یکی از این هدف‌هاست. چند مورد مهم دیگر عبارتند از:

- تعریف ویوها و تصویرهای لحظه‌ای
- تعریف مجموعه‌ای از تاپل‌ها برای درج، حذف و به‌روزرسانی در یک مرابطه خاص (یا کلی‌تر، تعریف مجموعه‌ای از تاپل‌ها جهت انتساب به یک مرابطه خاص)
- تعریف قیدها (در این مورد عبارت رابطه‌ای بخشی از یک عبارت بولی است که غالبا و نه همیشه دارای IS_EMPTY است)

و غیره (منحصر به این موارد نیست).

جبر می‌تواند معیاری برای اندازه‌گیری قدرت زبان‌های پایگاه باشد. یک زبان *رابطه‌ای تمام‌عیار* است اگر و تنها اگر قدرت آن دست‌کم با جبر برابری کند یعنی هر رابطه‌ای که با جبر قابل تعریف است بتواند بوسیله عبارت‌های این زبان نیز تعریف شود. این امر اساس ارزیابی قابلیت‌های زبان‌های مختلف است. اگر زبانی رابطه‌ای تمام‌عیار باشد (با در نظر گرفتن بقیه موارد) می‌توانیم کوئری‌هایی با پیچیدگی دلخواه در آن بنویسیم بدون اینکه نگران حلقه‌ها و خودبازگشتی‌ها باشیم. به بیان دیگر تمام‌عیار بودن زبان رابطه‌ای –اگر در عمل نشود دست‌کم به طور بالقوه– موجب می‌شود کاربران نهایی بتوانند بطور مستقیم و بدون عبور از گردنه بخش‌رایانه‌ها، به پایگاه دسترسی داشته باشند.

اهداف مدل رابطه‌ای

تنها به منظور استفاده از کتاب به عنوان مرجع، تصور می‌کنم بایستی در این فصل جایی برای نظر شخصی کاد در مورد هدف‌های مدل رابطه‌ای‌اش در نظر گرفته شود. فهرست زیر با اندکی ویرایش، برگرفته از یکی از مقالات وی با عنوان «تحقیقات جدید درباره پایگاه‌های داده رابطه‌ای» (مقاله مدعو کنگره IFIP سال 1974) می‌باشد.

1- فراهم کردن سطحی بالا از استقلال داده‌ای

2- ایجاد دیدگاهی همه فهم از داده‌ها با استفاده از یک سادگی درویشی. در یک تشکیلات بزرگ کاربرانی متفاوت - در محدوده‌ای بین تازه‌کاران تا خبرگان علم کامپیوتر- بایستی بتوانند با مدل کار کنند (البته اگر به دلایلی از کار روی بخشی از سیستم منع نشده باشند)

3- ساده کردن وظیفه دشوار DBA

4- معرفی یک زیربنای (هرچند کوچک) نظری برای مدیریت پایگاه داده‌ها (بخشی که متاسفانه فقدان اصول مستحکم و راهبردها در آن کاملا به چشم می‌خورد)

5- ادغام مباحث ذخیره و بازیابی و مدیریت فایل‌ها جهت ایجاد آمادگی سوار شدن بر روی محصولات نامرغوب دنیای تجاری در آینده

6- ارتقا برنامه‌های کاربردی پایگاه داده‌ها به سطحی کاملا جدید. سطحی که در آن بجای پردازش جزء به جزء، با عملگرهایی از جنس مجموعه (یا به طور خاص رابطه) سروکار خواهیم داشت.

در مورد این که مدل رابطه‌ای تا چه حد به این اهداف دست یافته است، قضاوت را به خودتان واگذار می‌کنم. ولی اگر نظر من را بپرسید می‌گویم که؛ بسیار عالی.

چند اصل در ارتباط با پایگاه داده‌ها

در فصل 1 گفتم که من به اصول اهمیت می‌دهم و نه محصولات، و سپس در جاهای مختلف این کتاب با چند اصل روبرو شدیم. به منظور استفاده به عنوان مرجع، آنها را در اینجا فهرست کرده‌ام:

اصل اطلاعات

پایگاه داده هیچ چیز به غیر از مرابطه‌ها ندارد. یعنی در هر لحظه از زمان تمامی اطلاعات پایگاه فقط و فقط به یک صورت می‌توانند نشان داده شوند. مقادیری روشن در جایگاه ویژگی‌های تاپل‌های رابطه.

فرض بسته بودن جهان

اگر تاپل t در یک زمان خاص بتواند در مرابطه R ظاهر شود ولی این اتفاق نیفتد، گزاره p که متناظر با t است در آن زمان غلط فرض می‌شود.

اصل تعویض پذیری

نبایستی هیچ تبعیض غیر ضروری و دلخواهی بین مرابطه‌های پایه و مجازی وجود داشته باشد.

اصل نرمال‌سازی

- مرابطه‌ای که 5NF نیست بایستی به چند پرتو 5NF تجزیه شود.
- تجزیه باید بی‌کم‌وکاست باشد.
- تجزیه بایستی از وابستگی‌ها محافظت کند.
- تمامی پرتوها بایستی برای بازسازی مورد نیاز باشند.
- تجزیه باید به محض رسیدن مرابطه‌ها به سطح 5NF متوقف شود. (این یکی به محکمی چهار مورد قبل نیست. دلایل قاطعی برای رسیدن به سطح 6NF وجود دارد، البته اگر سیستم بتواند بدون پرداخت هزینه‌های جدی، نرمال‌سازی کامل را تحمل کند.)

اصل طراحی متعامد (تفکیکی)

فرض کنید A و B دو مرابطه مجزا در یک پایگاه داده هستند. در تجزیه بی‌کم‌وکاست این دو مرابطه نبایستی پرتوی از A و پرتوی از B وجود داشته باشند، به گونه‌ای که یک تاپل بتواند در هر دو ظاهر شود.

اصل انتساب

بعد از انتساب مقدار v به متغیر V بایستی مقایسه $V=v$ مقدار صحیح برگرداند.

قاعده طلایی

هیچ به روزرسانی‌ای نمی‌تواند موجب غلط شدن هیچکدام از قیدهای پایگاه شود.

اصل تفاوت غیرقابل تشخیص

فرض کنید a و b دو چیز باشند (دو «موجودیت» دلخواه)، اگر هیچ راهی برای تمییز آنها وجود نداشته باشد (هیچ تفاوتی بین آنها قابل مشاهده نباشد)، آنها دو چیز نیستند بلکه یکی هستند.

توجه

من پیش از این *در این کتاب* اصل تفاوت غیرقابل تشخیص را شرح نداده بودم ولی در چند مورد مناسب بصورت ضمنی از آن استفاده کرده‌ام. این اصل می‌تواند به این صورت هم بیان شود: *هر موجودیت دارای شناسه‌ای یکتاست*. در مدل رابطه‌ای برای نمایش چنین شناسه‌هایی – مانند نمایش هر چیز دیگر– فقط یک راه وجود دارد، مقدار ویژگی‌ها (به اصل اطلاعات مراجعه شود). این واقعیت ساده فواید بی‌شماری به همراه دارد.

تفاوت میان مدل رابطه‌ای و مدل‌های دیگر

چیزی را از فصل 4 یادآوری می‌کنم، به عقیده من مدل رابطه‌ای دژی مستحکم، «برحق» و شکست ناپذیر است. من کاملا مطمئنم که تا چند صد سال آینده پایگاه‌های داده همچنان بر اساس مدل رابطه‌ای کاد بنا خواهند شد. چرا؟ چون زیربنای این مدل –نظریه مجموعه‌ها و منطق گزاره‌ای– دژی مستحکم است. منطق گزاره‌ها به 2000 سال قبل و دست‌کم زمان ارسطو برمی‌گردد.

در مورد بقیه مدل‌ها چه می‌توان گفت، مثلا «مدل شی‌گرا» یا «مدل سلسله‌مراتبی» یا «مدل شبکه‌ای» کوداسیل یا «مدل نیمه‌ساخت‌یافته»؟ به نظر من بقیه مدل‌ها اصلا در باغ نیستند. من جدا شک دارم که اصلا اسم آنها را بتوان مدل گذاشت.* مدل‌های سلسله‌مراتبی و شبکه‌ای که اصلا از اول هم مدل نبودند! منظورم مدل تجریدی است. تمام پیاده‌سازی‌های زودهنگام چنین بودند. در واقع آنها زمانی اختراع شدند که *کار از کار گذشته بود*. محصولات سلسله‌مراتبی و شبکه‌ای ابتدا ساخته شدند و سپس این مدل‌ها از روی آنها تعریف شدند. تنها کلمه مودبانه‌ای که در مورد این روش کار می‌توان گفت، کار الابختکی است. درباره مدل‌های شی‌گرا و نیمه‌ساخت‌یافته نیز انتقادات زیادی وارد است تا جایی که به سختی می‌توان آنها را تایید کرد. یک مشکل این است که توافق عمومی بر سر ترکیب این مدل‌ها وجود ندارد. نمی‌توان ادعا کرد که مثلا یک مدل شی‌گرا واحد، روشن و به گونه‌ای که همه آن را قبول داشته باشند، وجود دارد. همینطور در مورد مدل نیمه‌ساخت‌یافته (ممکن است برخی همین ادعا را در مورد واحد نبودن مدل رابطه‌ای نیز داشته باشند! کمی بعدتر پاسخ این ادعا را خواهم داد).

دلیل مهم دیگری که بر اساس آن معتقدم سایر مدل‌ها اساسا شایستگی آن را ندارند که مدل نامیده شوند چنین است. در درجه اول که امیدوارم با این واقعیت غیر قابل تردید موافق باشید که مدل رابطه‌ای فقط یک مدل است و ربطی به پیاده‌سازی ندارد. در مقابل بقیه مدل‌ها در بیشتر موارد هنگام جداکردن مباحث مربوط به مدل از مباحثی که بهتر است در پیاده‌سازی باشند، شکست می‌خورند. کمترین ضرر این وضعیت آن است که فضای بحث مانند آب گل‌آلود می‌شود؛ بنابراین فهم آنها سخت و پیاده‌سازی کنندگان با آزادی به مراتب کمتری –منظور در مقایسه با مدل رابطه‌ای است– می‌توانند برای رسیدن به پیاده‌سازی مورد نظر از ابتکار و خلاقیت خود استفاده کنند.

ادعا کرده‌اند که مدل‌های «رابطه‌ای» مختلفی نیز وجود دارند. یک کتاب جدید (واقعا جدید)† فصلی دارد با عنوان «مدل‌های رابطه‌ای مختلف» که در آن می‌بینیم:

---

* به همین دلیل من همه آنها را در گیومه قرار داده‌ام. من از اینجا به بعد گیومه‌ها را برمی‌دارم چرا که می‌دانم برای خواننده آزار دهنده هستند ولی تصور کنید که آنها هنوز هم داخل گیومه‌اند.

† داده‌ها و پایگاه داده‌ها افکار در عمل، نوشته جو سلکو 1999

مدل رابطه‌ای دیگر مانند هندسه نیست که فقط یک نوع مشخص از آن وجود داشته باشد*[غلط در متن اصلی]*.

و برای محکم‌کاری ادعا می‌کند که شش «مدل رابطه‌ای مختلف» وجود دارد.

من بلافاصله بعد از مشاهده این ادعاها پاسخی بر آن نوشتم که آن پاسخ را به صورت ویرایش شده در اینجا مشاهده می‌کنید:

البته درست این است که چند نوع هندسه مختلف وجود دارد (اقلیدسی، بیضوی، هذلولی و غیره) ولی آیا این مقایسه اساسا درست است؟ آیا اختلافی که «مدل رابطه‌ای مختلف» با هم دارند مانند اختلاف هندسه‌های مختلف است؟ از نظر من پاسخ به این سوال *نه* است. هندسه‌های بیضوی و هذلولی معمولا به صراحت *نااقلیدسی یا غیراقلیدسی* خوانده می‌شوند. اگر این قیاس درست باشد، حداقل پنج مورد از این شش مدل باید *غیررابطه‌ای* باشند و اگر چنین باشد آنها اصلا رابطه‌ای نخواهند بود و نقض غرض می‌شود. (من قبول دارم که برخی از این «شش مدل رابطه‌ای مختلف» مدل اصلا رابطه‌ای نیستند ولی در این صورت این استدلال که آنها «مدل‌های *رابطه‌ای* مختلف» هستند دچار *بی‌ثباتی* می‌شود.)

و سپس ادامه دادم (بازهم با قدری ویرایش):

من تصدیق می‌کنم که کاد در سال‌های 1970 تا 1980 در تعریف خود از مدل رابطه‌ای تجدید نظر کرد. یکی از پی آمدهای این امر آن بود که منتقدین بطور خاص کاد و بطور کلی مدل رابطه‌ای را به «جابجا کردن تیر دروازه » متهم کنند (تغییر قواعد موجود برای غلبه بر مشکلات). برای نمونه مایک استونبریکر نوشت* که «چه کسی می‌تواند چهار نگارش مختلف را تصور کند»:

- نگارش اول: تعریف شده در مقاله CACM سال 1970
- نگارش دوم: تعریف شده در مقاله جایزه تورینگ سال 1981
- نگارش سوم: تعریف شده توسط کاد در 12 قاعده و سیستم نمره‌دهی
- نگارش چهارم: تعریف شده در کتاب کاد

اجازه دهید بحث را موقتا قطع کنم و درباره مراجع توضیح دهم. همه آنها از کاد هستند. مقاله CACM عبارت بود از «یک مدل رابطه‌ای برای داده‌ها در بانک‌های اطلاعاتی مشترک و

* کتاب *انواع خواندن در سیستم‌های پایگاه داده* 1994 مقدمه فصل اول با عنوان «ریشه‌ها»

بزرگ» در *CACM13 شماره 6 ژوئن 1970*. مقاله جایزه تورینگ عبارت بود از «پایگاه رابطه‌ای؛ یک زیربنای کاربردی جهت افزایش بهره‌وری» *CACM25 شماره 2 فوریه 1982*. 12قاعده و سیستم نمره‌دهی در مطالب درج شده در مجله *دنیای کامپیوتر* تحت عناوین « آیا DBMS شما واقعا رابطه‌ای است؟» و «آیا DBMS شما تحت قاعده اجرا می‌شود؟»، 14 و 21 اکتبر 1985 آمده‌اند و در نهایت کتاب کاد به عنوان *مدل رابطه‌ای برای مدیریت پایگاه داده‌ها* ویرایش دوم در سال 1990 چاپ شده است. حال به پاسخ من بازگردیم:

احتمالا به دلیل حساسیت ما به چنین انتقادهایی، هیو داروین و من در کتاب‌مان به نام زیربنای *پایگاه‌های داده در آینده: بیانیه سوم 2000*، سعی کردیم نظر دقیق خود را درباره مدل رابطه‌ای ارائه دهیم* و می‌خواستیم که بیانیه‌مان در این مورد کاملا قاطع باشد. برای ذکر جزئیات شما را به آن کتاب ارجاع می‌دهم. در اینجا فقط می‌خواهم بگویم ما نقش خودمان در این کار را در حد گذاشتن نقطه بالای چند i و گذاشتن خط بالای چند t می‌بینیم، که کاد در کار اصلی خود آنها را بدون نقطه و خط باقی گذاشته بود. قصد ما این نیست که هیچکدام از اصول اساسی دیدگاه کاد را منسوخ کنیم، همانا این بیانیه به تمامی با روح عقاید کاد سازگاری دارد و در ادامه راهی است که او پایه گذاری کرد.

