

## مقدمه

نرمال سازی یا normalization باعث جلوگیری از تکرار و افزونگی اطلاعات می‌شود. و همچنین مانع از یکسری ناهنجاری‌ها در عملیات درج، بروز رسانی، حذف و انتخاب خواهد شد.

شکل‌های نرمال متعددی تعریف شده اند که به شرح زیر است:

شکل نرمال اول (1NF)

شکل نرمال دوم (2ND)

شکل نرمال سوم (3NF)

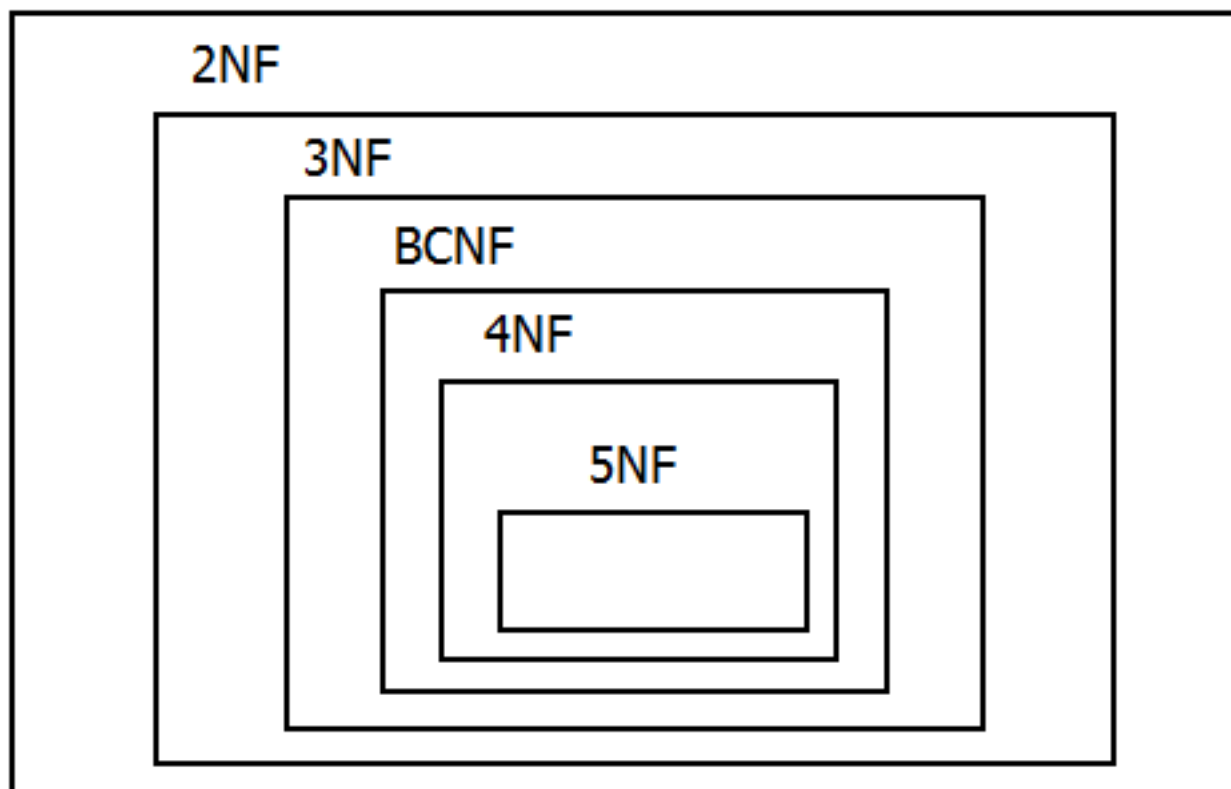
شکل نرمال بويس کاد (BCNF)

شکل نرمال چهارم (4NF)

شکل نرمال پنجم (5NF)

سه شکل اول نرمال یعنی 1NF ، 2NF و 3NF توسط دکتر Codd تعریف شده اند. شکل نرمال بويس کاد نیز که یک تعریف اصلاح شده و قوی‌تر از 3NF به Boyce و Codd منسوب است . بعد از آن Fagin شکل چهارم نرمال ( 4NF ) را تعریف کرد (چرا که در آن زمان BCNF شکل سوم نرمال خوانده می‌شد).

## 1NF



تصویر فوق می‌گوید اگر جدولی در شکل سوم نرمال باشد حتما دارای شکل دوم نرمال و شکل اول نرمال هم خواهد بود.

### شکل اول نرمال (First Normal Form)

تعریف رسمی:

یک متغیر رابطه ای به شکل اول نرمال است اگر و فقط اگر در هر مقدار مجاز آن متغیر رابطه ای، هر چندتایی فقط یک مقدار برای هر خصیصه داشته باشد.

منظور از اصطلاحات متغیر رابطه ای، چندتایی و خصیصه به طور غیر رسمی به ترتیب برابر است با جدول، سطر و ستون.

قسمت کلیدی تعریف، این جمله است: "فقط یک مقدار برای هر خصیصه داشته باشد"

به دو جدول زیر توجه کنید، این جداول به شکل اول نرمال نمی‌باشد چرا که به ازای هر مشتری برای خصیصه شماره تلفن چند مقدار خواهیم داشت:

Primary Key				
کد مشتری	نام مشتری	شماره تلفن	شماره تلفن	شماره تلفن
کد ۱	مشتری ۱	مقدار ۱	مقدار ۲	
کد ۲	مشتری ۲			
کد ۳	مشتری ۳	مقدار ۱		
کد ۴	مشتری ۴	مقدار ۱	مقدار ۲	مقدار ۳

Primary Key		
کد مشتری	نام مشتری	شماره تلفن ها
کد ۱	مشتری ۱	مقدار ۱ - مقدار ۲
کد ۲	مشتری ۲	
کد ۳	مشتری ۳	مقدار ۱
کد ۴	مشتری ۴	مقدار ۱ - مقدار ۲ - مقدار ۳

در جدول اول ستون شماره تلفن چند بار تکرار شده است. یعنی برای یک مشتری چند مقدار برای خصیصه شماره تلفن خواهیم داشت که این مغایر با تعریف شکل اول نرمال است. همین اتفاق نیز در جدول دوم افتاده است با این فرق که مقادیر خصیصه شماره تلفن در یک ستون درج شده اند.


برای تبدیل جدول غیر نرمال فوق به یک جدول نرمال اول، بایستی کاری کنیم که خصیصه شماره تلفن فقط یک مقدار را بگیرد. یعنی: در جدول فوق می بینید که برای خصیصه شماره تلفن به ازای هر سطر فقط یک مقدار داریم.

Primary Key		Primary Key	
شماره تلفن	نام مشتری	کد مشتری	شماره تلفن
مقدار ۱	مشتری ۱	کد ۱	
مقدار ۲	مشتری ۱	کد ۱	
	مشتری ۲	کد ۲	
مقدار ۱	مشتری ۳	کد ۳	
مقدار ۱	مشتری ۴	کد ۴	
مقدار ۲	مشتری ۴	کد ۴	
مقدار ۳	مشتری ۴	کد ۴	

در جدول غیر نرمال مثال پیشین چند مقدار برای یک خصیصه داشتیم. حال به مثالی می پردازیم که یک مجموعه از خصیصه ها چند

بار تکرار می‌شوند.

به جدول غیر نرمال زیر توجه کنید. دو خصیصه ترم و معدل چند بار در جدول تکرار می‌شوند. اصلاحا به این‌ها گروه‌های تکرار شونده می‌گویند.



کد دانشجو	نام دانشجو	ترم	معدل	ترم	معدل	ترم	معدل	ترم	معدل
کد ۱	نام ۱	۱	۱۴	۲	۱۵				
کد ۲	نام ۲	۱	۱۸	۲	۱۳	۳	۱۵	۴	۱۷
کد ۳	نام ۳	۱	۱۴	۲	۱۷	۳	۱۲		
کد ۴	نام ۴								

گروه‌های تکرار شونده را با آکولاد ({}) مشکل کرده ام. این گونه جداول (که حتی در شکل نرمال اول هم قرار ندارند) مشکلات فراوانی دارند که در زیر به مواردی اشاره خواهیم داشت:

چگونه معدل ترم ۵ را در جدول درج کنیم؟ پس برای اینکه بتوانیم تمام معدل‌ها را در جدول داشته باشیم باید به تعداد حداکثر ترم تحصیلی گروه‌های تکرار شونده در جدول داشته باشیم.