می‌خواهم به موارد قبل نکته‌ای را اضافه کنم که تصور می‌کنم بطور کامل ادعای نویسنده را رد می‌کند. من تایید می‌کنم که چند هندسه مختلف وجود دارند، ولی علت وجود تفاوت در این هندسه‌ها این است که: *آنها از اصول موضوعه متفاوتی آغاز شده‌اند*. در مقابل ما هرگز اصول موضوعه مدل رابطه‌ای را تغییر نداده‌ایم. ما در طی سالها تغییراتی در خود مدل ایجاد *کرده‌ایم*، برای مثال مقایسه‌های رابطه‌ای را اضافه کرده‌ایم در حالی که اصول موضوعه (که برپایه منطق گزاره‌ای کلاسیک هستند) از اولین مقاله کاد تاکنون دست نخورده باقی مانده‌اند. علاوه بر این، به نظر من تمامی تغییرات انجام شده دارای طبیعت اصلاح‌طلبانه بوده‌اند، نه انقلابی. بنابراین ادعا می‌کنم فقط یک مدل رابطه‌ای وجود دارد گر چه این مدل در طی زمان تغییر کرده باشد و احتمالا این تغییرات ادامه داشته باشند. همان‌طور که در فصل یک گفتم این مدل را می‌توان به

* کتاب *پایگاه داده‌ها، نوع‌ها و مدل رابطه‌ای: بیانیه سوم* جانشین این کتاب شد ولی پیام اصلی کتاب دست نخورده باقی ماند.

چشم شاخه کوچکی از ریاضیات نگریست و در نتیجه با اثبات قضایا و کشف نتایج جدید می‌توان شاهد رشد آن بود.

این تغییرات اصلاح‌طلبانه چه هستند؟ برخی از آنها عبارتند از:

- همانطور که گفته شد، مقایسه‌های رابطه‌ای را اضافه کرده‌ایم.
- تفاوت منطقی بین رابطه‌ها و مرابطه‌ها را روشن کرده‌ایم.
- به درک بهتری از جبر رابطه‌ای رسیده‌ایم. اهمیت عملگرهای مختلف و کارکرد آنها و رابطه‌های با درجه صفر روشن شدند و عملگرهای جدیدی (مانند گسترش) معرفی گردیده‌اند.
- درک بهتری از به‌روزرسانی پیدا کرده‌ایم. خصوصا به‌روزرسانی ویوها.
- موضوع فرم اول نرمال را روشن کرده‌ایم، بحث ویژگی‌های با مقدار رابطه نیز در همین راستا پوشش داده شده‌اند.
- بصورت عام به درک بالاتری از موضوع بنیادی قیدهای جامعیت رسیده‌ایم و در موارد خاص به نتایج تئوری خوبی دست یافته‌ایم.
- ارتباط طبیعی بین مرابطه‌ها و منطق گزاره‌ای را روشن کرده‌ایم.
- در نهایت، به درک روشن‌تری از رابطه مدل رابطه‌ای و نظریه نوع رسیده‌ایم (طبیعت دامنه‌ها را روشن کرده‌ایم).

چه کارهایی برای انجام باقی مانده‌اند؟

با وجود همه چیزهایی که گفته شد نباید تصور کنیم که همه کارها انجام شده‌اند و ما نباید پیشرفت را ادامه دهیم. من دست‌کم چهار زمینه (تا حدودی وابسته به هم) را در نقطه شروع و یا نیازمند پیشرفت می‌بینم: پیاده‌سازی، زیرساخت‌ها، سطح بالاتر تجرید و سطح بالاتر واسط‌ها.

پیاده‌سازی

به طریقی می‌توان پیام این کتاب را به گونه‌ای ساده چنین عنوان کرد:

بیایید مدل رابطه‌ای را پیاده‌سازی کنیم!

از همان فصل‌های اولیه مشخص بود که بایستی بینهایت دست و دل باز باشیم تا SQL را یک زبان رابطه‌ای به حساب آوریم و از این رو محصولات SQL فقط تا تقریب اول رابطه‌ای هستند. واقعیت این است که مدل رابطه‌ای هرگز به صورت تجاری، خوب پیاده‌سازی نشده است و کاربران از سودمندی‌های یک محصول رابطه‌ای مناسب بهره‌مند نگردیده‌اند و این یکی از دلایلی است که من و هیو داروین مدتها از وقت خود را صرف بیانیه سوم کردیم. بیانیه سوم –یا به طور خلاصه بیانیه– یک طرح پیشنهادی رسمی است که در آن زیربنایی قوی برای DBMS تشریح شده است. بدون اشاره به این که چه کاری باید واقعا انجام شود، نویسندگان مدل رابطه‌ای را تا جای ممکن با دقت و اختصار تعریف، و الفبای پیاده‌سازی بر این اساس را بیان کرده‌اند. (موضوع اثرات نظریه نوع، بر این مدل نیز بحثی مهم است. به خصوص ارائه یک مدل جامع که در آن ارث‌بری نوع بعنوان نتیجه منطقی نظریه نوع پیش‌بینی شده باشد.)

ما خوشحال می‌شویم اگر ببینیم که *بیانیه* در قالبی مناسب و به صورت تجاری پیاده‌سازی شده است (منظور از ما دارون است و من). ما معتقدیم چنین پیاده‌سازی‌ای به یک زیربنای مستحکم و تغییر ناپذیر نیاز دارد تا ساخت خیلی چیزهای دیگر مانند DBMSهای «شی‌گرا–رابطه‌ای»، DBMSهای «زمانمند»، DBMSهایی که در تعامل با وب هستند و «موتورهای استنتاج» (که بعنوان «سرورهای تجاری–منطقی» نیز شناخته می‌شوند) و بطور کلی نسل آتی DBMSهای همه منظوره، بر آن ممکن شود. ما همچنین معتقدیم که برای پشتیبانی از بقیه موارد مطلوبی که در این بخش پیشنهاد شده‌اند، به یک چهارچوب درست نیاز داریم. به نظر من تلاش برای پیاده‌سازی خارج از این چهارچوب صحیح سخت‌تر است. نقل قولی از ریاضی‌دان سرشناس گریگوری چاندنوسکی: «اگر این کار را به روشی احمقانه انجام دهی، بایستی آن را دوباره انجام دهی» (از مقاله‌ای در نیویورک‌تایمز 24دسامبر 1997).

تکرار می‌کنم، من به دنبال مدلی هستم که به خوبی پیاده‌سازی شود. در فصل قبل تکنولوژی پیاده‌سازی جدید و امیدبخشی بنام *مدل ترابطه‌ای* را معرفی کردم که ظاهرا می‌تواند برای این منظور انتخاب مناسبی باشد. این امکان هم اکنون در دست بررسی است.

زیرساخت‌ها

هنوز هم کارهای قابل توجهی در زمینه زیرساخت‌های تئوری، برای انجام وجود دارد (مسلما نمی‌توان گفت تمامی مسائل تئوری حل شده‌اند). در اینجا سه نمونه مشاهده می‌شود:

- فرض کنید rx یک عبارت رابطه‌ای باشد. رابطه r که نتیجه اجرای rx است دارای قیدی است به نام rc که از قیدهای مرابطه‌هایی که در عبارت rx به کار رفته‌اند، بدست می‌آید. آیا rc با کامپیوتر قابل محاسبه است؟
- آیا می‌توان دانش بیشتری به روند طراحی پایگاه تزریق کرد؟ به خصوص آیا می‌توان برای توصیف افزونگی راهی پیدا کرد؟
- در فصل قبل من پیش‌نویس طرحی را ارائه کردم که برپایه 6NF با مساله گم‌شدگی اطلاعات برخورد می‌شد. این دیدگاه چه مفهومی دارد؟

سطح بالاتر تجرید

یکی از راه‌های ایجاد پیشرفت در زبان‌های کامپیوتری، *بالا بردن سطح تجرید* آنها است. برای مثال در فصل 4 اشاره شد، کلمات آشنای کلید و کلیدخارجی در واقع مختصرنویسی قیدهایی هستند که می‌توانند بصورت قیدهای عمومی طولانی با هر زبان رابطه‌ای تمام عیار مانند توتوریال‌دی نوشته شوند. ولی این مختصرنویسی‌ها *مفیدند*. آنها به غیر از اینکه در نوشتن صرفه‌جویی می‌کنند، به ما این توانایی را می‌دهند که هنگام صحبت از بسته‌هایی استفاده نماییم که چند موضوع مربوط به هم را درخود دارند و به این ترتیب سطح تجرید بالا می‌رود. چنین به نظر می‌رسد که دیدن جنگل برای ما ساده‌تر از دیدن درختان است.

نمونه دیگر جبر رابطه‌ای است. در فصل 5 نشان دادم که بسیاری از عملگرهای جبر رابطه‌ای –شامل آن دسته از عملگرهایی که فراوان به کار می‌روند (حتی اگر ندانیم) مانند نیم‌پیوند– در واقع خلاصه‌نویسی شده چند عملگر دیگر هستند.* عملگرهای معمول دیگری هم بودند که به دلیل کمبود جا در آن فصل توضیح داده نشدند، این صحبت ممکن است در مورد آنها «درست‌تر» باشد. باز هم چیزی که در اینجا اتفاق می‌افتد بالا بردن درجه تجرید یا انتزاع است

* من و دارون در بیانیه نشان دادیم که هر عملگر جبری می‌تواند فقط با دو عملگر اولیه شبیه‌سازی شود، *remove* (که اساس پرتو است) و یکی از دو عملگر *nand* یا *nor*.

(همانطور که زیرروال‌ها و توابع در زبان‌های برنامه‌نویسی شناخته‌شده درجه تجرید را بالا می‌برند).

بالا بردن درجه تجرید در دنیای رابطه‌ای مانند ساختن بنا بر روی زیربنای نظریه رابطه‌ای است. این کار مدل را تغییر نمی‌دهد ولی آن را برای انجام کارهای خاصی سودمندتر می‌کند. در این زمینه یکی از موضوعاتی که در صورت پیشرفت واقعا پر ثمر خواهد بود، پایگاه‌های زمان‌مند است. در کتاب دیگر ما، *داده‌های زمان‌مند و مدل رابطه‌ای* 2003 نوشته هیو داروین، نیکوس لورنتزوس و من –که کار اصلی را لورنتزوس انجام داد– *نوع‌های مدت‌زمانی* بعنوان پایه پشتیبانی از داده‌های زمان‌مند در چهارچوب رابطه‌ای معرفی شدند. برای مثال «رابطه زمان‌مند» شکل 8-1 را در نظر بگیرید، که نشان می‌دهد توزیع‌کننده‌ای مشخص قطعه‌ای مشخص را در مدت زمان مشخصی توزیع کرده است. (می‌توان *d04* را بصورت «روز 4» و *d06* را بصورت «روز 6» خواند و از این رو [*d04*:*d06*] بصورت «در مدت زمان روز 4 تا روز6» خوانده می‌شود). ویژگی *DURING* در این رابطه دارای نوع مدت‌زمان است.

SNO	PNO	DURING
S1	P1	[*d04*:*d06*]
S1	P1	[*d09*:*d10*]
S1	P3	[*d05*:*d10*]
S2	P1	[*d02*:*d04*]
S2	P1	[*d08*:*d10*]
S2	P2	[*d03*:*d03*]
S2	P2	[*d09*:*d10*]
S3	P1	[*d02*:*d05*]
S3	P2	[*d08*:*d10*]
S3	P3	[*d08*:*d10*]

شکل 8-1. یک رابطه با ویژگی مدت‌زمان

پشتیبانی از ویژگی‌های مدت‌زمانی (و همچنین پایگاه‌های زمان‌مند) نیازمند بسیاری موارد از جمله نگارش تعمیم یافته عملگرهای معمولی جبری است. به دلیل اینکه این عملگرها در اینجا کم اهمیت هستند به این عملگرهای تعمیم یافته عملگرهای *U_* می‌گوییم، مانند عملگر *U_restrict*، عملگر *U_join*، عملگر *U_union* و غیره. ولی – نکته اساسی اینجاست– تمامی عملگرهای *U_* بجز مختصرنویسی ترکیب عملگرهای معمولی جبری، چیز دیگری نیستند. دوباره، چیزی که اساسا در اینجا اتفاق می‌افتد بالا بردن سطح تجرید است.

دو نکته دیگر درباره این موضوع، اول برخورد رابطه‌ای ما با داده‌های زمان‌مند تنها به نگارش _U عملگرهای جبر رابطه‌ای محدود نمی‌شود. موارد دیگری هم وجود دارد الف) کلیدها و کلیدهای خارجی بصورت_U ب)عملگرهای مقایسه‌ای بصورت_U ج)نگارش _U از INSERT، DELETE و UPDATE. ولی مجددا می‌گویم تمام این ساختارها مختصر نویسی هستند. دوم، مدل ارث‌بری نوع که در بیانیه شرح داده شده است، در پشتیبانی از زمان‌مندی نقشی حیاتی را بازی می‌کند. یک بار دیگر می‌گویم ما مثالی جامع از تمامی این موارد را بررسی کرده‌ایم.

سطح بالاتر واسط‌ها

راه دیگری هم برای ساخت بنا بر روی مدل رابطه‌ای وجود دارد، بدین معنا که برنامه‌های کاربردی گوناگون می‌توانند با استفاده از واسط رابطه‌ای اجرا شوند و خدمات متنوعی را عرضه نمایند. یک نمونه پشتیبانی از تصمیم، دیگری داده کاوی و دیگری رابط زبان طبیعی است. از نظر کاربران چنین برنامه‌هایی، مدل رابطه‌ای توسط پوششی-دست‌کم تا سطح مشخصی-کاملا پوشانده شده است. (حتی اگر چنین شود و ارتباط اکثر کاربران با پایگاه فقط از طریق واسط صورت گیرد، من باز هم تصور می‌کنم طراحی بایستی بر پایه اصول تغییر ناپذیر مدل رابطه‌ای انجام شود.)

در هر حال فرض کنید که پیاده‌سازی یک برنامه کاربردی به شما واگذار شده است. کدامیک را انتخاب می‌کنید؟ یک DBMS رابطه‌ای و انواع مشابه آن و یا یک DBMS شی‌گرا؟ اگر اولی را انتخاب کردید -که به نظر من بایستی همین کار را بکنید- بعد کدامیک را انتخاب می‌کنید، DBMSای که از مدل رابطه‌ای پشتیبانی می‌کند و یا DBMSای که از SQL پشتیبانی می‌کند؟

نظر من این است که ما روزهایی که SQL تلاش می‌کرد خود را زبانی معرفی کند که کاربران نهایی خودشان می‌توانند از آن استفاده کنند، را پشت سر گذاشته‌ایم و به همین دلیل هم بسیاری، انتقادات بی‌شمار من و دیگران به SQL را بی‌اهمیت تلقی می‌کنند. کاربران واقعی از آن استفاده نمی‌کنند. اینطور نیست؟ فقط برنامه‌نویسان از آن استفاده می‌کنند و در بخش عمده کد اجرایی SQL هم در واقع توسط شخص برنامه‌نویس نوشته نمی‌شود بلکه بوسیله برخی برنامه‌های

واسط تولید شده‌اند. با این حال به عقیده من به همان دلایلی که SQL زبان مبدا بدی است، زبان مقصد بدی هم هست و انتقادهای من همچنان پابرجا هستند.