برای دانشجویی که فقط یک ترم تحصیل کرده است تمام گروه‌های تکرار شونده به غیر از یکی خالی خواهد ماند. فضای بسیاری به هدر خواهد رفت.

گزارش گیری بسیار سخت خواهد شد. بطور نمونه، چطور می‌خواهید بالاتری معدل دانشجویان را بدست بیاورین؟

پس با تبدیل جدول غیر نرمال به شکل نرمال اول، به مشکلات فوق غلبه خواهیم کرد:

PK		PK	
معدل	ترم	نام دانشجو	کد دانشجو
۱۴	۱	نام ۱	کد ۱
۱۵	۲	نام ۱	کد ۱
۱۸	۱	نام ۲	کد ۲
۱۳	۲	نام ۲	کد ۲
۱۵	۳	نام ۲	کد ۲
۱۷	۴	نام ۲	کد ۲
۱۴	۱	نام ۳	کد ۳
۱۷	۲	نام ۳	کد ۳
۱۲	۳	نام ۳	کد ۳
		نام ۴	کد ۴

اما یک متغیر رابطه ای که فقط به صورت شکل اول نرمال است ساختاری دارد که به دلایل متعدد، نامطلوب است.

در جدول فوق اطلاعاتی وجود دارد که به دفعات تکرار شده است. مثلاً نام دانشجو به تعداد ترم‌ها تکرار شده است. در صورتی که باید نام دانشجو یکبار ذخیره شده باشد. پس یک جدولی که به فرم نرمال اول هست می‌تواند افزونگی اطلاعات داشته باشد.

در بخش بعدی ابتدا وابستگی تابعی مورد بررسی قرار خواهد گرفت سپس به فرم دوم نرمال پرداخته خواهد شد.

## نظرات خوانندگان

نویسنده: سعید

تاریخ: ۱۳۹۱/۱۱/۱۳ ۱۶:۳۲

با تشکر از مطلب خوبتان.

این نوع مباحث رو با orm ها و کلاس های دات نتى بهتر ميشه توضيح داد. مثلا يك مشتري داريم با چندتا تلفن. يك دانشجو داريم با چندتا ترم و درس و نمره. اين چندتا رو ميشه به صورت يك icollection تعريف كرد در يك كلاس بجاي اينكه پشت سر هم خاصيت اضافه كنيم. يا حتى زمانيكه مشتري سه تا تلفن داره مشكلي نداره تماشش در همان جدول اصلى قرار بگيره. در ef به اين نوع ها، [complextype](#) گفته ميشه (يك خاصيت تو در تو در كلاس، حاليكه خاصيت كلاس خودش از نوع يك كلاس هست اما اين كلاس تبديل به يك جدول جدا نميشه) يا مثلا در nhibernate به اون [component mapping](#) هم مى گن.

نویسنده: سمیرا

تاریخ: ۱۳۹۲/۱۰/۱۰ ۲۱:۲۰

با سلام

میدونید چرا رابطه های دو صفته BCNF هستن؟

## وابستگی تابعی

برای وارد شدن به بحث نظری نرمال سازی نیاز هست با مفهوم وابستگی تابعی آشنا شویم. وابستگی تابعی یک مبحث نسبتاً مفصل و تئوری هست که زمان زیادی برای شرح جزئیات آن نیاز هست در نتیجه در حد آشنایی و نیازمان به آن توجه خواهیم داشت.

به جدول زیر نگاه کنید:

primary key		primary key	
S#	City	P#	Qty
S1	London	P1	100
S1	London	P2	100
S2	Paris	P1	200
S2	Paris	P2	200
S3	Paris	P2	300
S4	London	P2	400
S4	London	P4	400
S4	London	P5	400

این جدول نشان می‌دهد هر عرضه کننده (S#) چه قطعه (P#) را به چه تعداد (Qty) تولید کرده است. City هم شهر است که عرضه کننده در آن سکونت دارد.

از داده‌های فعلی جدول می‌شود برداشت‌های مختلفی داشت که چندتای آن به قرار زیر:

عرضه کنندگان یکسان دارای شهرهای یکسان هستند

هر عرضه کننده و قطعه تنها با یک مقدار از qty در انتظار است.

## تعریف وابستگی تابعی یا functional dependency

تعریف رسمی:

اگر  $r$  یک رابطه و  $X$  و  $Y$  زیر مجموعه‌های دلخواهی از مجموعه خصیصه‌های  $r$  باشند آنگاه می‌گوییم  $Y$  به صورت تابعی وابسته به  $X$  است و آن را به صورت زیر می‌نویسیم:

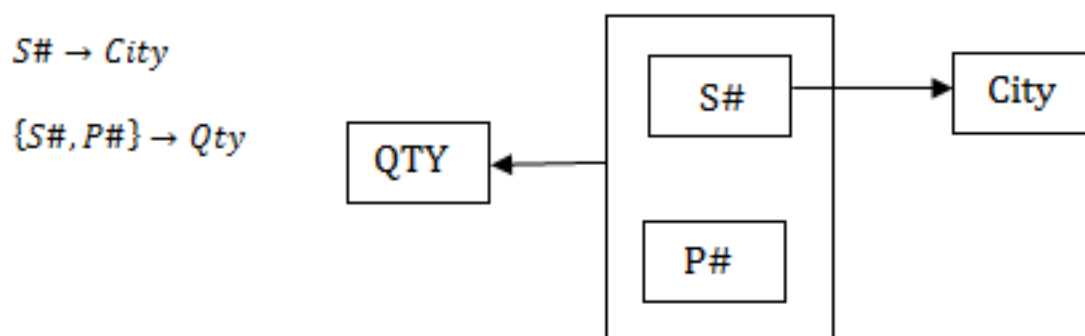
$X \rightarrow Y$

اگر و تنها اگر در هر مقدار مجاز و ممکن از  $r$ ، هر مقدار  $X$  متناظر با دقیقاً یک مقدار از  $Y$  باشد. یعنی به ازای هر  $X$  تنها یک  $Y$  داشته باشیم. به بیان دیگر هرگاه دو چندتایی از  $r$  مقدار مقدار  $X$  یکسانی داشته باشند آنگاه مقدار  $Y$  آنها یکسان باشد.

گفته شد که هر عرضه کنند تنها با یک شهر تناظر دارد. مثلاً عرضه کننده ای با مقدار S1 تنها با شهر London در تناظر است. و به ازای هر عرضه کننده قطعه تنها یک QTY خواهیم داشت مثلاً به ازای عرضه کننده با مقدار S4 و قطعه با مقدار P2 تنها یک سطر (در نتیجه یک Qty) وجود دارد (این دو خصیصه کلید هستند)

اما P# به S# وابستگی تابعی ندارد. مثلاً به ازای S4 ما چند عرضه کننده خواهیم داشت.

وابستگی تابعی را می‌توان بشکل نمودار در آورد. در زیر نمودار وابستگی همراه با وابستگی‌های تابعی جدول مورد نظر آمده است:



### تعریف شکل نرمال دوم

یک متغیر رابطه ای به شکل دوم نرمال است اگر و فقط اگر به شکل اول نرمال بوده و هر خصیصه غیر کلیدی وابسته به کلید اولیه باشد.

بر می‌گردیم به آخرین جدول مطلب گذشته یعنی:



معدل	ترم	نام دانشجو	کد دانشجو
۱۴	۱	نام ۱	کد ۱
۱۵	۲	نام ۱	کد ۱
۱۸	۱	نام ۲	کد ۲
۱۳	۲	نام ۲	کد ۲
۱۵	۳	نام ۲	کد ۲
۱۷	۴	نام ۲	کد ۲
۱۴	۱	نام ۳	کد ۳
۱۷	۲	نام ۳	کد ۳
۱۲	۳	نام ۳	کد ۳
		نام ۴	کد ۴