SQL چه اهمیتی دارد؟

SQL از ایجاد برخی زیربناهای مستحکمی که برای رشد و توسعه در آینده به آنها نیاز داریم، ناتوان است. مدل رابطه‌ای این زیربناها را فراهم کرده است. در بیانیه سوم من و دارون SQL را رد کردیم و گفتیم که به جای آن یک زبان رابطه‌ای درست مانند توتوریال‌دی بایستی هر چه زودتر پیاده‌سازی شود. البته ما آنقدر ناشی نیستیم که تصور کنیم SQL نابود می‌شود، بلکه بنا بر امیدواری ما توتوریال‌دی یا هر زبان درست رابطه‌ای دیگر، زبان برتر پایگاه داده‌ها خواهد شد (طبق نظریه طبیعی تکامل) و SQL هم در آن زمان زبان پایگاه «بازمانده از نسل‌های قبل» خواهد بود. نظیر همین قضیه در زبان‌های برنامه‌نویسی هم وجود دارد، کوبول هرگز نابود نشد (و نخواهد شد) ولی برای برنامه‌سازی تبدیل به زبانی شد که «از نسل‌های قبل باقی مانده است»، چرا که هم اکنون انتخاب‌های بهتری وجود دارند. ما SQL را به چشم مشابه پایگاه داده‌ای کوبول می‌بینیم و آرزو داریم روزی را ببینیم که زبان‌هایی بهتر از آن برای جایگزینی‌اش وجود دارند.

البته می‌دانیم که پایگاه‌ها و برنامه‌های کاربردی SQL سالها با ما بوده‌اند – تفکری غیر از این واقع‌بینانه نخواهد بود– و حال باید توجه خود را به این مساله جلب کنیم که با میراث SQL چه می‌توان کرد. *بیانیه* در این مورد پیشنهاداتی دارد و بطور خاص توصیه می‌کند که SQL بر روی زیربنای رابطه‌ای درست، مجددا بنا شود. در این صورت برنامه‌های کاربردی SQL می‌توانند همچنان به کار خود ادامه دهند. جزئیات بحث در مورد این پیشنهادها از محدوده این بحث خارج‌اند.

خلاصه

همانطور که در فصل مقدمه گفته شد، هدف این کتاب تشریح نظریه رابطه‌ای است، عمدتا برای مردم و به خصوص برای حرفه‌ای‌هایی که چیزهایی می‌دانند ولی این را هم می‌دانند که به درک بیشتری نیاز دارند. در این بخش آخر از فصل آخر کتاب بهتر است مروری داشته باشیم بر موضوعاتی که در طول این کتاب پوشش داده شده‌اند:

- فصل اول، مقدمه، صحنه نمایش را با توصیف (نه چندان رسمی) مدل اصلی چیدم. همانطور که در آن فصل گفتم، هدف من گفتن چیزهایی بود که امیدوارم اکنون آنها را بدانید. در مورد تفاوت منطقی مهمی که بین رابطه‌ها و مرابطه‌ها، بطور کلی مقدارها و متغیرها و بین مدل و پیاده‌سازی، وجود دارد نیز صحبت کردم. مورد آخر ما را به بحث استقلال داده‌ای رهنمون می‌شود.
- فصل دوم، تفاوت رابطه‌ها و نوع‌ها، دلایلی آوردم تا بگویم دامنه‌ها، همان نوع‌ها هستند و تنها نام‌شان عوض شده است، همچنین نوع‌ها می‌توانند به اندازه دلخواه پیچیده باشند. این سوال که چه نوع‌هایی مورد پشتیبانی قرار می‌گیرند ربطی به مدل رابطه‌ای ندارد (نوع‌ها مستقل از جدول‌ها هستند).
- فصل سوم، تاپل‌ها و رابطه‌ها، تعریف دقیقی از این مفاهیم بنیادی ارائه شد و نتایج آنها مورد بررسی قرار گرفتند. همچنین استدلالی واقع‌بینانه در مورد ممنوعیت تکرار و تهی آوردم و رابطه‌های خاص و مهم DUM و DEE را نیز معرفی نمودم.
- فصل چهارم، متغیرهای رابطه‌ای، جایی بود که در آن برای نخستین بار مفهوم گزاره‌نمای مرابطه را ارائه نمودم و نشان دادم که برای نمایش داده‌ها در سطح منطقی که مطلوب ماست، نوع‌ها و رابطه‌ها هر دو لازم و کافی هستند. جزئیات کلیدهای کاندید و خارجی را توضیح دادم و نگاهی دقیق‌تر به ویوها (که به نام مرابطه‌های مجازی هم شناخته شده‌اند)، انداختم.
- فصل پنجم، جبر رابطه‌ای، طولانی‌ترین فصل کتاب بود. من بر اهمیت خاصیت بسته بودن تکیه کردم و مجموعه قواعد استنتاج نوع رابطه را توصیف دادم. تعداد زیادی از عملگرهای جبری مهم را شرح دادم (به خصوص پیوند که اشتراک و ضرب حالات خاصی از آن هستند). یک الگوریتم ادراکی برای

ارزیابی SELECT-FROM-WHERE ارائه کردم. بحث مختصری در مورد رابطه جبر با بهینه‌سازی و همچنین عملگرهای به‌روزرسانی داشتم و عملگرهای مقایسه‌ای را نیز توضیح دادم.

- فصل ششم، قیدهای جامعیت، دو نوع پایه قیدها را شرح دادم، قیدهای نوع و قیدهای پایگاه. همچنین شرح مختصری درباره «نماممکن‌ها»، انتخابگرها و عملگرهای _THE ارائه کردم. بررسی قیدهای نوع بصورت قسمتی از اجرای انتخابگرها انجام می‌شود و قیدهای پایگاه «روی سمیکالن» بررسی می‌شوند. همچنین به تشریح انتساب چندتایی، قاعده طلایی و تفاوت بین صحت و ثبات پرداختم. و همچنین ادعا کردم که قیدها واجب و ضروری هستند (برای من مهم نیست سیستم شما چقدر سریع است، در صورتی که نتوانم به پاسخ‌هایی که به من می‌دهد اعتماد کنم)
- فصل هفتم، نظریه طراحی پایگاه داده‌ها، بر روی نظریه طراحی منطقی پایگاه تمرکز شده است. این نظریه در واقع بخشی از مدل رابطه‌ای نیست ولی برای ساختن در مدل رابطه‌ای از آن استفاده می‌شود. بر این مفهوم تکیه کردم که طراحی منطقی با قیدها و گزاره‌نماها رابطه تنگاتنگی دارد. بخش عمده نظریه طراحی مربوط به کاهش افزونگی است. نرمال‌سازی افزونگی داخلی مرابطه‌ها و تفکیک افزونگی سراسری مرابطه‌ها را کاهش می‌دهد. FDها، BCNF، JDها و 5NF را توضیح دادم مختصری هم از 6NF گفتم. این عقیده را که «این کار در واقع رسمی کردن یک احساس ذاتی است» را موشکافی کردم. لطفا درجه نرمال را کاهش ندهید! تفکیک را شرح دادم و چند توصیه عمومی در مورد طراحی فیزیکی ارائه کردم.
- در نهایت، همین فصل، *مدل رابطه‌ای چیست؟*، به برخی موضوعات مهمی که در رابطه با پرسش عنوان فصل هستند، نگاهی انداخته‌ام. پاسخی دقیق، مختصر و نه لزوما فوری قابل فهم به این پرسش دادم. پنج جز اصلی مدل را یک به یک موشکافی کردم. همچنین اهداف کاد را از مدل رابطه‌ای آوردم و بعد اصول گوناگونی که تاکنون در متن داشتیم را جمع‌بندی کردم –اصل اطلاعات و بقیه

اصول- تا جایی که می‌توانستم سعی کردم نشان دهم که بقیه «مدل‌ها» اصلا در باغ نیستند. هنوز هم مدل رابطه‌ای تنها مدلی‌ست که قبل از انجام پیاده‌سازی، تعریف شده است و تنها مدلی است که به گونه‌ای سخت‌گیرانه هیچ ملاحظه‌ای در مورد پیاده‌سازی ندارد. در پایان راه‌های مختلفی که از طریق آنها می‌توانیم این شاخه از علم را جلو ببریم را، مورد بحث قرار دادم.

تمرین‌ها

این تمرین‌ها مروری بر تمامی مطالب این کتاب -نه فقط این فصل- دارند و برخی از آنها تکرار تمرین‌های فصل‌های قبلی هستند.

1- مدل رابطه‌ای دقیقا چیست؟ شمار زیادی از تفاوت‌هایی -تا جایی که می‌توانید- که بین SQL و مدل رابطه‌ای وجود دارد را، شرح دهید.

2- *اصل اطلاعات* چیست؟ شناسه سطر چگونه آن را نقض می‌کند؟

3- گزاره‌نما چیست؟ بین رابطه‌ها و گزاره‌نماها چه ارتباطی وجود دارد؟

4- آیا تصور می‌کنید رابطه‌ها تخت و مسطح و یا «دوبعدی» هستند؟ پاسخ خود را توضیح دهید.

5- وابستگی پیوندی چیست؟ یک FD به صورت $A \rightarrow B$ در R برقرار است، چه JDای در این مرابطه برقرار خواهد بود؟

6- تفاوت واقعی بین دامنه و رابطه چیست؟

7- تعویق بررسی جامعیت چه اشکالی دارد؟

8- *اصل تعویض‌پذیری* چیست؟

9- معنای واقعی فرم اول نرمال چیست؟

10- «فرم نهایی نرمال» کدام است؟ این فرم از چه نظر نهایی است؟

11- تفاوت رابطه و مرابطه چیست؟

12- *اصل طراحی متعامد (تفکیکی)* چیست؟

13- آیا می‌توان یک عملگر مقایسه‌ای رابطه‌ای، مثلا «/»، تعریف کرد به گونه ای که r/s صحیح برگرداند اگر و تنها اگر r و s مجزا باشند (یعنی هیچ تاپل مشترکی نداشته باشند)؟ پاسخ خود را شرح دهید.

14- چرا ORDER BY یک عملگر رابطه‌ای نیست؟

15- تفاوت میان مرابطه‌های پایه و ویوها در این است که اولی بصورت فیزیکی ذخیره می‌شود ولی دومی چنین نیست. این جمله درست است یا غلط؟

16- چرا مدل رابطه‌ای تهی را مجاز نمی‌شمارد؟ تکرار را چطور؟

17- مدل رابطه‌ای انواع داده‌ای که باید مورد پشتیبانی قرار گیرند را، تعیین می‌کند. درست یا غلط؟

18- کلید اصلی و کلید کاندید چه تفاوتی با هم دارند؟

19- تجزیه بی‌کم‌وکاست چیست؟

20- آیا کاهش درجه نرمال در مواردی ضرورت پیدا می‌کند؟

21- ضرب حالت خاصی از پیوند است. درست یا غلط؟

22- قید نوع چیست؟ قید نوع در چه موقعی باید بررسی شود؟

23- آیا رابطه می‌تواند دارای یک ویژگی از نوع مجموعه باشد؟ آرایه چطور؟ رابطه چطور؟

24- چرا استفاده از کِرسِر (مکان نما) SQL در به‌روزرسانی موجب نقض مدل رابطه‌ای می‌شود؟

25- آیا ممکن است یک رابطه اصلا ویژگی نداشته باشد؟

26- جبر رابطه‌ای بسته است، یعنی چه؟ چرا این بسته بودن اهمیت دارد؟

27- هر مرابطه دلخواه r با یک گزینش خاص و یک پرتو خاص از آن برابر است. این موضوع را شرح دهید.

28- چرا واژه «ویو جامه‌عمل پوشیده»، دارای تناقض است؟

29- بین مدل رابطه‌ای و پیاده‌سازی آن چه تفاوتی وجود دارد؟

30- تفاوت منطقی اساسی که بین پایگاه‌های داده رابطه‌ای و سایر انواع پایگاه داده‌ها وجود دارد، چیست؟ (راهنمایی: *اصل اطلاعات* را به یاد بیاورید.)

31- بین یک DBMS شی‌گرا-رابطه‌ای واقعی و یک DBMS رابطه‌ای واقعی، چه تفاوتی وجود دارد؟

32- آیا گزینش روی اجتماع توزیع می‌شود؟ روی تفاضل چطور؟

33- آیا پرتو روی اجتماع توزیع می‌شود؟ روی تفاضل چطور؟

34- چرا XML با مدل رابطه‌ای سازگاری دارد؟

35- اگر یک رابطه خالی را خلاصه سازی کنیم، چه چیزی به دست خواهیم آورد؟

36- فرض بسته بودن جهان چیست؟

37- عملگرهای نیم‌پیوند و نیم‌تفاضل را تعریف کنید.

38- الف)BCNF ب)5NF ج)6NF را تعریف کنید. آیا تابحال برایتان پیش آمده که نام دوتا از آنها -و همین‌طور 4NF- را با هم اشتباه کنید.

39- آیا پیوند جابجایی‌پذیر است؟ شرکت‌پذیر چطور؟ خودتوان چطور؟

40- هر مرابطه باینری BCNF است. درست یا غلط؟

41- از پیوند چه درکی دارید؟

42- معنای اینکه گفته می‌شود یک FD یا JD بی‌اهمیت است، را بطور رسمی یا غیر رسمی بیان کنید.

43- این که بگوییم یک کلید هیچ ویژگی‌ای ندارد به چه معناست؟ در مورد کلید خارجی چطور؟

45- برای اینکه نتیجه عبارت SELECT در SQL اصلا رابطه نیست، حداقل سه دلیل بیاورید. می‌توانید فرض کنید که در محیطی تعاملی هستیم، یعنی فقط محدود به SELECTهای تکی نیستیم (SELECTهایی که حداکثر یک سطر را بازیابی می‌کنند).

46- اشتراک نوع خاصی از پیوند است. درست یا غلط؟

47- فرض کنید که R از درجه سه است. چند پرتو مجزا (متفاوت) از R وجود دارد؟

48- فرض کنید که R از درجه سه است. R حداکثر چند کلید می‌تواند داشته باشد؟ و حداکثر چند FD؟

49- آیا دو عملگر ضرب دکارتی (رابطه‌ای) و تقسیم (رابطه‌ای)، عکس یکدیگرند؟ (*پرسش کمکی*: چرا تاکید کردم ضرب دکارتی *رابطه‌ای*؟)

50- حاصل پیوند n رابطه چه می‌شود اگر n=3 باشد؟ اگر n=1 باشد چطور؟ اگر n=0 باشد چطور؟

51- چگونه در SQL می‌توان مقایسه‌های رابطه‌ای انجام داد؟

52- آیا تعریف کلید برای ویوها، معنی دارد؟

53- الف) فرض کنید $A \rightarrow B$ در مرابطه R برقرار است. A و B مجموعه‌ای از ویژگی‌ها هستند، حال چه می‌شود اگر یکی از این مجموعه‌ها خالی باشند؟ ب)فرض کنید K کلید مرابطه R است. K مجموعه‌ای از ویژگی‌هاست، حال چه می‌شود اگر این مجموعه خالی باشد؟

54- کدام یک از این هم‌ارزی‌ها مجازند؟

a. *r* INTERSECT *s* ≡ *r* MINUS (*r* MINUS *s*)
b. *r* UNION (*r* INTERSECT *s*) ≡ *r*
c. *r* INTERSECT (*r* UNION *s*) ≡ *r*
d. *r* MATCHING *r* ≡ *r*

55- از واژه «بهینه‌سازی معنایی» چه می‌فهمید؟

56- چرا استفاده از ویژگی‌هایی با مقدار رابطه، معمولا به دور از صرفه هستند (دست‌کم در مرابطه‌های پایه)؟

57- DEE در جبر رابطه‌ای همان نقشی را ایفا می‌کند که 0 در ریاضیات بر عهده دارد. این جمله را توضیح دهید.

58- الف)فرض کنید *Op* یک عملگر رابطه‌ای یک‌تایی است. اگر تنها عملوند *Op*، DUM یا DEE باشد چه اتفاقی می‌افتد؟ ب) فرض کنید *Op* یک عملگر رابطه‌ای دوتایی است. اگر یکی از دو عملوند *Op*، DUM یا DEE باشد چه اتفاقی می‌افتد؟

59- «فقط یک مدل رابطه‌ای وجود دارد» اگر با این ادعا موافق یا مخالف هستید، دلایل خود را بیان کنید.

60- اینکه حرفه‌ای‌های پایگاه داده بتوانند به چنین پرسش‌هایی پاسخ صحیح بدهند چه اهمیتی دارد؟

ضمیمه

کمی درباره منطق

در فصل 1 گفتم جبر رابطه‌ای جایگزینی دارد که حساب رابطه‌ای نامیده می‌شود. حساب رابطه‌ای همیشه همراه با حساب گزاره‌ها و متناسب با پایگاه‌های رابطه‌ای مورد استفاده قرار می‌گیرد. کوئری‌ها و قیدها می‌توانند بجای جبر رابطه‌ای با استفاده از حساب رابطه‌ای نوشته شوند. این کار در مواردی آسان‌تر است و در برخی موارد هم جبر رابطه‌ای ساده‌تر است. به نظر من حداقل یک آشنایی مقدماتی با حساب رابطه‌ای برای حرفه‌ای‌های پایگاه داده‌ها مفید است و از این‌رو در این ضمیمه مطالبی مختصر و کاملا غیر رسمی در این زمینه ارائه می‌نمایم.

بنابر پاراگراف قبل زبان‌های رابطه‌ای بر اساس جبر رابطه‌ای و یا حساب رابطه‌ای ساخته می‌شوند. حال سوال این است که SQL بر مبنای کدامیک ایجاد شده است. پاسخ این است که بخشی از آن بر اساس این دو و بخشی از آن بر اساس هیچکدام. زمانی که SQL برای نخستین بار طراحی شد بنا بر آن گذاشته شد که این زبان با جبر رابطه‌ای و حساب رابطه‌ای متفاوت باشد و در مقابل هدف اصلی آن بود که زبانی بر اساس «زیرکوئری» ساخته شود اما در گذر زمان برخی ویژگی‌های جبر و حساب رابطه‌ای در آن ظاهر شده و زبان برای پذیرفتن آنها توسعه یافته است. پس امروزه زبان SQL در برخی موارد شبیه جبر رابطه‌ای است و در برخی موارد شبیه حساب رابطه‌ای و در برخی موارد به هیچکدام از آنها شباهت ندارد.

گزاره‌ها

یادآوری از فصل 4، گزاره عبارتی است که به گونه‌ای روشن درست یا غلط است. برخی ویژگی‌های مربوط به گزاره‌ها در اینجا ذکر می‌شوند. می‌توان گزاره‌ها را به روش‌های مختلفی با هم ترکیب کرد و گزاره‌های جدیدی از آنها بدست آورد. برای این منظور ارتباط‌دهنده‌های «چنین نیست»، «و»، «یا»، «اگر.. آنگاه...»(شرط) و «اگر و تنها اگر» (دو شرطی) قابل استفاده‌اند. برای مثال در اینجا چند گزاره داریم که میتوان با استفاده از آنها گزاره‌های

جدیدی ایجاد نمود. الف) مشتری یک ستاره است، ب)مریخ دو ماه دارد ج)زهره بین زمین و عطارد قرار دارد.

(مشتری یک ستاره است) یا (مریخ دو ماه دارد)

(مشتری یک ستاره است) و (مشتری یک ستاره است)

(زهره بین زمین و عطارد قرار دارد) و چنین نیست (مشتری یک ستاره است)

اگر (مشتری یک ستاره است) آنگاه (مریخ دو ماه دارد)

بدیهی است که می‌توان ارتباط‌دهنده‌ها را به چشم عملگرهای منطقی نگریست. آنها گزاره‌ها را به عنوان ورودی دریافت می‌کنند و گزاره دیگری را بعنوان خروجی باز می‌گردانند. (من در مثال به این علت از پرانتز استفاده کردم که محدوده ارتباط‌دهنده‌ها را بطور واضح مشخص کنم. در عمل می‌توان با استفاده از قوانین اولویت بسیاری پرانتزها را حذف کرد، البته نوشتن آنها غلط نیست حتی اگر ضروری نباشند.)

گزاره‌ای که هیچ ارتباط‌دهنده‌ای نداشته باشد یک گزاره *ساده* است و گزاره‌ای که ساده نباشد مرکب است. درستی یک گزاره مرکب از درستی گزاره‌های ساده تشکیل دهنده آن و با استفاده از جداول درستی قابل محاسبه است:

p	NOT *p*
T	F
F	T

p *q*	*p* AND *q*	*p* OR *q*	*p* IMPLIES *q*	*p* IFF *q*
T T	T	T	T	T
T F	F	T	F	F
F T	F	T	T	F
F F	F	F	T	T

به دلیل محدودیت جا من بجای درست و غلط از T و F استفاده کرده‌ام. برای مثال گزاره مرکب زیر را در نظر بگیرید.

اگر (مریخ دو ماه دارد) آنگاه (زهره بین زمین و عطارد قرار دارد)

این گزاره به صورت اگر p آنگاه q است. هر دو گزاره ساده صحیح هستند. جدول درستی برای شرط نشان می‌دهد که گزاره اصلی نیز صحیح است. مشاهده می‌کنید که در برخی موارد دچار سردرگمی می‌شویم. در جهان واقعی این مساله که زهره بین زمین و عطارد قرار دارد هیچ ربطی به اینکه مریخ دو ماه دارد، ندارد. در منطق ما به سادگی می‌گوییم که اگر بخواهیم گزاره اگر p آنگاه q درست باشد، اگر q درست باشد بدون توجه به شرایط p شرایط لازم وجود

دارد و اگر p غلط باشد بدون توجه به شرایط q شرایط لازم وجود دارد. بنابراین اگر (مریخ دو ماه دارد) آنگاه (زهره بین زمین و عطارد قرار دارد) درست است! این امر را می‌توان به این صورت تفسیر کرد که اگر به چیزی غیر حقیقی اعتقاد داشته باشی(p) آنگاه می‌توانی به هر چیز معتقد باشی(q). باخدا باش و پادشاهی کن، بی‌خدا باش و هر چه خواهی کن!

توجه

اگر این بحث احساس بدی به شما منتقل نموده، خیلی‌ها مثل شما هستند. منطق‌دانان بهتر از من می‌توانند شما را توجیه کنند که چرا وقتی p غلط و q درست است، p آنگاه q درست می‌شود. این بحث اینجا اهمیت زیادی ندارد و از اهداف این ضمیمه نیست.

از این مطلب نتیجه می‌گیریم که p آنگاه q معادل (چنین نیست p) یا (q) است. این امر نشان می‌دهد که ارتباط‌دهنده‌های منطقی «چنین نیست»، «و»، «یا»، «شرطی» و «دو شرطی»، اولیه نیستند. در حقیقت هر عبارت منطقی می‌تواند از ترکیب مناسب «چنین نیست» با یکی از ارتباط‌دهنده‌های «و» و یا «یا» حاصل شود. (تمرین: این ادعا را بررسی کنید.) و از آن جالب تر اینکه تمامی ارتباط‌دهنده‌ها می‌توانند با یک ارتباط‌دهنده اولیه شبیه سازی شوند. آیا می‌توانید آنرا پیدا کنید؟

گزاره‌نماها

به عبارات زیر توجه کنید:

x یک ستاره است.

x دو ماه دارد.

x دارای n ماه است.

x بین زمین و y قرار دارد.

x بین y و z قرار دارد.

اینها گزاره نیستند چرا که صراحتا درست و یا غلط نمی‌باشند. دلیل این امر آن است که این عبارات دارای *پارامتر* (*جانگهدار* یا *متغیر آزاد*) هستند. مثلا عبارت «x ستاره است» دارای پارامتر x است و تا زمانی که ندانیم x چیست نمی‌توانیم در مورد درستی عبارت اظهار نظر کنیم.

با جایگزینی مقدار آرگومان‌ها می‌توان از چنین عباراتی گزاره بدست آورد (تکرار از فصل 4). مثلا با جایگزینی خورشید بجای x به عبارت «خورشید یک ستاره است» می‌رسیم که گزاره است چرا که درست یا غلط (البته درست) است. تکرار می‌کنم عبارت اصلی «x ستاره است» گزاره نیست. بلکه گزاره‌نمایی است که مانند یک تابع به هنگام اجرا مقدار درست یا غلط را بازمی‌گرداند. این تابع مانند همه تابع‌ها دارای چند پارامتر است که به هنگام فراخوانی مقدارهایی جایگزین این پارامترها می‌شوند. این جایگزینی گزاره‌نما را تبدیل به گزاره می‌کند و می‌گوییم آرگومان‌ها گزاره‌نما را قانع می‌کنند اگر و تنها اگر با استفاده از آنها یک گزاره درست ساخته شود. مثلا خورشید گزاره‌نمای «x ستاره است» را قانع می‌کند ولی ماه آنرا قانع نمی‌کند.

به عنوان یک حاشیه، منطق‌دانان به جای فراخوانی گزاره‌نما از کلمه شناساندن استفاده می‌کنند و به همین دلیل این کلمه عمومی‌تر است. به هر حال من همچنان از کلمه فراخوانی استفاده خواهم کرد.

این نکته را هم از فصل 4 یادآوری می‌کنم که گزاره یک گزاره‌نمای خراب شده است. برای توضیح بیشتر گزاره‌نمایی است که مجموعه پارامترهایش خالی است (و تابع همیشه و با هر بار اجرا نتیجه یکسانی را –درست یا غلط– برمی‌گرداند). به بیان دیگر همه گزاره‌ها گزاره‌نما هستند ولی اکثر گزاره‌نماها گزاره نیستند.

حال به مثال «x دارای n ماه است» توجه کنید. این گزاره‌نما دارای دو پارامتر x و n است. جایگزینی مریخ بجای x و 2 بجای n یک گزاره درست به بار می‌آورد ولی جایگزینی زمین بجای x و 2 بجای n یک گزاره غلط تولید می‌کند.

گزاره‌نماها را می‌توان به سادگی بر اساس کاردینالیتی پارامترها دسته بندی کرد. زمانی که از گزاره‌نمای n–جایی صحبت می‌کنیم منظور گزاره‌نمایی با دقیقا n پارامتر است. مثلا «x بین y و z قرار دارد» یک گزاره‌نمای 3–جایی و «x دارای n ماه است» یک گزاره‌نمای 2–جایی است.گزاره یک گزاره‌نمای 0–جایی است.

توجه

یک گزاره‌نمای n–جایی یا nتایی نیز نامیده می‌شود. اگر n=1 آنگاه یکتایی و اگر n=2 باشد دوتایی خوانده می‌شود.

اگر تعدادی گزاره‌نما داشته باشیم می‌توانیم آنها را به صورت‌های مختلف و با استفاده از ارتباط‌دهنده‌های منطقی که مورد بحث قرار گرفتند، مانند چنین نیست، و، یا و غیره با هم ترکیب نموده و گزاره‌نماهای جدیدی بسازیم (به بیان دیگر این ارتباط‌دهنده‌ها، عملگرهای منطقی هستند که می‌توانند بر روی گزاره‌نماها –نه فقط گزاره‌ها– عمل کنند). گزاره‌نمایی که ارتباط‌دهنده‌ای نداشته باشد ساده، و در غیر اینصورت مرکب نامیده می‌شود. در اینجا یک مثال ساده در مورد گزاره‌نماهای مرکب می‌بینید:

(x یک ستاره است) یا (x بین زمین و y قرار دارد)

این گزاره‌نما دوتایی است. نه به این دلیل که از دو گزاره‌نمای ساده تشکیل شده است، بلکه به این خاطر که دارای دو پارامتر است. x و y.

سورها

در بخش قبل نشان دادم که یکی از راه‌های تبدیل گزاره‌نما به گزاره، فراخوانی آنها با مشخص نمودن مقدار آرگومان‌هاست. راه دیگری هم برای این کار وجود دارد که استفاده از سورهاست. فرض کنید $p(x)$ یک گزاره‌نمای دوتایی باشد (من برای وضوح x که تنها پارامتر آن است را صراحتا نشان داده‌ام). آنگاه:

وجوددارد x، ($p(x)$)

یک گزاره است و معنای آن این است که: «حداقل یک مقدار a وجود دارد که با جایگزینی آن به جای x، $p(a)$ گزاره‌ای صحیح خواهد شد» (به عبارت دیگر مقدار a گزاره‌نمای p را قانع می‌کند). مثلا اگر p گزاره‌نمای «x یک منطق‌دان است» باشد آنگاه:

وجوددارد x، (x یک منطق‌دان است)

یک گزاره است که اتفاقا درست هم هست (مثلا اگر a برتراند راسل فرض شود).

این عبارت:

هرچه‌باشد x، ($p(x)$)

هم یک گزاره است و بدین معناست که «با تمامی مقادیر ممکن a که جایگزین x می‌شود، $p(a)$ گزاره ای صحیح خواهد شد» (به عبارت دیگر تمامی مقادیر a گزاره‌نمای p را قانع می‌کند). برای مثال اگر p مجددا گزاره‌نمای «x یک منطق‌دان است» باشد آنگاه:

هرچه‌باشد x، (x یک منطق‌دان است)

یک گزاره است که اتفاقا غلط است (مثلا اگر a جورج دبلیو بوش فرض شود).

توجه داشته باشید که برای نشان دادن درستی یک گزاره «وجود دارد» و یا نشان دادن غلط بودن یک گزاره «هرچه‌باشد»، تنها یک مثال کفایت می‌کند. همچنین در هر دو مورد پارامتر بایستی لزوما «داخل محدوده» مقادیر ممکن باشد (مجموعه تمامی انسان‌ها در این مثال). من در بخش «قیدهای پایگاه » دوباره به این موضوع خواهم پرداخت.

برای عبارت‌های وجوددارد x و هرچه‌باشد x در بحث فوق از اصطلاح «سور» استفاده می‌شود. سورهای وجوددارد، سور وجودی و سورهای هرچه‌باشد سور عمومی نامیده می‌شوند. (در متون علم منطق معمولا از E معکوس برای سور وجودی از A سروته برای سور عمومی استفاده می‌شود. من به دلایل چاپی و فارغ از مساله خوانایی از وجوددارد و هرچه‌باشد استفاده کرده‌ام.)

مثالی دیگر، فرض کنید q یک گزاره‌نمای دوتایی به شکل «x بلندتر از yاست» و «یک سور وجودی بر روی x» است. درک می‌کنیم:

وجوددارد x (x بلندتر از yاست)

این عبارت یک گزاره نیست، چرا که صحیح یا غلط بودن آن صراحتا نامعلوم است. این عبارت در حقیقت یک گزاره‌نمای یکتایی است و یک پارامتر دارد. y. اگر این گزاره‌نما را با آرگومان استیو فراخوانی کنیم:

وجوددارد x (x بلندتر از استیو است)

این عبارت یک گزاره *است* (و اگر حداقل یک نفر مثل آرنولد بلندتر از استیو وجود داشته باشد، این گزاره درست است). راه دیگر بدست آوردن گزاره از گزاره‌نمای اصلی این است که *هر دو* پارامتر x و y را سوری نماییم. مثلا:

وجوددارد x (وجوددارد y (x بلندتر از yاست))

این هم یک گزاره است و در صورتی غلط است که هیچ کس بلندتر از دیگری نباشد و در غیر این صورت صحیح است. (در این مورد فکر کنید!).