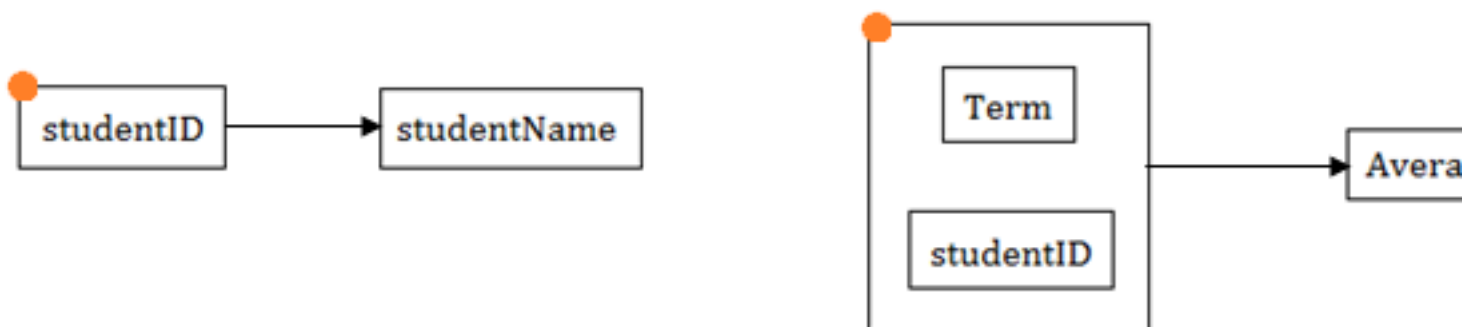
کلید اولیه این جدول از ترکیب دو ستون کد دانشجو و ترم تشکیل شده است. معدل را کلید اولیه تعیین می‌کند یعنی معدل وابسته به مقدار کلید اولیه است، اما نام دانشجو وابستگی به کلید اولیه ندارد و به جای آن وابسته به ستون کد دانشجو است. در نتیجه طبق تعریفی که داشتیم این جدول به شکل دوم نرمال نیست. این جدول دقیقا مشابه به جدول عرضه کننده - قطعات است (که در ابتدا مطلب آمده است) پس نمودار آن نیز با FD این جدول برابر است.

برای تبدیل از فرم 1 به فرم 2 نرمال باید جدول را تجزیه کنیم به دو جدول:

جدول دانشجو (کد دانشجو - نام دانشجو)

جدول معدل (کد دانشجو - ترم - معدل)

به نمودار FD جدول فوق بعد از تجزیه شدن دقت بفرمایید:



همانطور که مشاهده می‌شود فلش‌ها تنها از خصیصه‌های کلید اولیه خارج شده اند در حالی که قبل از تجزیه شدن فلش ای وجو داشت که از کلید اولیه خارج نشده بود. کلیدهای اولیه توسط نقطه نارنجی رنگ علامت گذاری شده اند.

و بالاخره فرم دوم نرمال جدول سابق:

نام دانشجو	کد دانشجو
نام ۱	کد ۱
نام ۲	کد ۲
نام ۳	کد ۳
نام ۴	کد ۴

معدل	ترم	کد دانشجو
۱۴	۱	کد ۱
۱۵	۲	کد ۱
۱۸	۱	کد ۲
۱۳	۲	کد ۲
۱۵	۳	کد ۲
۱۷	۴	کد ۲
۱۴	۱	کد ۳
۱۷	۲	کد ۳
۱۲	۳	کد ۳

کلیدهای اولیه با نقطه بنفش علامت گذاری شده است.

در اینجا با تجزیه جدول، به شکل سوم نرمال رسیدیم. در پست بعدی مثالی از یک جدول نرمال دوم خواهیم آورد و همزمان با بررسی معایب آن شکل سوم نرمال را نیز معرفی خواهیم نمود.

مرجع

کتاب پایگاه داده‌ی C.J. Date

## نظرات خوانندگان

نویسنده: senaps

تاریخ: ۱۸:۴۷ ۱۳۹۱/۱۱/۱۳

خوب من خیلی خوشحالم....

من همیشه دیتابیس رو به همین شکل طراحی میکنم! (یعنی حداقل جداولم حد نرمال دوم رو دارن!) .... حالا تا ببینم در آینده چی میشه ماجرا که ببینم بر این اساس، ایا من کلا جداولم رو نرمال طراحی میکنم یا چی؟!  
 اخه من هیچوقت نرمال سازی رو یاد نگرفتم(البته تو دانشگاه هم درس نداد این مسئله رو استاد مربوطه...) ولی خوب طراحی دیتابیس رو دوتایی با هم اینجوری کار کردیم که من معمولا مثل جدول های اخر این پست کار میکنم....

نویسنده: محمد سلم آبادی

تاریخ: ۲۰:۵۸ ۱۳۹۱/۱۱/۱۳

این دو جدول آخر به شکل سوم نرمال هستند. یعنی شرط نرمال سوم را نیز محقق کرده اند. در مطلب بعدی یک مثال از جدولی خواهم آورد که به شکل دوم نرمال بوده ولی به شکل سوم نرمال نباشد.

نویسنده: حسینی

تاریخ: ۱۷:۴ ۱۳۹۲/۰۵/۲۶

با سلام؛ شما ترکیب کد دانشجو و ترم رو کلید اصلی در نظر گرفتید؛ به نظرتون بهتره که کلید اصلی رو به ستون جدا در نظر بگیریم یا همین کاری که شما انجام دادید؟ لطفا مزایا و معایب هر کدام را بفرمائید.  
 با سپاس فراوان

نویسنده: محمد سلیم آبادی

تاریخ: ۲۲:۳۲ ۱۳۹۲/۰۵/۲۶

این امکان هم وجود داره که یک ستون دیگه به جدول اضافه کنید و آن را به عنوان PK در نظر بگیرید. اما باید به این نکته بسیار مهم نیز توجه داشته باشید که همیشه آن دو ستون (کد دانشجو و ترم) را همینطور به حال خود رها کرد. با این فرض که با اضافه شدن این ستون دیگه هیچ دو سطر تکراری به خاطر uniqueness بودن PK نخواهیم داشت.  
 شما لازمه که یک قید منحصر بفرد تکریمی در کنار PK برای آن دو ستون در جدول ایجاد کنید. تا به ازای یک ترم معین و یک دانشجو معین تنها یک معدل ثبت بشه.  
 پس با لحاظ توضیحات فوق جدول به این شکل در می آید:

```
create table Avgs
(
    identifier int not null identity(1,1) primary key,
    student_id varchar(10) not null
        references Students
    term_id tinyint not null
        references Terms
    average tinyint,
    check (average between 0 and 20),
    unique (student_id, term_id)
)
```

ستونی به نام identity وظیفه PK را به عهده می گیره. و از نوع identity هم هست.  
 دو ستون کد دانشجو و کد معدل کلیدهای خارجی هستند. و ترکیب این دو ستون برای حفظ یکپارچگی و جامعیت داده ها منحصر بفرد نظر گرفته شدن.  
 یک قید هم برای معدل گذاشته شده که معدل غیر متعارف در آن درج نشه.

به سناریوی زیر توجه کنید:

فرض کنید میخواهید بر اساس کد دانشجو و یک ترم معین در جدول برای بدست آوردن معدل جستجو داشته باشید. خوب لازم است که بر اساس آن دو ستون جستجو داشته باشید نه آن ستونی که به عنوان PK در نظر گرفته شده. پس محتویات ستون identity کاملاً مصنوعی و غیر طبیعی بوده و بطور مستقیم قابل استفاده نیست.

البته لازم به ذکر است که عموماً کلید اولیه همزمان unique clustered index نیز در نظر گرفته میشود. اگر داده‌های این ستون بطور متوالی و پشت سر هم در جدول درج نشدن باعث ایجاد fragmentation میشود. و لازمه که ایندکس rebuild بشود. و اگر کلید اولیه ترکیبی باشد کار در ارتباطات کمی دشوار میشود چون نیاز به کلیدهای خارجی ترکیبی نیز هست. در joinها نیز چون پیوند بر اساس کلید اولیه و کلید خارجی هست، هر چه کلید اولیه سبک‌تر باشد (حجم کمتری داشته باشد و از نوعی باشد که سریع‌تر توسط پردازنده پردازش بشود) سرعت پردازش نیز طبیعتاً افزایش پیدا میکند.

نویسنده: محسن خان  
تاریخ: ۱۳۹۲/۰۵/۲۷ ۰:۴۳

به این نوع کلیدها [surrogate key](#) هم می‌گویند.

نویسنده: محمد سلیم آبادی  
تاریخ: ۱۳۹۲/۰۵/۲۷ ۲:۲۱

بله همینطور.

سایت ویکی توضیحات خوبی راجب معایب و مزایای استفاده از surrogate key داده. به مرور بسیار جزئی که به معایب و مزایا داشتیم متوجه شدیم که در پست قبلیم از معایب به normalization و از مزایای به performance اش اشاره ای داشتیم.

## معایب شکل دوم نرمال

ابتدا اجازه دهید که مثالی از یک جدول بیاورم که به شکل دوم نرمال بوده ولی به شکل سوم نرمال نباشد. برای این منظور دو جدول زیر که هر دو در شکل سوم نرمال به سر می‌برند را با هم ترکیب می‌کنیم. ستون هایی از جدول که با نقاط قرمز رنگ علامت گذاری شده اند کلیدهای اولیه جدول می‌باشند.