از این مثال چندین درس می‌توان گرفت:

- برای اینکه از یک گزاره‌نمای nتایی به یک گزاره برسیم لازم است بر روی هر یک از پارامترها سور زده شود. بصورت کلی‌تر اگر روی m پارامتر سور زده شود و$(m >> n)$ آنگاه به یک گزاره‌نمای kتایی خواهیم رسید به گونه ای که $k=m-n$.
- بیایید یک لحظه بر سور وجودی تمرکز کنیم. ظاهرا با زدن «سور بر روی همه چیز» در این مثال دو گزاره مختلف حاصل می‌شود:

  وجوددارد x (وجوددارد y (x بلندتر از y است))

  وجوددارد y (وجوددارد x (x بلندتر از y است))

  ظاهرا بدیهی است اما در هر حال، هر دو عبارات به یک معنا هستند: «دو نفر به نام‌های x و y وجود دارند که x از y بلندتر است.» بطور کلی‌تر درک یک سلسله از سورها که همگی عمومی یا همگی وجودی هستند، آسان است و ترتیب نوشتن هیچ تغییری در معنا نمی‌دهد. از این رو می‌توانیم از پرانتزهای غیر ضروری صرفنظر کنیم.

  وجوددارد y وجوددارد x (x بلندتر از y است)

  در مقابل برای سورهای غیر همنوع ترتیب اهمیت دارد (نکته بعد را ببینید).
- می‌توان وجوددارد و هرچه‌باشد را در هر عبارتی که شامل «سور بر روی همه چیز است» به کار برد. در مثال شش گزاره مجزا وجود دارد که می‌توانند با سور همراه شوند.(در حقیقت هشت تا. ولی دو گزاره با توجه به خاصیت قبل حذف می‌گردند.) من شرح مختصری برای هر مورد نوشته‌ام. در این مورد فرض کرده‌ام که لااقل دو نفر آدم در «جهان» وجود دارد. در مورد این فرض بعدا بحث خواهم کرد (در مبحث محدوده خالی بخش مطالبی بیشتر در مورد سورها).

وجوددارد x (وجوددارد y (x بلندتر از y است))

معنی: کسی هست که قدش بلندتر از کس دیگری است، صحیح، مگر اینکه همه هم قد باشند.

وجوددارد x (هرچه‌باشد y (x بلندتر از y است))

معنی: کسی هست که قدش از همه (از جمله خودش!) بلندتر است. بدیهی است غلط.

هرچه‌باشد x (وجوددارد y (x بلندتر از y است))

معنی: هر کس را در نظر بگیریم قدش از برخی بلندتر است. بدیهی است، غلط.

وجوددارد y (هرچه‌باشد x (x بلندتر از y است))

معنی: کسی هست که از همه (از جمله خودش!) کوتاهتر است. بدیهی است، غلط.

هرچه‌باشد y (وجوددارد x (x بلندتر از y است))

معنی: هر کس را در نظر بگیریم قدش از برخی کوتاه‌تر است. بدیهی است، غلط.

هرچه‌باشد x (هرچه‌باشد y (x بلندتر از y است))

معنی: هر کسی بلندتر از هر کسی است. بدیهی است، غلط.

- در نهایت (شاید نیازی به گفتن نباشد)، پنج مورد از شش عبارت فوق غلط هستند. این به آن معنا نیست که آنها با هم از نظر معنایی معادل‌اند و در حقیقت هیچکدام از آنها با دیگری هم معنی نیستند.*

متغیرهای آزاد و مقید

پیش‌تر گفتم اصطلاح دیگری که برای پارامتر وجود دارد، *متغیر آزاد* است. سور زدن بر روی متغیرهای آزاد آنها را به *متغیرهای مقید* تبدیل می‌کند. مثلا دوباره گزاره‌نمای 2-جایی q را از بخش قبل، در نظر بگیرید:

x بلندتر از y است

پارامترهای x و y در اینجا متغیرهای آزادند. با زدن سور وجودی روی x به این نتیجه می‌رسیم:

وجوددارد x (x بلندتر از y است)

در اینجا y هنوز آزاد است ولی x مقید شده است. با زدن سور کامل:

وجوددارد x وجوددارد y (x بلندتر از y است)

x و y مقیداند و هیچ متغیر آزادی باقی نمانده است (گزاره‌نما تبدیل به گزاره شده است).

در برنامه نویسی متغیر آزاد معادل پارامتر شناخته می‌شود. در مقابل متغیرهای مقید جایی در برنامه نویسی ندارند و اما در اینجا هم نقشی مانند مترسک دارند و تنها وظیفه‌شان این است که ارتباط گزاره‌نمای داخل پرانتز را با سورهای خارج از آن برقرار کند. برای مثال گزاره‌نمای(در حقیقت گزاره) ساده زیر را در نظر بگیرید:

وجوددارد x ($x>3$)

این گزاره فقط ادعا دارد که عددی صحیح بزرگتر از سه وجود دارد. (من فرض کردم در اینجا x «در محدوده» اعداد صحیح است. مجددا، این نکته را در بخش «قیدهای پایگاه » شرح

* برای تمرین گزاره‌نمای «x به y سیلی زده است» را در نظر بگیرید و معنای آنرا در حالات مختلف بنویسید. با انجام این تمرین تفاوت حالت‌های مختلف کاملا مشخص می‌شود. مترجم

خواهم داد.) توجه، اگر هر متغیر دیگری جایگزین دو x این عبارت شود هیچ تفاوتی در معنای گزاره به وجود نخواهد آمد. به زبان دیگر گزاره:

وجوددارد y $(y>3)$

با گزاره قبل دقیقا هم معناست.

حال گزاره‌نمای زیر را ملاحظه کنید:

وجوددارد x $(x>3)$ و $x<0$

در اینجا سه x هست ولی هر سه به یک مفهوم اشاره نمی‌کنند. دوتای اول مقید هستند و می‌توانند با چیز دیگری مثل y جایگزین شوند بدون اینکه تغییری در معنا به وجود آید. ولی سومی آزاد است و نمی‌توان آن‌را بدون عوارض جانبی تغییر داد. پس در مورد دو گزاره‌نمای زیر می‌توان گفت که اولی معادل گزاره‌نمای اصلی است ولی دومی با آن تفاوت دارد:

وجوددارد y $(y>3)$ و $x<0$

وجوددارد y $(y>3)$ و $y<0$

در انتها تاکید می‌کنم که اکنون می‌توانیم با تعریف (دوباره) گزاره‌نماها، آنها را به گزاره تبدیل کنیم که در این صورت تمامی متغیرها مقید خواهند شد.

مطالبی بیشتر در مورد سورها

در این بخش شماری از موضوعات گوناگون درباره سورها مورد بحث قرار خواهند گرفت.

نیازی به هر دو سور نداریم

احتمالا اکنون می‌دانید که ما واقعا به هر دو سورهای «وجوددارد» و «هرچه‌باشد» نیازی نداریم. دلیل این امر آن است که عبارت دارای «وجوددارد» را می‌توان با «هرچه‌باشد» هم نشان دهد و بالعکس. برای مثال دوباره به این گزاره‌نما توجه کنید.

وجوددارد x (x بلندتر از استیو است)

«کسی هست که بلندتر از استیو باشد.» راه دیگر اعلام این موضوع:

چنین نیست (هرچه‌باشد x (چنین نیست (x بلندتر از استیو است)))

«چنین نیست که کسی بلندتر از استیو نباشد.» بطور کلی:

وجوددارد x ($p(x)$)

از نظر منطقی معادل این عبارت است:

چنین نیست (هرچه‌باشد x (چنین نیست ($p(x)$)))

(گزاره‌نمای p می‌تواند غیر از x پارامترهای دیگری نیز داشته باشد.) به همین ترتیب عبارت:

هرچه‌باشد x ($p(x)$)

از نظر منطقی معادل این عبارت است:

چنین نیست (وجوددارد x (چنین نیست ($p(x)$)))

(که مجددا، گزاره‌نمای p می‌تواند غیر از x پارامترهای دیگری نیز داشته باشد.)

پیامد این بحث این است که لزومی ندارد زبان رسمی از هر دو کلمه «وجوددارد» و «هرچه‌باشد» پشتیبانی نماید ولی حضور این دو، کار را در عمل بسیار راحت‌تر می‌نمایند. دلیل این امر آن است که استفاده از وجوددارد در برخی عبارات و استفاده از هرچه‌باشد در برخی دیگر از عبارات «طبیعی‌تر» است. برای مثال SQL از وجوددارد (EXISTS) پشتیبانی می‌کند ولی از هرچه‌باشد (FORALL) پشتیبانی نمی‌کند. در نتیجه برخی کوئری‌ها در SQL بسیار بد عبارت‌نویسی می‌شوند. مثلا کوئری «توزیع‌کنندگانی را بده که تمام قطعات را عرضه می‌کنند.»، در منطق می‌تواند به سادگی بدین صورت نوشته شود:

s WHERE FORALL *p* EXISTS *sp* (*s*.SNO = *sp*.SNO AND *sp*.PNO = *p*.PNO)

(«توزیع‌کنندگانی مانند s را بده که هر چه باشد قطعه p، وجود دارد سفارش sp، که رابطه s و p را برقرار می‌کند.» متغیرهای s، p، و sp به ترتیب در محدوده مقادیر فعلی S، P و SP هستند.) ولی در SQL کوئری به این شکل است:

```
SELECT s.*
FROM   S AS s
WHERE  NOT EXISTS
   ( SELECT p.*
     FROM   P AS p
     WHERE  NOT EXISTS
        ( SELECT sp.*
          FROM   SP AS sp
          WHERE  s.SNO = sp.SNO
          AND    sp.PNO = p.PNO ) )
```

«توزیع‌کنندگان s را بده که قطعه‌ای موجود نباشد p که سفارشی برای آن وجود نداشته باشد sp که رابطه s و p را برقرار می‌کند.» یک نقیض (چنین نیست) به اندازه کافی بد هست (بسیاری کاربران با آن مشکل دارند) اما دو نقیض -مثل این کوئری SQL- دیگر قوز بالا قوز است.

محدوده‌های خالی

دوباره این عبارت‌ها را در نظر بگیرید:

وجوددارد x ($p(x)$)

و:

چنین نیست (هرچه‌باشد x (چنین نیست ($p(x)$)))

که از نظر منطقی معادل یکدیگرند. همانطور که بارها اشاره کردم مقدار x بایستی «داخل محدوده» مقادیر مجاز باشد. حال تصور کنید که مجموعه مقادیر مجاز خالی باشد (مثلا مجموعه افراد بلندتر از 15 متر). آنگاه:

- سور وجودداردx بطور واضح غلط خواهد بود. (چرا که اصلا چنین x ای وجود ندارد.)
- عبارت وجوددارد x ($p(x)$)، «xای وجوددارد که $p(x)$ را تبدیل به صحیح می‌کند»، نیز غلط می‌شود بدون توجه به اینکه $p(x)$ چه باشد. مثلا عبارت «شخص بلندتر از 15متری وجود دارد که در IBM کار می‌کند» غلط است(تعجبی ندارد).
- بنابراین نقیض عبارت یعنی چنین نیست که وجوددارد $p(x)$، که معادل هرچه‌باشد x (چنین‌نیست که $p(x)$)، همواره درست است (بازهم بدون توجه به $p(x)$). مثلا عبارت «تمام افراد بلندتر از 15 متر در IBM کار نمی‌کنند» و یا مصطلح‌تر «هیچ فرد بلندتر 15 متری در IBM کار نمی‌کند»، صحیح است (بازهم تعجبی ندارد).

- اگر $p(x)$ گزاره‌ای دلخواه باشد آنگاه $not(p(x))$ هم گزاره‌ای دلخواه خواهد بود. حال نتیجه تعجب‌آور است. عبارت هرچه‌باشدx (....) در هر صورت و بدون توجه به آنچه درون پرانتزهاست صحیح خواهد بود. بنابراین عبارت «تمام افراد بلندتر از 15 متر در IBM کار می‌کنند.» نیز صحیح است! چرا که هیچ فرد بلندتر از 15 متری وجود ندارد.

پیاده‌سازی این مساله شما را به یک نتیجه عجیب می‌رساند (البته اگر منطق ندانید). مثلا کوئری زیر:

```
s WHERE FORALL p
      ( IF p.COLOR = 'Purple'
        THEN EXISTS sp ( s.SNO = sp.SNO AND sp.PNO = p.PNO ) )
```

«توزیع‌کنندگانی را بده که قطعه‌های بنفش توزیع می‌کنند.» اگر قطعه بنفشی وجود نداشته باشد، تمام توزیع‌کنندگان را بازمی‌گرداند. تمرین: این عبارت را به زبان SQL بنویسید.

تعریف وجوددارد و هرچه‌باشد

ممکن است حدس زده باشید که وجوددارد و هرچه‌باشد می‌توانند به‌ترتیب بصورت «یای مکرر» و «و مکرر» تعریف شوند. ابتدا در مورد وجوددارد صحبت می‌کنم. فرض کنید $p(x)$ گزاره‌نمایی با پارامتر x است و x در محدوده $X = \{x1, x2,..., xn\}$ می‌باشد. در این صورت:

وجوددارد x $(p(x))$

یک گزاره است و معادل (و خلاصه شده) گزاره زیر می‌باشد:

غلط یا $p(x_1)$ یا $p(x_2)$ یا..... یا $p(x_n)$

غلط زمانی به کار می‌آید که X خالی باشد (همانطور که می‌دانید). مثلا اگر $p(x)$ عبارت «x ماه دارد» و X مجموعه {عطارد، زهره، زمین، مریخ} باشد آنگاه عبارت وجوددارد x $(p(x))$ تبدیل به وجوددارد x (x ماه دارد) می‌شود که خلاصه‌ای است از عبارت:

غلط یا (عطارد ماه دارد) یا (زهره ماه دارد) یا (زمین ماه دارد) یا (مریخ ماه دارد)

که صحیح است چرا که (مثلا) «مریخ ماه دارد» درست است. به همین ترتیب می‌توان گفت:

هرچه‌باشد x $(p(x))$

یک گزاره‌نما است و معادل (و خلاصه شده) این گزاره‌نما می‌باشد:

صحیح و $p(x_1)$ و $p(x_2)$ و..... و $p(x_n)$

صحیح زمانی به کار می‌آید که X خالی باشد (مجددا همانطور که می‌دانید). اگر $p(x)$ و X مانند قسمت قبل باشند آنگاه هرچه‌باشد x $(p(x))$ تبدیل به هرچه‌باشد x (x ماه دارد) می‌شود که خلاصه‌ای است از عبارت:

صحیح و (عطارد ماه دارد) و (زهره ماه دارد) و (زمین ماه دارد) و (مریخ ماه دارد)

که غلط است چرا که (مثلا) «زهره ماه دارد» غلط است.

یک حاشیه، همانطور که با مثال دقیقا نشان دادم وجوددارد و هرچه‌باشد به صورت «یا» و «و» مکرر تعریف می‌شوند. بدین معنا که هر گزاره دارای سور معادل گزاره‌ای است که سور ندارد. حال ممکن است این سوال پیش آید که پس سورها واقعا به چه درد می‌خورند؟ آیا این همه دعوا سر لحاف ملانصرالدین بود؟ پاسخ این است که: ما می‌توانیم سورها را به صورت «و» و «یا» مکرر تعریف کنیم *فقط به این علت که – خوشبختانه–، با مجموعه‌ای متناهی سروکار داریم* (ما در قلمرو کامپیوتر کار می‌کنیم و کامپیوترها متناهی و محدود هستند). در منطق اصیل ما چنین پیش شرطی نداریم و طبیعتاً این تعریف‌ها بی اعتبار خواهند بود.

فکر می‌کنم باید این نکته را هم اضافه کنم که حتی اگر ما همیشه با مجموعه‌های محدود سروکار داشته باشیم و وجوددارد و هرچه‌باشد فقط برای خلاصه‌نویسی باشند، باز هم خلاصه‌نویسی‌های بسیار خوبی هستند! خود من تمایلی ندارم بسیاری کوئری‌ها را بدون استفاده از سورها و صرفا بوسیله «و» و «یا» عبارت‌نویسی کنم.

انواع دیگر سورها

درست است که وجوددارد و هرچه‌باشد مهمترین سورها هستند ولی سورهای دیگری هم وجود دارند. دلیلی وجود ندارد که سوری به این شکل نداشته باشیم:

حداقل سه x وجود دارد که..

و یا

اکثر x ها چنین اند که..

و یا

تعداد زوجی x وجود دارد که..

(و غیره).

یکی از مهمترین آنها «دقیقا یک x وجود دارد که...» است. من از کلمه یگانه‌است برای این منظور استفاده خواهم کرد. در اینجا چند مثال را مشاهده می‌کنید:

یگانه‌است x (x از آرنولد بلندتر است)

معنی: دقیقا یک نفر هست که از آرنولد بلندتر است. احتمالا غلط.

یگانه‌است x (x دارای کد ملی y است)

معنی: دقیقا یک نفر هست که کد ملی‌اش y باشد (y پارامتر است). نمی‌توان در مورد درستی عبارت اظهار نظر کرد چرا که این یک گزاره‌نماست نه یک گزاره.

هرچه‌باشد y یگانه‌است x (x دارای کد ملی y است)

معنی: کد ملی هر فرد یکتاست. (فرض کرده‌ام که y در محدوده کدهای ملی واگذار شده قرار دارد نه تمامی اعداد ممکن. با این حال آیا این گزاره صحیح است؟)

به عنوان یک تمرین دیگر، این عبارت به چه معناست؟

هرچه‌باشد x یگانه‌است y (x دارای کد ملی y است)

قیدهای پایگاه

حال می‌خواهم نشان‌دهم که این موارد چگونه در عبارت‌نویسی قیدهای پایگاه به کار می‌آیند. برای این منظور از مثال‌های فصل 6 استفاده کرده‌ام. در ضمن توجه داشته باشید که عبارات نوشته شده تنها حالت‌های ممکن نیستند.

مثال 1

مقدارهای رتبه فقط می‌توانند بین 1 تا 100 باشند.

```
CONSTRAINT C1
   FORALL s ( IF s ∈ S THEN s.STATUS ≥ 1 AND s.STATUS ≤
100 ) ;
```

«هرچه‌باشد s، اگر s تاپلی در مرابطه S باشد آنگاه مقدار STATUS آن بایستی در محدوده مقرر باشد». در فصل 5 گفته شد «$s \in S$» بصورت «s درون S [است]» خوانده می‌شود.

مطمئنم که اکنون می‌توانید قیدهای مختلفی که همگی به شکل گزاره زیر هستند را شناسایی کنید:

FORALL t (IF $t \in R$ THEN p)

(که p یک گزاره است). می‌توان این گزاره را به این صورت خلاصه کرد:

FORALL $t \in R$ (p)

مقید بودن متغیر t *تنوع* مرابطه R (در حال حاضر) را مشخص می‌کند و عبارت هرچه‌باشد $R \in t$ می‌گوید محدوده متناظر را نشان می‌دهد. با استفاده از این خلاصه نویسی C1 به این صورت در می‌آید:

```
CONSTRAINT C1
  FORALL s ∈ S ( s.STATUS ≥ 0 AND s.STATUS ≤ 100 ) ;
```

به همین صورت، چنین عبارتی نیز می‌تواند به کار آید:

EXISTS $t \in R$ (p)

که خلاصه شده این عبارت است:

EXISTS t (($t \in R$) AND (p))

در حقیقت ما می‌توانیم مسائل را به این‌صورت ساده کنیم که برای متغیرهای مقید، محدوده قایل شویم. مثلا:

s RANGES OVER { S }

حال می‌توان قید C1 را افزود*:

```
CONSTRAINT C1
  FORALL s ( s.STATUS ≥ 0 AND s.STATUS ≤ 100 ) ;
```

در ادامه این بخش من محدوده‌های زیر را در نظر خواهم گرفت:

```
s RANGES OVER { S }
x RANGES OVER { S }
y RANGES OVER { S }
p RANGES OVER { P }
sp RANGES OVER { SP }
```

در زمان لازم متغیرهای مقید دیگری را هم معرفی خواهم کرد.

---

* SQL قیدهای ساده‌ای مانند این را مجاز می‌شمارد. میتوان جمله "CHECK constraint" به این شکل را در تعریف جدول پایه S به کار برد:

CONSTRAINT C1 CHECK (STATUS >= 0 AND STATUS <= 100) همچنین اگر بخواهیم می‌توانیم "CONSTRAINT C1" را حذف کنیم.

مثال 2

توزیع‌کنندگان لندنی بایستی رتبه 20 داشته باشند.

```
CONSTRAINT C2
  FORALL s ( IF s.CITY = 'London' THEN s.STATUS = 20 ) ;
```

مثال 3

شماره توزیع‌کننده یکتاست.

```
CONSTRAINT C3
  FORALL x, y ( IF x.SNO = y.SNO THEN x.SNAME  =
y.SNAME
                AND  x.STATUS = y.STATUS
                AND  x.CITY   = y.CITY ) ;
```

سور FORALL *x,y* خلاصه شده FORALL *x* FORALL *y* است.

مثال 4

توزیع‌کنندگان با رتبه کمتر از 20 نمی‌توانند قطعه P6 را توزیع کنند.

```
CONSTRAINT C4
  FORALL s ( IF s.STATUS < 20 THEN
    FORALL sp ( IF sp.SNO = s.SNO THEN
          sp.PNO ≠ PNO('P6') ) ) ;
```

مثال 5

هر سفارش باید یک توزیع‌کننده (معتبر) داشته باشد.

```
CONSTRAINT C5
  FORALL sp ( EXISTS s ( sp.SNO = s.SNO ) ) ;
```

مثال 6

هیچ شماره توزیع‌کننده‌ای بطور مشترک در دو مرابطه LS و NLS وجود ندارد.

```
CONSTRAINT C6
  FORALL m ∈ LS FORALL n ∈ NLS ( m.SNO ≠ n.SNO ) ;
```

مثال 7

توزیع‌کننده S1 و قطعه P1 هرگز نبایستی در دو شهر متفاوت باشند.

```
CONSTRAINT C7
  NOT ( EXISTS s EXISTS p ( s.SNO = SNO('S1') AND
                            p.PNO = PNO('P1') AND
                            s.CITY ≠ p.CITY ) ) ;
```

جهت آموزش چند نکته جدید، مثال‌های دیگری (نه بر پایه فصل 6) می‌آورم:

مثال 8

همیشه حداقل یک توزیع‌کننده باید وجود دارد.

```
CONSTRAINT C8
  EXISTS s ( TRUE ) ;
```

مثال 9

تمام قطعات آبی و یا سبز هستند و حداقل یک قطعه وجود دارد.

```
CONSTRAINT C9
    FORALL p ( p.COLOR = 'Blue' OR p.COLOR = 'Green' )
AND
    EXISTS p ( TRUE ) ;
```

توجه داشته باشید که در مثال بعد دو متغیر مقید مستقل وجود دارند که هر دو *p* نامیده شده‌اند.

مثال 10

تمام قطعات آبی و یا سبز هستند و یا حداقل یک قطعه قرمز وجود دارد.

```
CONSTRAINT C10
  FORALL p ( p.COLOR = 'Blue' OR p.COLOR = 'Green' ) OR
  EXISTS p ( p.COLOR = 'Red' ) ;
```

کوئری‌ها

در این قسمت نشان می‌دهم که چگونه می‌توان از ریاضیات در نوشتن کوئری‌ها استفاده کرد. در اینجا نیز از مثال‌های فصل 5 استفاده شده است و دوباره تذکر می‌دهم که عبارات نوشته شده تنها حالت‌های ممکن نیستند.

مثال 1

توزیع‌کنندگان پاریسی را بده.

s WHERE *s*.CITY = 'Paris'

مثال 2

توزیع‌کنندگانی که دست‌کم یک قطعه توزیع می‌کنند را بده.

s WHERE EXISTS *sp* (*sp*.SNO = *s*.SNO)

مثال 3

تمام ترکیبات ممکن نام، رتبه و شهر را بده.

s.SNAME, *s*.STATUS, *s*.CITY

می‌توانستیم بدون اینکه چیزی عوض شود WHERE TRUE را به این عبارت اضافه کنیم.

مثال 4

حاصل پیوند توزیع‌کنندگان و قطعات را بده.

s, *sp*.PNO, *sp*.QTY WHERE *s*.SNO = *sp*.SNO

مثال 5

حاصل اجتماع شهرهای توزیع‌کنندگان و شهرهای قطعات را بده.

u RANGES OVER { *s*.CITY, *p*.CITY }
u

متغیر مقید *u* در محدوده اجتماع شهرهای توزیع‌کنندگان و شهرهای قطعات تعریف شده است.

مثال 6

توزیع‌کنندگانی را بده که هیچ قطعه‌ای توزیع نمی‌کنند.

```
s WHERE NOT ( EXISTS sp ( sp.SNO = s.SNO ) )
```

مثال 7

قطعات را با وزن بر حسب گرم بده.

```
p, ( p.WEIGHT * 454 ) AS GMWT
```

مثال 8

شماره توزیع‌کننده و تعداد قطعات سفارش داده شده مربوط به هر توزیع‌کننده را بده.

```
s.SNO, COUNT ( sp WHERE sp.SNO = s.SNO ) AS P_COUNT
```

چند معادله

این ضمیمه را با نکاتی در مورد چند معادله مهم به پایان می‌برم هر چند ممکن است قبلا با آنها برخورد کرده باشید. اول عملگر IS_EMPTY را یادآوری می‌کنم که در فصل 5 معرفی شد و در فصل 6 بارها به کار گرفته شد. اگر سیستمی از این عملگر پشتیبانی کند دیگر نیازی به سورها نخواهد داشت به دلیل معادله‌های زیر:

```
EXISTS x ( p)  ≡  NOT ( IS_EMPTY ( x WHERE p ) )
```

و همچنین:

```
FORALL x ( p )  ≡  IS_EMPTY ( x WHERE NOT ( p) )
```

در حقیقت SQL به همین صورت و بر پایه این معادلات از سورها پشتیبانی می‌کند. در این زبان سور وجوددارد –نه دقیقا این سور چون متغیر مقیدی در آنها وجود ندارد– و بدین گونه تعریف می‌شود: فراخوانی EXISTS(*tx*) که *tx* یک عبارت جدولی است، صحیح بازمی‌گرداند اگر و تنها اگر جدول *t* با توجه به عبارت *tx* خالی نباشد.* در مقابل SQL از هرچه‌باشد پشتیبانی

* در اینجا یک تناقض مهم وجود دارد. همانطور که در فصل سوم گفته شد از آنجا که SQL از تهی پشتیبانی می‌کند بر پایه منطق سه سطحی است (بجای منطق دو سطحی که مدل رابطه‌ای بر پایه آن است). در این منطق سور وجودی سه مقدار مختلف را بر می‌گرداند: درست، غلط، و مقداری که «نامعلوم» نامیده می‌شود. ولی EXIST در SQL همواره صحیح یا غلط برمی‌گرداند و هرگز به نامعلوم برخورد نمی‌کند. در نتیجه اولا EXIST در SQL پیاده‌سازی

نمی‌کند چرا که اولا ما به هر دو سور وجوددارد و هرچه‌باشد نیاز نداریم ثانیا دلیل مهمتر اینکه ساختار FORALL(*tx*) زمانی که *tx* یک عبارت جدولی باشد، بی معناست. مثلا عبارت زیر چه معنایی ممکن است داشته باشد؟

FORALL (SELECT * FROM S)

به همین صورت می‌توان گفت اگر سیستم از عملگر COUNT پشتیبانی کند نیازی به سورها نخواهیم داشت. دلیل این ادعا معادله‌های زیر هستند.

EXISTS *x* (*p*) ≡ COUNT (*x* WHERE *p*) > 0

و همچنین:

FORALL *x* (*p*) ≡ COUNT (*x* WHERE *p*) = COUNT (*x*)

من قطعا طرفدار این عقیده نیستم که عبارت‌های دارای COUNT جایگزین سورها شوند–البته اگر از یک چهارچوب جبری ضعیف استفاده کنیم در برخی موارد مجبور به این کار خواهیم شد– ولی بی‌انصافی بود اگر این راه را به شما نشان نمی‌دادم.

خلاصه

در مقدمه این ضمیمه گفتم به نظر من، حرفه‌ای‌های بانک اطلاعاتی بایستی حداقل با مبانی حساب رابطه‌ای (یا منطق گزاره‌ها و مسائل وابسته بدان) آشنایی داشته باشند و امیدوارم اکنون با من موافق شده باشید.

من تصور می‌کنم دانستن منطق باعث می‌شود که بهتر فکر کنیم (و قطعا در رشته کاری ما بهتر فکر کردن اهمیت ویژه‌ای دارد). به خصوص که شما را به درک مناسبی از منطق سوری می‌رساند. زبان محاوره‌ای معمولا غیر دقیق است و دقت کردن در منطق سوری موجب می‌شود که معنای واقعی جملات را بفهمید. برای مثال ممکن است بخواهید *دقیقا* منظور آبراهام لینکلن را از جمله معروف وی بفهمید، و یا ممکن است فقط حدس بزنید منظور چه بوده است: « همه اوقات می‌توان سر بعضی آدمها کلاه گذاشت. همچنین بعضی اوقات می‌توان سر همه آدم‌ها کلاه گذاشت. ولی نمی‌توان همه اوقات سر همه مردم را کلاه گذاشت.» البته می‌دانم که خیلی‌ها با من موافق نیستند و عقیده دارند مردم عادی با توانایی محدود نبایستی درگیر موضوع پیچیده‌ای چون

قابل اعتمادی از سور وجودی در منطق سه سطحی نیست. ثانیا در اینجا هم کوئری‌های SQL گاهی اوقات جواب اشتباه برمی‌گردانند.

منطق شوند. آنها می‌گویند منطق مشکل‌تر از آن است که همه بتوانند با آن سر و کار داشته باشند. ممکن است این نظر بطور کلی درست باشد (منطق موضوع وسیعی است)، ولی لازم نیست تمام آن یک‌باره فراگرفته شود. من تصور نمی‌کنم شما به بیش از آنچه در این ضمیمه گفته شد نیاز داشته باشید در حالی که همین مقدار برایتان بسیار مفید خواهد بود. من چند نکته اساسی در این مورد و بر پایه مطالب این ضمیمه را در مقاله «منطق قوانین تجاری» ژوئن 2004 گردآورده‌ام. (www.dbdebunk.com) و نتیجه‌گیری پایانی را از آن نقل قول می‌کنم:

مطمئنا کمی تلاش برای آشنا شدن با [مطالب این ضمیمه] سرمایه‌ای ارزشمند خواهد بود و با استفاده از آن می‌توان از مشکلات مربوط به [قراردادها و قوانین دوپهلو و مبهم تجاری]دوری کرد. ابهام در این قوانین موجب تاخیر (در بهترین حالت) و یا مشکلات اجرایی (در بدترین حالت) می‌شوند و این تاخیرها و اشکالات مسلما موجب خسارت می‌شوند. خسارت‌های مکرری که هزینه آنها به مراتب بیشتر و تحمل آنها سخت‌تر از یک بار یادگیری این اصول است. با بیانی دیگر سازماندهی مناسب قوانین و قواعد تجاری امری بسیار جدی است و این کار نیازمند شایستگی و توانایی کافی است.

همانطور که ملاحظه می‌کنید این توصیه‌ها برای قوانین تجاری تنظیم شده‌اند، ولی من فکر می‌کنم که کاربردهای گسترده‌تری داشته باشند.