کد دانشجو	نام دانشجو

نام رشته	نوع رشته	تعداد کل واحدها

اگر این دو جدول را با هم ترکیب کنیم، جدولی حاصل می‌شود که به فرم دوم نرمال است یعنی تمام خصیصه‌های غیر کلیدی وابسته به کلید اولیه (کد دانشجو) می‌باشند. اما همانطور که در بخش بعدی گفته خواهد شد، به شکل سوم نرمال نمی‌باشد.

کد دانشجو	نام دانشجو	نام رشته	نوع رشته
دانشجو ۱	نام ۱	رشته ۱	نوع ۱
دانشجو ۲	نام ۲	رشته ۱	نوع ۱
دانشجو ۳	نام ۳	رشته ۱	نوع ۱
دانشجو ۴	نام ۴	رشته ۲	نوع ۳
دانشجو ۵	نام ۵	رشته ۲	نوع ۳
دانشجو ۶	نام ۶	رشته ۳	نوع ۲

خصیصه "نوع رشته" به کلید اولیه جدول وابستگی تابعی دارد ولی از نوع متعددی (یعنی وابستگی از طریق خصیصه نام دانشجو می‌تواند بدست باید، چرا که نوع رشته به نام رشته و نام رشته به نام دانشجو وابستگی تابعی دارد)، این موضوع علاوه بر افزونگی اطلاعات باعث بی نظمی در به هنگام سازی خواهد شد. بطور نمونه

ایراد در عمل insert: این واقعیت که یک رشته خاص دارای یک نوع رشته خاص است را نمی‌توان اضافه کنیم، مثلاً نمی‌توانیم بیان کنیم که رشته ریاضی از نوع علوم پایه است مگر آن که دانشجویی باشد در رشته ریاضی مشغول به تحصیل است.

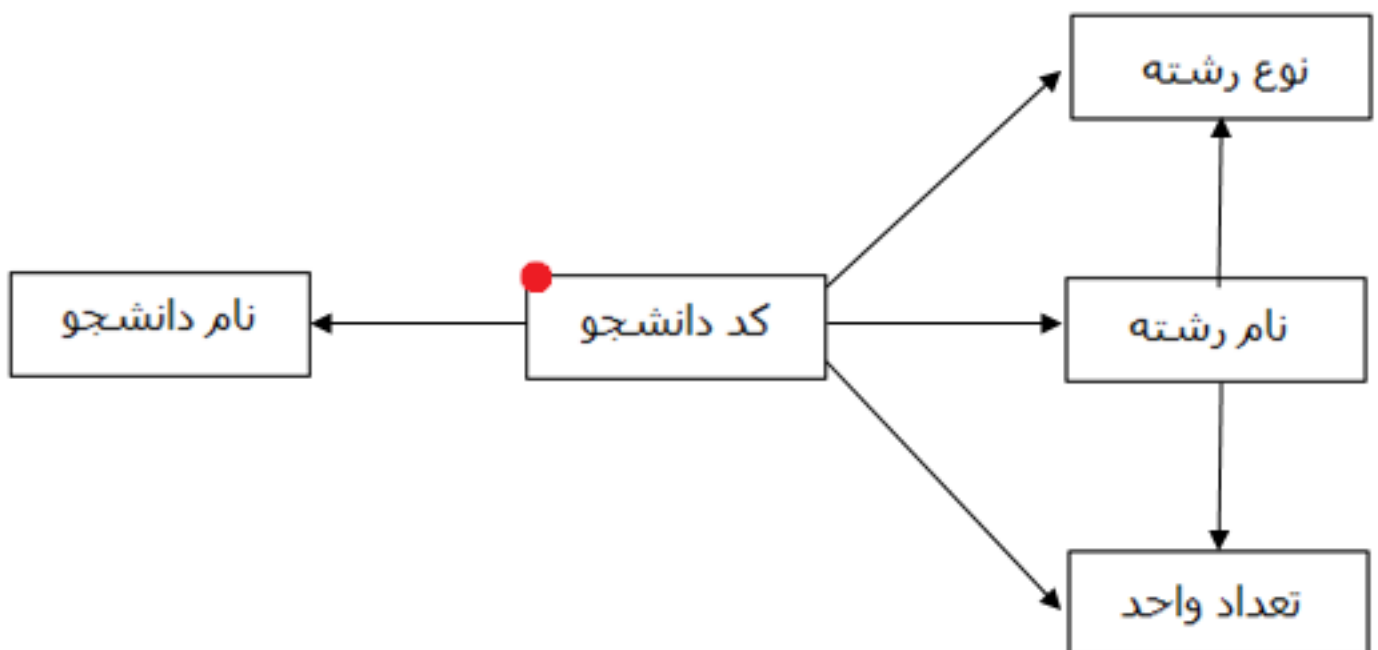
ایراد در عمل delete: با حذف یک دانشجو نه تنها اطلاعات مربوط به دانشجو بلکه اطلاعات مربوط به رشته تحصیلی نیز ممکن است حذف شود. مثلاً با حذف سطر مربوط به دانشجوی شماره 6 تمام اطلاعات مربوط به رشته شماره 3 نیز حذف خواهد شد.

ایراد در عمل update: اگر فرضاً بخواهیم نوع رشته ای به نام رشته 1 را تغییر دهیم به جای یک سطر باید چندین سطر (سه سطر در داده‌های نمونه) را بروز رسانی کنیم.

### تعریف شکل نرمال سوم

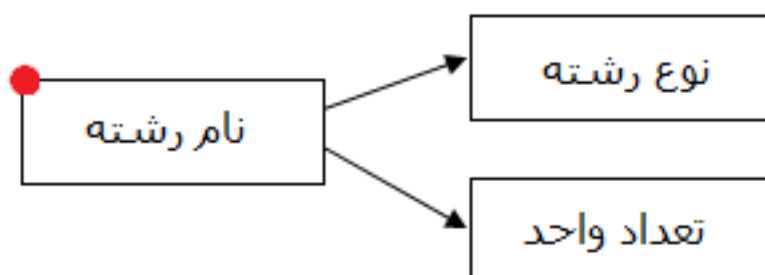
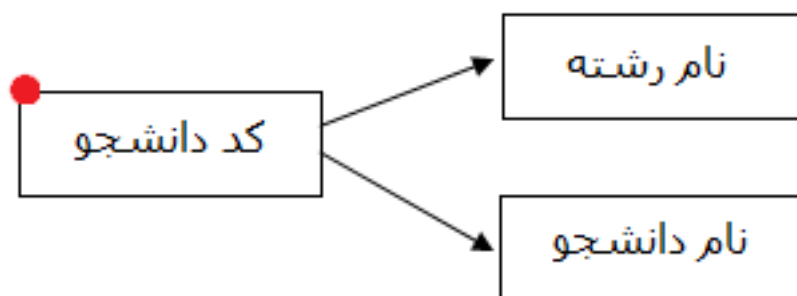
یک متغیر رابطه ای به شکل سوم نرمال است اگر به شکل دوم نرمال بوده و وابستگی‌های با واسطه (متعدی) نداشته باشد.

بر می‌گردیم به جدول ترکیبی قبل، نمودار FD جدول مورد نظر به صورت زیر است:



در این نمودار واضح است که وابستگی خصیصه نوع رشته به کد دانشجو از طریق خصیصه نام رشته بدست می‌آید. همینطور برای خصیصه "تعداد واحد". پس دو خصیصه‌ی نوع رشته و تعداد واحد با واسطه به کد دانشجو مرتبط هستند.