عملگرهای جبری و به‌روزرسانی در این کتاب به زبان توتوریال‌دی

RENAME:	*<relation>* RENAME (*<attribute>* AS *<name>*)
Restrict:	*<relation>* WHERE *<boolean>*
Project:	*<relation>* { <attributes> }
JOIN:	*<relation>* JOIN *<relation>* | JOIN { *<relations>* }
INTERSECT:	*<relation>* INTERSECT *<relation>* | INTERSECT { *<relations>* }
UNION:	*<relation>* UNION *<relation>* | UNION { *<relations>* }
Difference:	*<relation>* MINUS *<relation>*
Product:	*<relation>* TIMES *<relation>*
Divide:	*<relation>* DIVIDEBY *<relation>*
Semijoin:	*<relation>* MATCHING *<relation>*
Semidifference:	*<relation>* NOT MATCHING *<relation>*
EXTEND:	EXTEND *<relation>* ADD (*<exp>* AS *<name>*)
SUMMARIZE:	SUMMARIZE *<relation>* PER (*<relation>*) ADD (*<summary>* AS *<name>*)
GROUP:	*<relation>* GROUP ({ *<attributes>* } AS *<name>*)
UNGROUP:	*<relation>* UNGROUP (*<attributes>*)
Relation Assign:	*<relvar>* := *<relation>*
INSERT:	INSERT *<relvar>* *<relation>*
DELETE:	DELETE *<relvar>* [WHERE *<boolean>*]
UPDATE:	UPDATE *<relvar>* [WHERE *<boolean>*] (*<attribute assigns>*)
Attribute Assign:	*<attribute>* := *<exp>*

پایگاه داده توزیع‌کنندگان و قطعات با مقدارهای نمونه

S

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

PNO	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12.0	London
P2	Bolt	Green	17.0	Paris
P3	Screw	Blue	17.0	Oslo
P4	Screw	Red	14.0	London
P5	Cam	Blue	12.0	Paris
P6	Cog	Red	19.0	London

SP

SNO	PNO	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

درباره نویسنده کتاب

سی‌جی‌دیت نویسنده، مدرس و محقق مستقل و پژوهشگر در زمینه مدل رابطه‌ای پایگاه‌داده است. وی از اولین کسانی بود که اهمیت بنیادی نخستین ارائه مدل رابطه‌ای توسط کاد، را دریافت. وی همچنین عضو گروه طراحی IBM برای محصولات SQL/DS و DB2 در آزمایشگاه سانتاترزا واقع در سن‌جونز کالیفرنیا بود. او معمولا بخاطر کتابهایش خصوصا کتاب *مقدمه‌ای بر پایگاه‌داده‌ها* شناخته می‌شود که تاکنون بیش از هشتصدهزار نسخه از آن در سراسر جهان به فروش رفته است. شهرت وی همچنین بخاطر توانایی تشریح مسائل پیچیده فنی، به زبان ساده و قابل فهم است.

برای مطالعه بیشتر

This appendix gives a short list of suggestions for further reading. I apologize for the fact that the majority of the references are to publications for which I'm either the author or a coauthor....They're in chronological order by date of first publication.

Robert R. Stoll: Sets, Logic, and Axiomatic Theories. **San Francisco, Calif.: W. H. Freeman and Company (1961)**.The relational model is, of course, solidly founded on logic and set theory. This book provides a fairly formal but not too difficult introduction to these topics. See also the book by Hodges, later.

E. F. Codd: "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks," IBM Research Report RJ599 (August 19th, 1969); "A Relational Model of Data for Large Shared Data Banks," CACM 13, No. 6 (June 1970).The 1969 paper was Codd's very first paper on the relational model; essentially it's a preliminary version of the 1970 paper, with a few interesting differences. That 1970 paper—which was republished in "Milestones of Research," *CACM 26*, No. 1 (January 1982) and elsewhere—was the first widely available paper on the subject. It's usually credited with being the seminal paper in the field, though that characterization is a little unfair to its 1969 predecessor. In my strong opinion, every database professional should read one or both of these papers every year. *Exercise:* This latter sentence (not counting that initial "In my strong opinion," of course) is a business rule! Try casting it into the formalism of Appendix A.

C. J. Date: *An Introduction to Database Systems*, Eighth Edition. Boston, Mass.: Addison-Wesley (2004). The first edition of this book was published in 1975 A college-level text on all aspects of database management. SQL discussions are at the SQL:1999 level, with a few comments on SQL:2003.

Patrick Hall, Peter Hitchcock, and Stephen Todd: "An Algebra of Relations for Machine Computation," Conf. Record of the 2nd ACM Symposium on Principles of Programming Languages, Palo Alto, Calif. (January 1975). This paper is perhaps a little "difficult," but I think it's important. In particular, the version of the relational algebra I've described in this book has its roots in this paper.

Wilfrid Hodges: *Logic*. London, England: Penguin Books (1977). This book is a very gentle introduction to logic for the uninitiated.

C. J. Date and Hugh Darwen: *A Guide to the SQL Standard*, Fourth Edition. Reading, Mass.: Addison-Wesley (1997). The first edition of this book was published in 1987. A complete tutorial reference and critical guide to the SQL standard as of 1997. Although that version of the standard has since been superseded, first by SQL:1999 and then by the current standard SQL:2003, just about everything in the book is still accurate and applicable today.

Jim Gray and Andreas Reuter: *Transaction Processing: Concepts and Techniques*. San Mateo, Calif.: Morgan Kaufmann (1993). The standard text on transaction management.

E. F. Codd and C. J. Date: "Much Ado about Nothing," in C. J. Date, *Relational Database Writings 1991-1994*. Reading, Mass.: Addison-Wesley (1995). The article was also published on the web site www.dbdebunk.com (February 2005). Codd was perhaps the foremost advocate of nulls and 3VL as a basis for dealing with missing information (a curious state of affairs, you might think, given that nulls violate Codd's own *Information Principle*). This article contains the text of a debate between Codd and myself on the subject. It includes the following delightful remark: "Database management would be easier if missing values didn't exist" (Codd). *Note:* I've included this particular reference—out of a huge number of available publications on the topic—because it at least touches on most of the arguments on both sides of the issue.

Hugh Darwen: "What a Database *Really* Is: Predicates and Propositions," in C. J. Date, *Relational Database Writings 1994-1997*. Reading, Mass.: Addison-Wesley (1998). A very approachable tutorial on relvar predicates and related matters.

C. J. Date and Hugh Darwen: *Databases, Types, and the Relational Model: The Third Manifesto*, Third Edition. Reading, Mass.: Addison-Wesley (2006). The first edition of this book appeared (under a different title) in 1998. An advanced (graduate-level?) text on database technology. It includes a comprehensive definition of the language **Tutorial D**,which I use as a basis for examples throughout the body of the present book. Don't be fooled by that 2006 copyright date, by the way; the book is actually due to be published in 2005.

Fabian Pascal: ***Practical Issues in Database Management: A Reference for the Thinking Practitioner*****. Boston, Mass.: Addison-Wesley (2000)**. This book focuses on a number of common database issues—normalization, redundancy, integrity, missing information, and others—and discusses theoretically correct approaches to those issues, with the emphasis on the practical application of that theory.

C. J. Date: ***The Database Relational Model: A Retrospective Review and Analysis*****. Reading, Mass.: Addison-Wesley (2001)**. An unbiased retrospective review and analysis of Codd's relational contribution as documented in the papers he published in the period 1969-1979.

C. J. Date: "Double Trouble, Double Trouble" (in two parts), www.dbdebunk.com (April 2002). A detailed treatment of the problems caused by duplicates. The discussion of duplicates in chapter 3 is based on an example from this paper.

Jim Melton and Alan R. Simon: ***SQL:1999—Understanding Relational Components;*** **Jim Melton:** ***Advanced SQL:1999—Understanding Object-Relational and Other Advanced Features*****. San Francisco, Calif.: Morgan Kaufmann (2002 and 2003, respectively)**. So far as I know, these two books are the only comprehensive tutorials available on all aspects of SQL:1999 (the immediate predecessor of the current standard SQL:2003). Melton was the editor of the SQL standard for many years.

C. J. Date, Hugh Darwen, and Nikos A. Lorentzos: ***Temporal Data and the Relational Model*****. San Francisco, Calif.: Morgan Kaufmann (2003)**. Some indication of what this book covers can be found in chapter 8 of the present book.

Hugh Darwen: "How to Handle Missing Information Without Using Nulls" (presentation slides), www.thethirdmanifesto.com (May 9th, 2003). In Chapter 7 of the present book, I briefly touched on a technique for representing missing information without using nulls. This presentation elaborates on another such technique, similar but not identical to the one mentioned in Chapter 7.

C. J. Date: "What First Normal Form Really Means," www.dbdebunk.com (June 2003). First normal form has been the subject of much misunderstanding over the years. This paper is an attempt to set the record straight—even to be definitive, as far as possible. The crux of the argument, as indicated in chapter 2 of the present book, is that the concept of *atomicity* (in terms of which first normal form was originally defined) has no absolute meaning.

C. J. Date: "A Sweet Disorder," www.dbdebunk.com (August 2003). Relations don't have a left-to-right ordering to their attributes, but SQL tables do have such an ordering to their columns. This paper explores some of the consequences of this state of affairs, which turn out to be rather less trivial than you might think.

C. J. Date: "On the Notion of Logical Difference," "On the Logical Difference Between Model and Implementation," and "On the Logical Differences Between Types, Values, and Variables" (in two parts), **www.dbdebunk.com (July 2004)**. The titles say it all.

C. J. Date: "Data Redundancy and Database Design" (in three parts),www.dbdebunk.com (March/April/May 2005). A discussion of the theory (such as it is) of database design, with the emphasis on reducing data redundancy. In particular, the paper includes a detailed treatment of *The Principle of Orthogonal Design*.

C. J. Date: *Go Faster! The TransRelational™ Approach to DBMS Implementation*. To appear. A detailed tutorial on *The TransRelational? Model* (see Chapter 7 of the present book).

لغت‌نامه فارسی

n تایی	n-adic
آبراهام لینکولن	Lincoln, Abraham
آلن سیمون	Simon, Alan R.
آندریاس رویتر	Reuter, Andreas
آنومالی به‌روزرسانی	update anomalies
اتمی بودن	atomicity
اجبار	coercion
اجتماع	union
اجتماع مجزا	disjoint union
ادوارد ابی	Abbey, Edward
ارسطو	Aristotle
اریتی (درجه)	arity
از گروه درآوردن	ungroup
استفان تاد	Todd, Stephen
استفان فارلوت	Faroult, Stephane
استقلال داده‌ای	data independence
اسکالر	scalar
اشاره‌گر	pointers

اشتراک	intersect
اصل اطلاعات	Information Principle, The
اصل انتساب	Assignment Principle, The
اصل تعویض‌پذیری	Principle of Interchangeability, The
اصل طراحی متعامد (تفکیکی)	Principle of Orthogonal Design, The
اصل موضوعه	axiom (database)
اصل نرمال‌سازی	Principles of Normalization
اصل تفاوت غیرقابل تشخیص	Principle of Identity of Indiscernibles, The
اعلانی	declarative
افزونگی	redundancy
انتخابگر	selector
انتخابگر تاپل	tuple selector
انتخابگر رابطه	relation selector
انتساب	assignment
انتساب چندتایی	multiple assignment
انتساب رابطه‌ای	relational assignment
بازنویسی کوئری	query rewrite
بدنه	body
برتراند راسل	Russell, Bertrand
بسته بودن	closure
بسط خارجی	extension
بسط داخلی	intension

update	به‌روزرسانی
optimizer	بهینه‌ساز
semantic optimization	بهینه‌سازی معنایی
boolian	بولی (منطقی, صحیح-غلط)
Third Manifesto, The	بیانیه سوم
Hall, Patrick	پاتریک هال
multi-dimensional database	پایگاه چند بعدی
database	پایگاه داده‌ها
project	پرتو
identity projection	پرتو اصلی
triggered procedure	پروسیجرهای تریگردار
Hitchcock, Peter	پیتر هیچکاک
natural join	پیوند طبیعی
implementation	پیاده‌سازی
join	پیوند
theta-join	پیوند تتا
equijoin	پیوند مساوی
function	تابع
aggregate operator	تابع جمعی
tuple	تاپل
empty tuple	تاپل خالی
0-tuple	تاپل صفر

expression transformation	تبدیل عبارت
lossy decomposition	تجزیه کم‌وکاست‌دار
nonloss decomposition	تجزیه بی‌کم‌وکاست
orthogonal decomposition	تجزیه متعامد
TransRelational™ Model, The	ترارابطه‌ای
transaction	تراکنش
transitivity	تراگذری
order	ترتیب
equality	تساوی
relation equality	تساوی رابطه‌ها
cascade	تسلسلی
snapshot	تصویر لحظه‌ای
direct image	تصویر مستقیم
rename	تغییر نام
difference (minus)	تفاضل
intended interpretation	تفسیر مورد نظر
orthogonality	تعامد (تفکیک)
divide	تقسیم
isolation	تنهایی
null	تهی
Tutorial D	توتوریال‌دی
distributivity	توزیع پذیری

duplicate	تکرار
literal	ثابت
relation constant	ثابت رابطه‌ای
consistency	ثبات
relcon	ثرابطه (ثابت رابطه‌ای)
commutativity	جابجایی پذیری
integrity	جامعیت
referential integrity	جامعیت ارجاعی
entity integrity	جامعیت وجودی
Gennick, Jonathan	جاناتان جنیک
surrogate	جایگزین
relational algebra	جبر رابطه‌ای
distinct	جدا
table	جدول
Bush, George W.	جرج دبلیو بوش
Celko, Joe	جو سلکو
Gray, Jim	جیم گری
Melton, Jim	جیم ملتون
delete	حذف
relational calculus	حساب رابطه‌ای
dependency preservation	حفظ وابستگی
summarize	خلاصه کردن

idempotence	خودتوانی
bag	خورجین (مجموعه با تکراراعضا)
connection trap	دام ارتباطی
domain	دامنه
insert	درج
degree	درجه
Knuth, Donald E.	دونالد نوث
retrieval view	دید بازیابی‌کننده
updating view	دید به‌روزرسانی‌کننده
materialized view	دید جامه‌عمل‌پوشیده
Stoll, Robert R.	رابرت استول
connective	رابط
relation	رابطه
derived relation	رابطه بدست آمده
binary relation	رابطه باینری (دو ستونی)
base relation	رابطه پایه
ternary relation	رابطه ترنری (سه ستونی)
empty relation	رابطه خالی
n-ary relation	رابطه درجهn
flat relation	رابطه مسطح
unary relation	رابطه یونری (یک ستونی)
relational completeness	رابطه‌ای تمام‌عیار بودن

Fagin, Ron	رن فاگین
procedural	روندگرا
temporal	زمانمند
spatiotemporal	زمانی-مکانی
subset	زیرمجموعه
procedure	زیرروال
proper subset	زیرمجموعه محض
column	ستون
row	سطر
quantifier	سور
universal quantifier	سور عمومی
existential quantifier	سور وجودی
Date, C. J.	سی‌جی دیت
database management system (DBMS)	سیستم مدیریت پایگاه‌داده‌ها
logical system	سیستم منطقی
associativity	شرکت پذیری
ID	شناسه
object	شیء
product (times)	ضرب
cartesian product	ضرب دکارتی
physical design	طراحی فیزیکی
domain check override	عدم تطبیق با دامنه

operator	عملگر
primitive operators	عملگرهای اولیه
set-level operations	عملگرهای مجموعه‌ای
referential action	عملیات ارجاعی
heading	عنوان
empty heading	عنوان خالی
nonscalar	غیراسکالر
superkey	فراکلید
Closed World Assumption, The	فرض بسته‌بودن جهان
first normal form (1NF)	فرم اول نرمال
fifth normal form (5NF)	فرم پنجم نرمال
fourth normal form (4NF)	فرم چهارم نرمال
second normal form (2NF)	فرم دوم نرمال
third normal form (3NF)	فرم سوم نرمال
sixth normal form (6NF)	فرم ششم نرمال
normal form	فرم نرمال
Boyce/Codd normal form (BCNF)	فرم نرمال بویس کاد
projection join normal form (PJ/NF)	فرم نرمال پرتو-پیوند
Pascal, Fabian	فوبین پاسکال
Golden Rule, The	قاعده طلایی
Fagin's theorem	قضیه فاگین
Heath's theorem	قضیه هیث

rules of inference	قواعد استنتاج
business rules	قواعد تجاری
strong typing	قویا نوع‌دار
constraint	قید
database constraint	قید پایگاه
single-relvar constraint	قید تک‌مرابطه‌ای
integrity constraint	قید جامعیت
multi-relvar constraint	قید چندمرابطه‌ای
transition constraint	قید گذار
relvar constraint	قید مرابطه
type constraint	قید نوع
cardinality	کاردینالیتی
primary key	کلید اصلی
candidate key	کلید کاندید
CODASYL	کوداسیل
group	گروه‌بندی
Chudnovsky, Gregory	گریگوری چاندنوسکی
proposition	گزاره
n-place predicate	گزاره n پارامتری
predicate	گزاره‌نما
relvar predicate	گزاره‌نمای مرابطه
restrict	گزینش

extend	گسترش‌دادن
da Vinci, Leonardo	لئوناردو داوینچی
Wittgenstein, Ludwig	لودویک ویتجنستین
Carroll, Lewis	لویس کارول
durability	ماندگاری
Stonebraker, Mike	مایک استونبریکر
bound variable	متغیر مقید
variable	متغیر
free variable	متغیر آزاد
relation variable	متغیر رابطه‌ای
superset	مجموعه مادر
proper superset	مجموعه مادر محض
empty range	محدوده خالی
interval	مدت زمان
model	مدل
data model	مدل داده
relational model	مدل رابطه‌ای
hierarchic model	مدل سلسله‌مراتبی
network model	مدل شبکه‌ای
object-oriented model	مدل شی‌گرا
object-relational model	مدل شی‌گرا–رابطه‌ای
semistructured model	مدل نیمه‌ساخت‌یافته

E/R modeling	مدل‌سازی موجودیت–رابطه
database administrator (DBA)	مدیر پایگاه داده‌ها
relvar	مرابطه (متغیر رابطه‌ای)
derived relval	مرابطه بدست‌آمده
base relvar	مرابطه پایه
self-referencing relvar	مرابطه خودارجاع
virtual relvar	مرابطه مجازی
referenced relvar	مرابطه مورد مراجعه
semantic	معنایی
relational comparisons	مقایسه رابطه‌ای
value	مقدار
relation value	مقدار رابطه‌ای
two-valued logic (2VL)	منطق دوحالته
three-valued logic (3VL)	منطق سه‌حالته
normalization	نرمال سازی
DCO	نقض تطبیق دامنه
possrep	نماممکن
possible representation	نمایش امکان‌پذیر
type	نوع
system-defined type	نوع تعریف شده در سیستم
user-defined type	نوع تعریف‌شده بوسیله کاربر
built-in type	نوع توکار

empty type	نوع خالی
data type	نوع داده
relation type	نوع رابطه
generated type	نوع ساختگی
type generator	نوع‌ساز
tuple type generator	نوع‌ساز تاپل
Lorentzos, Nikos A.	نیکوس لورنتزوس
smiminus	نیم تفاضل
semijoin	نیم‌پیوند
semidifference	نیم‌تفاضل
forall	هرچه‌باشد
Darwen, Hugh	هیو داروین
dependency	وابستگی
nontrival dependency	وابستگی با اهمیت
trivial dependency	وابستگی بی‌اهمیت
join dependency (JD)	وابستگی پیوندی
functional dependency (FD)	وابستگی تابعی
multi-valued dependency (MVD)	وابستگی چندمقداری
exists	وجود دارد
Hodges, Wilfrid	ویلفرد هادگس
Shakespeare, William	ویلیام شکسپیر
attribute	ویژگی

relation valued attribute (RVA)	ویژگی با مقدار رابطه
ACID properties	ویژگی‌های اثتم
view	ویو
Heath, Ian	یان هیث
performance	کارایی
denormalization	کاهش درجه نرمال
irreducibility	کاهش ناپذیری
key	کلید
foreign key	کلید خارجی
empty key	کلید خالی
query	کوئری
quota query	کوئری سهمیه‌ای
uniqueness (key)	یکتایی (کلید)

لغت‌نامه انگلیسی

0-tuple	تاپل صفر
1NF	فرم اول نرمال
2NF	فرم دوم نرمال
2VL	منطق دوحالته
3NF	فرم سوم نرمال
3VL	منطق سه‌حالته
4NF	فرم چهارم نرمال
5NF	فرم پنجم نرمال
6NF	فرم ششم نرمال
Abbey, Edward	ادوارد ابی
ACID properties	ویژگی‌های اتم
aggregate operator	تابع جمعی
Aristotle	ارسطو
Arity	اریتی (درجه)
assignment	انتساب
Assignment Principle, The	اصل انتساب
associativity	شرکت پذیری
atomicity	اتمی بودن
attribute	ویژگی
axiom (database)	اصل موضوعه

Bag	خورجین (مجموعه با تکرار اعضا)
base relation	رابطه پایه
base relvar	مرابطه پایه
BCNF	فرم نرمال بویس کاد
binary relation	رابطه باینری (دو ستونی)
Body	بدنه
boolian	بولی (منطقی, صحیح-غلط)
bound variable	متغیر مقید
Boyce/Codd normal form	فرم نرمال بویس کاد
built-in type	نوع توکار
Bush, George W.	جرج دبلیو بوش
business rules	قواعد تجاری
candidate key	کلید کاندید
cardinality	کاردینالیتی
Carroll, Lewis	لویس کارول
cartesian product	ضرب دکارتی
cascade	تسلسلی
Celko, Joe	جو سلکو
Chudnovsky, Gregory	گریگوری چاندنوسکی
Closed World Assumption, The	فرض بسته بودن جهان
closure	بسته بودن
CODASYL	کوداسیل

coercion	اجبار
column	ستون
commutativity	جابجایی پذیری
connection trap	دام ارتباطی
connective	رابط
consistency	ثبات
constraint	قید
da Vinci, Leonardo	لئوناردو داوینچی
Darwen, Hugh	هیو داروین
data independence	استقلال داده‌ای
data model	مدل داده
data type	نوع داده
database	پایگاه داده‌ها
database administrator	مدیر پایگاه داده‌ها
database constraint	قید پایگاه
database management system	سیستم مدیریت پایگاه‌داده‌ها
Date, C. J.	سی‌جی دیت
DBA	مدیر پایگاه داده‌ها
DBMS	سیستم مدیریت پایگاه‌داده‌ها
DCO	نقض تطبیق دامنه
declarative	اعلانی
DEE	-

degree	درجه
delete	حذف
denormalization	کاهش درجه نرمال
dependency	وابستگی
dependency preservation	حفظ وابستگی
derived relation	رابطه بدست‌آمده شده
derived relval	مرابطه بدست‌آمده
difference	تفاضل
direct image	تصویر مستقیم
disjoint union	اجتماع مجزا
distinct	جدا
distributivity	توزیع پذیری
divide	تقسیم
domain	دامنه
domain check override	عدم تطبیق با دامنه
DUM	-
duplicate	تکرار
durability	ماندگاری
E/R modeling	مدل‌سازی موجودیت–رابطه
empty heading	عنوان خالی
empty key	کلید خالی
empty range	محدوده خالی

empty relation	رابطه خالی
empty tuple	تاپل خالی
empty type	نوع خالی
entity integrity	جامعیت وجودی
equality	تساوی
equijoin	پیوند مساوی
existential quantifier	سور وجودی
exists	وجود دارد
expression transformation	تبدیل عبارت
extend	گسترش‌دادن
extension	بسط خارجی
Fagin, Ron	رن فاگین
Fagin's theorem	قضیه فاگین
Faroult, Stephane	استفان فارلوت
FD	وابستگی تابعی
fifth normal form	فرم پنجم نرمال
first normal form	فرم اول نرمال
flat relation	رابطه مسطح
forall	هرچه‌باشد
foreign key	کلید خارجی
fourth normal form	فرم چهارم نرمال
free variable	متغیر آزاد

function	تابع
functional dependency	وابستگی تابعی
generated type	نوع ساختگی
Gennick, Jonathan	جاناتان جنیک
Golden Rule, The	قاعده طلایی
Gray, Jim	جیم گری
group	گروه‌بندی
Hall, Patrick	پاتریک هال
heading	عنوان
Heath, Ian	یان هیث
Heath's theorem	قضیه هیث
hierarchic model	مدل سلسله‌مراتبی
Hitchcock, Peter	پیتر هیچکاک
Hodges, Wilfrid	ویلفرد هادگس
ID	شناسه
Idempotence	خودتوانی
identity projection	پرتو اصلی
Implementation	پیاده‌سازی
Information Principle, The	اصل اطلاعات
Insert	درج
Integrity	جامعیت
integrity constraint	قید جامعیت

intended interpretation	تفسیر موردنظر
Intension	بسط داخلی
Intersect	اشتراک
Interval	مدت زمان
Irreducibility	کاهش ناپذیری
Isolation	تنهایی
JD	وابستگی پیوندی
Join	پیوند
join dependency	وابستگی پیوندی
Key	کلید
Knuth, Donald E.	دونالد نوث
Lincoln, Abraham	آبراهام لینکولن
Literal	ثابت
logical system	سیستم منطقی
Lorentzos, Nikos A.	نیکوس لورنتزوس
lossy decomposition	تجزیه کم‌وکاست‌دار
materialized view	دید جامه‌عمل‌پوشیده
Melton, Jim	جیم ملتون
minus	تفاضل
model	مدل
multi-dimensional database	پایگاه چندبعدی
multiple assignment	انتساب چندتایی

multi-relvar constraint	قید چندمرابطه‌ای
multi-valued dependency	وابستگی چندمقداری
MVD	وابستگی چندمقداری
n-adic	n تایی
n-ary relation	رابطه درجهn
natural join	پیوند طبیعی
network model	مدل شبکه‌ای
nonloss decomposition	تجزیه بی‌کم‌وکاست
nonscalar	غیراسکالر
nontrival dependency	وابستگی با اهمیت
normal form	فرم نرمال
normalization	نرمال سازی
n-place predicate	گزاره n پارامتری
null	تهی
object	شیء
object-oriented model	مدل شی‌گرا
object-relational model	مدل شی‌گرا–رابطه‌ای
operator	عملگر
optimizer	بهینه‌ساز
order	ترتیب
orthogonal decomposition	تجزیه متعامد
orthogonality	تعامد (تفکیک)

Pascal, Fabian	فوبین پاسکال
performance	کارایی
physical design	طراحی فیزیکی
PJ/NF(5NF)	فرم نرمال پرتو-پیوند
pointers	اشاره‌گر
possible representation	نمایش امکان‌پذیر
possrep	نماممکن
predicate	گزاره‌نما
primary key	کلید اصلی
primitive operators	عملگرهای اولیه
Principle of Identity of Indiscernibles, The	اصل تفاوت غیرقابل تشخیص
Principle of Interchangeability, The	اصل تعویض‌پذیری
Principle of Orthogonal Design, The	اصل طراحی متعامد
Principles of Normalization	اصل نرمال‌سازی
Procedural	روندگرا
Procedure	زیرروال
Product	ضرب
Project	پرتو
projection join normal form	فرم نرمال پرتو-پیوند
proper subset	زیرمجموعه محض
proper superset	مجموعه مادر محض
Proposition	گزاره

Quantifier	سور
Query	کوئری
query rewrite	بازنویسی کوئری
quota query	کوئری سهمیه‌ای
Redundancy	افزونگی
referenced relvar	مرابطه مورد مراجعه
referential action	عملیات ارجاعی
referential integrity	جامعیت ارجاعی
Relation	رابطه
relation constant	ثابت رابطه‌ای
relation equality	تساوی رابطه‌ها
relation selector	انتخابگر رابطه
relation type	نوع رابطه
relation value	مقدار رابطه‌ای
relation valued attribute	ویژگی با مقدار رابطه
relation variable	متغیر رابطه‌ای
relational algebra	جبر رابطه‌ای
relational assignment	انتساب رابطه‌ای
relational calculus	حساب رابطه‌ای
relational comparisons	مقایسه رابطه‌ای
relational completeness	رابطه‌ای تمام‌عیار بودن
relational model	مدل رابطه‌ای

Relcon	ثرابطه (ثابت رابطه‌ای)
Relvar	مرابطه (متغیر رابطه‌ای)
relvar constraint	قید مرابطه
relvar predicate	گزاره‌نمای مرابطه
Rename	تغییر نام
Restrict	گزینش
retrieval view	دید بازیابی‌کننده
Reuter, Andreas	آندریاس رویتر
Row	سطر
rules of inference	قواعد استنتاج
Russell, Bertrand	برتراند راسل
RVA	ویژگی با مقدار رابطه
Scalar	اسکالر
second normal form	فرم دوم نرمال
Selector	انتخابگر
self-referencing relvar	مرابطه خود ارجاع
Semantic	معنایی
semantic optimization	بهینه‌سازی معنایی
Semidifference	نیم‌تفاضل
Semijoin	نیم‌پیوند
semistructured model	مدل نیمه‌ساخت‌یافته
set-level operations	عملگرهای مجموعه‌ای

Shakespeare, William	ویلیام شکسپیر
Simon, Alan R.	آلن سیمون
single-relvar constraint	قید تک‌مرابطه‌ای
sixth normal form	فرم ششم نرمال
Smiminus	نیم تفاضل
Snapshot	تصویر لحظه‌ای
Spatiotemporal	زمانی–مکانی
SQL	-
Stoll, Robert R.	رابرت استول
Stonebraker, Mike	مایک استونبریکر
strong typing	قویا نوع‌دار
Subset	زیرمجموعه
Summarize	خلاصه کردن
Superkey	فراکلید
Superset	مجموعه مادر
Surrogate	جایگزین
system-defined type	نوع تعریف شده در سیستم
Table	جدول
Temporal	زمانمند
ternary relation	رابطه ترنری (سه ستونی)
theta-join	پیوند تتا
Third Manifesto, The	بیانیه سوم

third normal form	فرم سوم نرمال
three-valued logic	منطق سه‌حالته
Times	ضرب
Todd, Stephen	استفان تاد
transaction	تراکنش
transition constraint	قید گذار
transitivity	تراگذری
TransRelational™ Model, The	ترارابطه‌ای
triggered procedure	پروسیجرهای تریگردار
trivial dependency	وابستگی بی‌اهمیت
tuple	تاپل
tuple selector	انتخابگر تاپل
tuple type generator	نوع‌ساز تاپل
Tutorial D	توتوریال‌دی
two-valued logic	منطق دوحالته
type	نوع
type constraint	قید نوع
type generator	نوع‌ساز
unary relation	رابطه یونری (یک ستونی)
ungroup	از گروه درآوردن
union	اجتماع
uniqueness (key)	یکتایی (کلید)

universal quantifier	سور عمومی
update	به‌روزرسانی
update anomalies	آنومالی به‌روزرسانی
updating view	دید به‌روزرسانی‌کننده
user-defined type	نوع تعریف‌شده بوسیله کاربر
value	مقدار
variable	متغیر
view	ویو
virtual relvar	مرابطه مجازی
Wittgenstein, Ludwig	لودویک ویتجنستین
XML	-