پس با تجزیه این نمودار به صورت زیر شرط شکل سوم نرمال هم محقق خواهد شد:



کافیه خصیصه کلید اولیه جدول "رشته ها" را به جدول "دانشجو" اضافه کنیم تا هر دو جدول به شکل نرمال سوم در بیایند. نقطه قرمز به معنای کلید اولیه و نقطه آبی به معنای کلید خارجی می باشد:

کد دانشجو	نام دانشجو	نام رشته

نام رشته	نوع رشته	تعداد واحد

موفق باشید

## نظرات خوانندگان

نویسنده: سعید  
تاریخ: ۱۳۹۱/۱۱/۱۴ ۱:۷

خوب، اگر این سه قسمت رو بخوایم با EF Code first مدل کنیم:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

public class Student
{
    public int Id { set; get; }
    public string Name { set; get; }

    //هر دانشجو چند ترم در دانشگاه خواهد بود
    public virtual ICollection<Semester> Semesters { set; get; }
    //هر دانشجو چندین واحد دارد
    public virtual ICollection<Unit> Units { set; get; }
}

public class Semester
{
    public int Id { set; get; }
    public string Name { set; get; }
    public int Average { set; get; }

    [ForeignKey("StudentId")]
    public virtual Student Student { set; get; }
    public int StudentId { set; get; }
}

public class Unit
{
    public int Id { set; get; }
    public string Name { set; get; }
    public string UnitType { set; get; }
    public int NumberOfUnits { set; get; }

    [ForeignKey("StudentId")]
    public virtual Student Student { set; get; }
    public int StudentId { set; get; }
}
```

به نظر می‌رسد که خاصیت Average جاش در کلاس Semester نیست. حتی به Unit هم نباید به صورت مستقیم ارتباط پیدا کنه. نیاز به یک کلاس دیگر هست که بتونه به ازای هر دانشجو، ترم و واحد، نمره ثبت کرد. میانگین، یک خاصیت آماری است که می‌تونه اصلاً لحاظ نشه و در گزارشات محاسبه بشه. و یا هر ترم یک سری واحد داره. اینطوری چطور؟ چون الان مشخص نیست در هر ترم چه واحدهایی برداشته.

نویسنده: محمد سلم ابادی  
تاریخ: ۱۳۹۱/۱۱/۱۴ ۸:۵۴

موضوعی که شما مطرح می‌کنید خارج از بحث مطرح شده است. من تصمیم نداشتم که یک محیط عملیاتی را پیاده سازی کنم. تنها مثال هایی برای درک بهتر موضوع آوردم. بله میانگین یک خاصیت آماری است، اما ما می‌توانیم برای سرعت بخشیدن به query هایمان برای بدست آوردن معدل، آن را بصورت فیزیکی ذخیره داشته باشیم. چون معدل بعد از ثابت و تعیین شدن دیگر تغییر نخواهد کرد.

نویسنده: سعید  
تاریخ: ۱۳۹۱/۱۱/۱۴ ۹:۱۰

مشکلی که بودن میانگین در کلاس ترم ایجاد می‌کنه وابسته کردن آن به دانشجو است درحالیکه ترم باید یک موجودیت واحد و مستقل باشد. با این طراحی فعلی باید کل اطلاعات ثابت یک ترم به ازای هر دانشجو یکبار دیگر هم ثبت شود.



یک نکته‌ای رو چند وقت قبل حین کار با ef بهش برخوردم که جالب بود. از دید ef ، کلید خارجی یا کلید اصلی «فقط خواندنی» هستند. یعنی اگر در اینجا کسی بخواد نام رشته رو تغییر بده مشکل ساز خواهد شد.

به دو دلیل:

- استفاده از رشته‌ها نسبت به یک عدد چون طولانی‌تر هستند کندتر است برای حالت تعریف کلید

- اگر تغییری قرار است رخ دهد، باید به تمام جداول اعمال شود. (تعریف اطلاعات تکراری)

در یک برنامه فروشگاه، جداول مشتری و خریدهای او را در نظر بگیرید. خرید 3 سال قبل مشتری خاصی به آدرس قبلی او ارسال شده‌است. خرید امروز او به آدرس جدید او ارسال خواهد شد. سؤال: آیا با وارد کردن و به روز رسانی آدرس جدید مشتری، باید سابقه اطلاعاتی قبلی او حذف شود؟ اجناس ارسالی پیشین او، واقعا به آدرس دیگری ارسال شده‌اند و نه به آدرس جدید او. چگونه باید اینگونه اطلاعاتی را که در طول زمان تغییر می‌کنند، در بانک‌های اطلاعاتی رابطه‌ای نرمال شده مدیریت کرد؟ از این نمونه‌ها در دنیای کاری واقعی بسیارند. برای مثال قیمت اجناس نیز چنین وضعی را دارند. یک بستنی مگنوم، سال قبل 300 تومان بود؛ امسال شده است 1500 تومان. یک سطل ماست 2500 تومان بود؛ امروز همان سطل ماست 6500 تومان است. چطور باید سابقه فروش این اجناس را نگهداری کرد؟

## منابع مطالعاتی مرتبط

[این موضوع](#) اینقدر مهم است که تابحال چندین کتاب در مورد آن تالیف شده است:

[Temporal Data & the Relational Model](#)

[Trees and Hierarchies in SQL](#)

[Developing Time-Oriented Database Applications in SQL](#)

[Temporal Data: Time and Relational Databases](#)

[Temporal Database Entries for the Springer Encyclopedia of Database Systems](#)

[Temporal Database Management](#)

نکته مهمی که در این مآخذ وجود دارند، واژه کلیدی « [Temporal data](#) » است که می‌تواند در جستجوهای اینترنتی بسیار مفید واقع شود.

## بررسی ابعاد زمان

فرض کنید کارمندی را استخدام کرده‌اید که ساعتی 2000 تومان از ابتدای فروردین ماه حقوق دریافت می‌کند. حقوق این شخص از ابتدای مهرماه قرار است به ساعتی 2400 تومان افزایش یابد. اگر مامور مالیات در بهمن ماه در مورد حقوق این شخص سؤال پرسید، ما چه پاسخی را باید ارائه دهیم؟ قطعاً در بهمن ماه عنوان می‌کنیم که حقوقش ساعتی 2400 تومان است؛ اما واقعیت این است که این عدد از ابتدای استخدام او ثابت نبوده است و باید تاریخچه تغییرات آن، در نحوه محاسبه مالیات سال جاری لحاظ شود.

بنابراین در مدل سازی این سیستم به دو زمان نیاز داریم:

الف) actual time یا زمان رخ دادن واقعه‌ای. برای مثال حقوق شخصی در تاریخ ابتدای مهر ماه تغییر کرده است. به این تاریخ در منابع مختلف Valid time نیز گفته می‌شود.

ب) record time یا زمان ثبت یک واقعه؛ مثلاً زمان پرداخت حقوق. به آن Transaction time هم گفته شده است. یک مثال:

record date	actual date	حقوق دریافتی
1392/01/01	1392/01/01	روز/2000
1392/02/01	1392/01/01	روز/2000
1392/07/01	1392/07/01	روز/2400
1392/17/01	1392/07/01	روز/2400

در این لیست، ریز حقوق پرداختی به یک شخص را ملاحظه می‌کنید. actual date ها، زمان‌هایی هستند که حقوق پایه شخص در

آن‌ها تغییر کرده و record date زمان‌هایی هستند که به شخص حقوق داده شده‌است. به ترکیب Valid Time و Transaction Time، اصطلاحاً Bitemporal data می‌گویند.

### مشکلات طراحی‌های متداول اطلاعات وابسته به زمان

در طراحی‌های متداول، عموماً یک جدول کارمندان وجود دارد و یک جدول لیست حقوق‌های پرداختی. رکوردهای لیست حقوق‌های پرداختی نیز توسط یک کلید خارجی به اطلاعات هر کارمند متصل است؛ از این جهت که نمی‌خواهیم اطلاعاتی تکراری را در جدول لیست حقوقی ثبت کنیم و طراحی نرمال سازی شده‌ای مدنظر می‌باشد. خوب؛ اول مهرماه حقوق شخصی تغییر کرده است. بنابراین کارمند بخش مالی اطلاعات شخص را به روز می‌کند. با این کار، کل سابقه حقوق‌های پرداختی شخص نیز از بین خواهد رفت. چون وجود این کلید خارجی به معنای استفاده از آخرین اطلاعات به روز شده یک کارمند در جدول لیست حقوقی است. الان اگر از جدول لیست حقوقی گزارش بگیریم، کارمندان همواره از آخرین حقوق به روز شده خودشان استفاده خواهند کرد.

### راه حل‌های متفاوت مدل سازی اطلاعات وابسته به زمان

برای رفع این مشکل مهم، راه حل‌های متفاوتی وجود دارند که در ادامه آن‌ها را بررسی خواهیم کرد.

#### الف) نگهداری اطلاعات وابسته به زمان در جدول نهایی مرتبط

اگر حقوق پایه شخص در زمان‌های مختلف تغییر می‌کند، بهتر است عدد نهایی این حقوق پرداختی نیز در یک فیلد مشخص، در همان جدول لیست حقوقی ثبت شود. این مورد به معنای داشتن «داده‌ای تکراری» نیست. از این جهت که داده‌ای تکراری است که اطلاعات آن در تمام زمان‌ها، دارای یک مقدار و مفهوم باشد و اطلاعات حقوق یک شخص اینچنین نیست.

#### ب) نگهداری اطلاعات تغییرات حقوقی در یک جدول جداگانه

یک جدول ثانویه حقوق جاری کارمندان مرتبط با جدول اصلی کارمندان باید ایجاد شود. در این جدول هر رکورد آن باید دارای بازه زمانی (valid\_start\_time و valid\_end\_time) مشخصی باشد. مثلاً از تاریخ X تا تاریخ Y، حقوق کارمند شماره 11، 2000 تومان در ساعت بوده است. از تاریخ H تا تاریخ Z اطلاعات دیگری ثبت خواهند شد. به این ترتیب با گزارشگیری از جدول لیست حقوق‌های پرداخت شده، سابقه گذشته اشخاص محو نشده و هر رکورد بر اساس قرارگیری در یک بازه زمانی ثبت شده در جدول ثانویه حقوق جاری کارمندان تفسیر می‌شود. در این حالت باید دقت داشت که بازه‌های زمانی تعریف شده، با هم تداخل نکنند و برنامه ثبت کننده اطلاعات باید این مساله را به ازای هر کارمند کنترل کند و یا با ثبت record\_date، اجازه ثبت بازه‌های تکراری را نیز بدهد (توضیحات در قسمت بعد). به این جدول، یک Temporal table نیز گفته می‌شود. نمونه دیگر آن، نگهداری قیمت یک کالا است از یک تاریخ تا تاریخی مشخص. به این ترتیب می‌توان کوئری گرفت که بستنی مگنوم فروخته شده در ماه آبان سال قبل، بر مبنای قیمت آن زمان، دقیقاً چقدر فروش کرده است و نه اینکه صرفاً بر اساس آخرین قیمت روز این کالا گزارشگیری کنیم که در این حالت اطلاعات نهایی استخراج شده صحیح نیستند. حال اگر به این طراحی در جدولی دیگر Transaction time یا زمان ثبت یک رکورد یا زمان ثبت یک فروش را هم اضافه کنیم، به جدول حاصل Bitemporal Tables می‌گویند.

### مدیریت به روز رسانی‌ها در جدول Temporal

در جدول Temporal، حذف فیزیکی اطلاعات مطلقاً ممنوع است؛ چون سابقه سیستم را تخریب می‌کند. اگر اطلاعاتی در این جدول دیگر معتبر نیست باید تنها تاریخ پایان دوره آن به روز شوند یا یک رکورد جدید بر اساس بازه‌ای جدید ثبت گردد. همچنین به روز رسانی‌ها در این جدول نیز معادل هستند با یک Insert جدید به همراه فیلد record\_date و نه به روز رسانی واقعی یک رکورد قبلی (شبیه به سیستم‌های حسابداری باید عمل کرد). یک مثال:

فرض کنید حقوق کارمندی که مثال زده شد، در مهرماه به ساعتی 2400 تومان افزایش یافته است و حقوق نهایی نیز پرداخته شده است. بعد از یک ماه مشخص می‌شود که مدیر عامل سیستم گفته بوده است که ساعتی 2500 تومان و نه ساعتی 2400 تومان! (از این نوع مسایل در دنیای واقعی زیاد رخ می‌دهند!) خوب؛ اکنون چه باید کرد؟ آیا باید رفت و رکورد ساعتی 2400 تومان را به روز کرد؟ خیر. چون سابقه پرداخت واقعی صورت گرفته را تخریب می‌کند. به روز رسانی شما ابداً به این معنا نخواهد بود که دریافتی

واقعی شخص در آن تاریخ خاص، ساعتی 2500 بوده است.

بنابراین در جداول Temporal، تنها «تغییرات افزودنی» مجاز هستند و این تغییرات همواره به عنوان آخرین رکورد جدول ثبت می‌شوند. به این ترتیب می‌توان اصطلاحاً «مابه‌التفاوت» حقوق پرداخت نشده را به شخص خاصی، محاسبه و پرداخت کرد (می‌دانیم در یک بازه زمانی خاص به او چقد حقوق داده‌ایم. همچنین می‌دانیم که این بازه در یک record\_date دیگر لغو و با عددی دیگر، جایگزین شده‌است).

برای مطالعه بیشتر

[Bitemporal Database Table Design - The Basics](#)

[Temporal Data Techniques in SQL](#)

[Database Design: A Point in Time Architecture](#)

[Temporal database](#)

[Temporal Patterns](#)

راه حلی دیگر؛ استفاده از بانک‌های اطلاعاتی NoSQL

بانک‌های اطلاعاتی NoSQL برخلاف بانک‌های اطلاعاتی رابطه‌ای برای اعمال Read بهینه‌سازی می‌شوند و نه برای Write. در چند دهه قبل که بانک‌های اطلاعاتی رابطه‌ای پدیدار شدند، یک سخت دیسک 10 مگابایتی حدود 4000 دلار قیمت داشته است. به همین جهت مباحث نرمال‌سازی اطلاعات و ذخیره نکردن اطلاعات تکراری تا این حد در این نوع بانک‌های اطلاعاتی مهم بوده است. اما در بانک‌های اطلاعاتی NoSQL امروزی، اگر قرار است فیش حقوقی شخصی ثبت شود، می‌توان کل اطلاعات جاری او را یکجا داخل یک سند ثبت کرد (از اطلاعات شخص در آن تاریخ تا اطلاعات تمام اجزای فیش حقوقی در قالب یک شیء تو در توی JSON). به همین جهت بسیار سریع هستند برای اعمال Read و گزارش‌گیری. همچنین این نوع سیستم‌ها برای نگهداری نگارش‌های مختلف یک سند بهینه‌سازی شده‌اند و جزو ساختار توکار آن‌ها است. بنابراین در این نوع سیستم‌ها اگر نیاز است از یک سند خاصی گزارش بگیریم، دقیقاً اطلاعات همان تاریخ خاص را دارا است و اگر اطلاعات پایه سیستم را به روز کنیم، از امروز به بعد در سندهای جدید ثبت خواهد شد. این نوع سیستم‌ها رابطه‌ای نیستند و بسیاری از مباحث نرمال‌سازی اطلاعات در آن‌ها ضرورتی ندارد. قرار است یک فیش حقوقی شخص را نمایش دهیم؟ خوب، چرا تمام اطلاعات مورد نیاز او را در قالب یک شیء JSON تو در توی حاضر و آماده نداشته باشیم؟

### نظرات خوانندگان

نویسنده: ناصر  
تاریخ: ۰۵:۵۴ ۱۳۹۲/۰۷/۱۷

من هم قبلا از این روش برای قیمت‌های جدید و قدیم یک کالا در یک سیستم فروشگاهی استفاده کردم. با نوشتن یک تریگر که به محض تغییر روی قیمت کالا ، بلافاصله داخل یک جدول دیگه این تغییرات درج میشد. و موقع نمایش ، قیمت جدید و قدیم هر دو با هم به مشتری نمایش داده میشد.

نویسنده: سیروس  
تاریخ: ۱۱:۱۱ ۱۳۹۲/۰۷/۱۸

به نظر من در نگهداری به این روش نیازی به تاریخ پایان نیست، مثلا هنگام تغییر قیمت کالا، رکوردی با تاریخ روز در جدول temporal ثبت می‌کنیم و در تغییر دوباره رکورد جدید دیگری ثبت می‌شود. کارکردن به این روش آسانتر به نظر می‌رسد و یک فیلد کمتر داریم و نیازی هم به چک کردن درست بودن بازه‌ی تاریخی نیست.

Date	Price	ProdcutId
1392/01/01	1000	1
1392/03/05	1500	1
1392/06/27	1780	1

نویسنده: ایلیا اکبری فرد  
تاریخ: ۱۱:۴۵ ۱۳۹۳/۰۴/۰۲

البته روش شما برای حالتی مناسب است که بازه‌های تاریخی به هم متصل باشند.

نویسنده: وحید نصیری  
تاریخ: ۱۰:۲۶ ۱۳۹۴/۰۳/۰۸

### یک نکته‌ی تکمیلی

SQL Server 2016 مفهومی به نام [Temporal Tables](#) را پشتیبانی می‌کند.

در این مقاله قصد داریم اطلاعات مفیدی را در مورد طراحی دیتابیس‌های چند زبانه، در اختیار شما بگذاریم. مدتی قبل به طراحی دیتابیس‌های چند زبانه بودن توضیحات کالا را برای مشتریانی از کشورهای مختلف پشتیبانی می‌کرد، نیاز داشتم. وقتی شروع به پیاده سازی طرح دیتابیس کردم، جواب سرراست نبود. زمانیکه در وب برای بهترین راه جستجو می‌کردم، با نظرات و روش‌های زیادی مواجه شدم. در ادامه بعضی از روش‌های محبوب را بیان می‌کنم.

**ستون اضافی :** این ساده‌ترین راه است و به ازای هر ستونی که نیاز به ترجمه داشته باشد، ستون اضافی در نظر می‌گیریم.

```
CREATE TABLE app_product (
  Id Int IDENTITY NOT NULL,
  Description_en Text,
  Description_pl Text,
  PRIMARY KEY (Id)
);
```

مزایا :

سادگی

کوئری‌های آسان (بدون نیاز به join)

معایب :

اضافه کردن زبان جدید نیاز به تغییر جداولی که چند زبانه هستند دارد

اگر وارد کردن داده برای همه زبان‌ها الزامی نباشد (بعضی جاها فقط زبان پیش فرض الزامی است) ممکن است داده‌های زیاد و یا فیلدهای خالی در دیتابیس ایجاد شود

نگهداری آن مشکل است

یک جدول ترجمه : این روش تمیزترین راه از دیدگاه ساختار دیتابیس به نظر می‌رسد. شما همه متن‌هایی را که نیاز به ترجمه دارد، در یک جدول ذخیره می‌کنید.

```
CREATE TABLE ref_language (
  Code Char(2) NOT NULL,
  Name Varchar(20) NOT NULL,
  PRIMARY KEY (Code)
);

CREATE TABLE app_translation (
  Id Int IDENTITY NOT NULL,
  PRIMARY KEY (Id)
);

CREATE TABLE app_translation_entry (
  TranslationId Int NOT NULL,
  LanguageCode Char(2) NOT NULL,
  Text Text NOT NULL,
  FOREIGN KEY (TranslationId) REFERENCES app_translation(Id),
  FOREIGN KEY (LanguageCode) REFERENCES ref_language(Code)
);

CREATE TABLE app_product (
  Id Int IDENTITY NOT NULL,
  Description Int NOT NULL,
  PRIMARY KEY (Id),
  FOREIGN KEY (Description) REFERENCES app_translation(Id)
);
```

مزایا :

اضافه کردن زبان جدید به تغییر طرح دیتابیس نیاز ندارد

به نظر تمیز است و رویکرد رابطه‌ای دارد

همه ترجمه‌ها در یک مکان است (بعضی‌ها می‌گویند این جز معایب است چون امکان خوانایی و نگهداری کمتر است)

معایب :

کوئری‌های پیچیده (به join های چندگانه نیاز دارد تا شرح کالا را به درستی نمایش دهد)

پیچیدگی زیاد

**جدول ترجمه اضافی به ازای هر جدول چند زبانه :** برای هر جدولی که نیاز به ترجمه دارد یک جدول اضافی ساخته می‌شود.

جدول اصلی داده‌های غیر مرتبط به زبان و جدول دوم همه اطلاعات ترجمه شده را ذخیره می‌کند.

```
CREATE TABLE ref_language (  
    Code Char(2) NOT NULL,  
    Name Varchar(20) NOT NULL,  
    PRIMARY KEY (Code)  
);  
  
CREATE TABLE app_product (  
    Id Int IDENTITY NOT NULL,  
    PRIMARY KEY (Id)  
);  
  
CREATE TABLE app_product_translation (  
    ProductId Int NOT NULL,  
    LanguageCode Char(2) NOT NULL,  
    Description Text NOT NULL,  
    FOREIGN KEY (ProductId) REFERENCES app_product(Id),  
    FOREIGN KEY (LanguageCode) REFERENCES ref_language(Code)  
);
```

مزایا :

اضافه کردن زبان جدید به تغییر طرح دیتابیس نیاز ندارد

کوئری‌های نسبتاً ساده است

معایب :

ممکن است تعداد جداول دو برابر شود

سه مثال نشان داده شده در بالا به ما ایده می‌دهند که چگونه روش‌های مختلف ممکن است استفاده شوند. البته اینجا همه گزینه‌های ممکن نیستند، فقط محبوبترین روش‌ها هستند و شما می‌توانید آنها را ویرایش کنید؛ به عنوان مثال با تعریف View های اضافی که join های پیچیده شما را در کدها، ذخیره می‌کنند. راه حلی که شما انتخاب می‌کنید به نیازمندی‌های پروژه وابسته است. اگر شما به سادگی نیاز دارید و مطمئن هستید تعداد زبان‌های پشتیبانی کم و ثابت است، می‌توانید راه حل اول را انتخاب کنید. اگر به انعطاف پذیری بیشتری نیاز دارید، می‌توانید join های ساده در کوئری‌های چند زبانه را انتخاب کنید. راه حل سوم انتخاب مناسبی است.

منبع :

<http://fczaja.blogspot.com/2010/08/multilanguage-database-design.html>

## نظرات خوانندگان

نویسنده: ناصر فرجی  
تاریخ: ۱۶:۱۴ ۱۳۹۲/۰۸/۰۹

روشی که من خودم مدتی استفاده میکردم اضافه کردن یک فیلد language به هر تیبیل بود. موقع ثبت دیتا هر زبانی بود همون زبان رو تو این فیلد نگه میداشتم. مثلا برای فارسی fa و انگلیسی en و ... , موقع نمایش هم بر اساس زبان سایت یک کوئری ساده گرفته می‌شد و اطلاعات زبان مورد نظر لود می‌شد.

نویسنده: محسن موسوی  
تاریخ: ۱۹:۲۹ ۱۳۹۲/۰۸/۰۹

با سلام و تشکر از مطلب خوبتون  
طراحی با یک جدول زبان و نگه داشتن کلید خارجی در جداول مربوطه بهتر میشه  
چندید ساله که از این طراحی استفاده میکنیم و جواب داده.  
یکی از مزایایی که داره میتونی مدیریت سامانه را نسبت به هر زبان بطور مستقل انجام بدی  
و هر جا که نیاز داشتی همزمان چند رکورد را درج کنید.  
ساده و روان.  
البته استراتژی سیستم استفاده از الگوی مناسب رو توجیه میکنه.

نویسنده: محمد پهلوان  
تاریخ: ۱۰:۵۰ ۱۳۹۲/۰۸/۱۰

استفاده از یک فیلد language در هر تیبیل باعث می‌شود شما برای یک موجودیت کالا مثلا در سه زبان مختلف 3 رکورد درج کنید که در ارتباط این کالا با دیگر جداول دچار مشکل خواهید شد

نویسنده: محمد پهلوان  
تاریخ: ۱۱:۳۶ ۱۳۹۲/۰۸/۱۰

روش دوم واقعا روش تمیز و جمع و جوریه اما کوئری هاش واقعا پیچیده اس و انعطاف نداره. به نظر من مخصوصا برای استفاده از EF روش سوم روش مناسبتری باشه. این کوئری‌ها رو با توجه به حجم داده زیاد و سایت پرتراфик بررسی کنید.  
خودم بین روش دو و سه مرددم. از دوستان می‌خوام نظراتشون را با دلائل بیان کنن تا به نتیجه خوبی برسیم

نویسنده: محسن خان  
تاریخ: ۱۱:۵۱ ۱۳۹۲/۰۸/۱۰

مطلب [Globalization در ASP.NET MVC - قسمت ششم](#) در همین راستا مفید است.

نویسنده: بختیاری  
تاریخ: ۱۵:۱۶ ۱۳۹۲/۰۸/۱۰

سلام  
من روش آقای موسوی را منطقی می‌دانم خودم هم با این روش یک سایت طراحی کردم که الان درست و بدون مشکل کار می‌کند

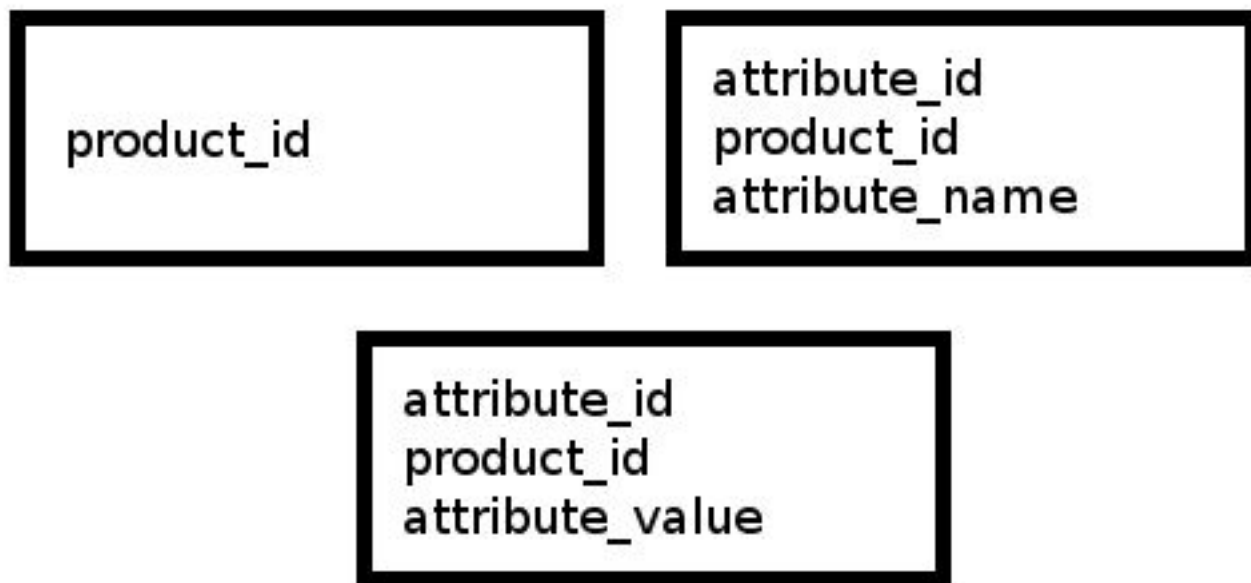


EAV مخفف ( Entity Attribute Value ) می‌باشد، مدلی از طراحی دیتابیس که کاربر را به آیتم‌های ثابت محدود نمی‌کند، فرض کنید در یک فروشگاه می‌خواهید چندین کالا بفروشید هر کالا هم برای خودش ویژگی‌های منحصر به فرد دارد، آیا با ویژگی‌های ثابت برای کالاهای متفاوت می‌توان پاسخگوی نیاز مشتری بود؟ یقیناً پاسخ منفی خواهد بود. موجودیت ( Entity ): در یک سیستم می‌تواند کالا، مشتری، فروشنده و... باشد. ویژگی ( Attribute ): برای کالا: رنگ، وزن و... برای مشتری: نام، تلفن، آدرس و... می‌باشد مقدار ( value ) : هر ویژگی برابر مقداری می‌باشد مثلاً برای رنگ‌ها آبی، قرمز و.. می‌باشد جداول پایه طراحی شده:



ورود داده‌ها:

شیوه ورود داده‌ها را برای موجودیت کالا بیان می‌کنیم  
ابتدا کالا در جدول موجودیت ثبت می‌گردد  
سپس عنوان ویژگی‌های آن مانند رنگ، وزن و... در جدول ویژگی‌ها ثبت می‌گردد.  
مقدار هر ویژگی هم در جدول مقدارها ثبت می‌شود.  
در زیر شیوه ذخیره به صورت شکل مشاهده می‌کنید.



شیوه خواندن داده ها:

این قسمت هم به راحتی با 2 inner join می توان به کالا، ویژگی ها و مقادیر آن دست پیدا کرد.

نکات:

نکته 1: این 3 جدول را باید برای هر موجودیت قابل توسعه ایجاد کرد، مثلا برای کالا، مشتری و...

نکته 2: می توان برای گروه بندی کالاها و همچنین ویژگی ها جداول جداگانه ایی تعریف کرد.

نکته 3: از مهمترین ویژگی های این تفکر قابل گسترش بودن سیستم می باشد.

نکته 4: می توان برای آیتم هایی مثل نمایش داده شود یا خیر، چیدمان نمایش و... آیتم هایی به جدول ویژگی ها اضافه کرد.

نکته 5: این مدل در نرم افزار magento استفاده شده است.

همچنین جهت مطالعه بیشتر ساختار دیتابیس مجنتو در لینک زیر می باشد.

[MAGENTO\\_v1.0.19700---Database-Diagram.zip](#)

[Entity-attribute-value model](#) منابع:

## نظرات خوانندگان

نویسنده: افشار محبی  
تاریخ: ۱۳۹۲/۰۹/۰۱ ۲۳:۵۰

روش مفید و موثری است. ولی نمی‌دانم با داده‌های حجم بالا هم می‌توان خوب کار کند یا نه. علاوه بر این روش‌های متعارف query روی دیتابیس را به چالش می‌کشد. به عنوان مثال دیگر نمی‌توان با یک Query معمولی فهرست کالاها و مشخصات و گروه بندی بر اساس فلان Attribute را استخراج کرد.

نویسنده: حسین صفدری  
تاریخ: ۱۳۹۲/۰۹/۰۲ ۸:۳۴

اگر جدول را خوب ایندکس گذاری کنیم مشکل سرعت حل می‌شود از طرفی نحوه نمایش هم باید بهینه باشد، مثلا 10 تا 10 نمایش داده دهیم و...  
اتفاقا قسمت قشنگ ماجرا اینجا می‌شود که شما تمام Attribute‌های یک گروه کالا را به کاربر نمایش می‌دهید و بنا به درخواست کاربر داده‌های آن ویژگی را می‌توان نمایش داد.

نویسنده: سجاد ف  
تاریخ: ۱۳۹۲/۰۹/۰۲ ۸:۵۱

میشه توضیح بدین اگه چند Attribute با نوع مختلف داشتیم چطور ذخیره کنیم ؟ نوع Attribute Value باید string باشه ؟

نویسنده: vahid  
تاریخ: ۱۳۹۲/۰۹/۰۲ ۸:۵۸

باسلام  
سایت ebay بگونه ای طراحی شده که وقتی ما کالای را انتخاب میکنیم میتوانیم آن را با ویژگی‌ها خاص خود آن کالا آن را فیلتر نماییم و همان زمان تعداد موجودی آن کالا و قسمت نسبت به ویژگی‌ها و ... را می‌دهد چگونه به این سرعت عمل میکند؟ با آن حجم اطلاعات؟ ممنون از شما

نویسنده: محسن خان  
تاریخ: ۱۳۹۲/۰۹/۰۲ ۹:۳۳

[چه زمانی بهتر است از بانک‌های اطلاعاتی NoSQL استفاده کرد و چه زمانی خیر؟](#)

نویسنده: حسین صفدری  
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۴:۴۵

طوری که مجتو طراحی کرده است مطابق فایل ضمیمه برای هر type یک مجموعه جدول طراحی کرده است. فکر می‌کنم این شیوه بهینه می‌باشد.

نویسنده: حسین صفدری  
تاریخ: ۱۳۹۲/۰۹/۰۲ ۱۴:۴۷

آن طور که بنده اطلاع دارم، در دیتا بیس‌های NoSQL برای سیستم هایی با حجم داده‌های بزرگ قابل استفاده است (چند صد میلیون)، و شیوه طراحی آن به گونه ایی است که هرچه داده‌ها بیشتر شوند سرعت جستجو هم مطابق تابع نمایی بالا می‌رود.

نویسنده: حسین صفدری  
تاریخ: ۱۳۹۲/۰۹/۰۲ ۲۳:۴۱

سوال قشنگی بود، طوری که امروز کلی تو سایت‌ها راجع بهش مطلب خوندم.. اگر شما ویژگی‌های کالا را طبق مدل بالا انجام دهید و آن‌ها را گروه بندی کنید می‌توانید به کاربر نمایش دهید و از طرفی به ازای افزودن هر فیلتر در عبارت شرطی خود AND و ویژگی و مقدار آن را می‌آوریم.

اما ebay! طراحی دیتابیس سایت ebay منحصر به فرد است، همچنین طبق [اینجا](#) قسمت search suggestion and the internal Cloud Manager از MongoDB که بر پایه NO SQL می‌باشد کار می‌کند. همچنین جهت آشنایی بیشتر با ebay این [لینک](#) را ببینید.

نویسنده: افشار محبی  
تاریخ: ۱۳۹۲/۰۹/۰۳ ۱۰:۲۶

چرا توی هر موضوع جدیدی ذهن بیشتر افراد فوراً منحرف می‌شود به سمت Amazon و eBay و فیسبوک و بقیه غول‌ها؟ مگه ما تو ایران چند تا از این غول‌ها داریم یا خواهیم داشت؟ بیشترین نیاز نرم‌افزارها و مشاغل مختلف در ایران نیازهای کوچک و متوسط هستند.

کاش اینقدر غیر عملی و غیر کاربردی فکر نمی‌کردیم. در داخل کشور بیشتر از این که نیاز به ارسال ماهواره به فضا داشته باشیم نیاز به داشتن یک خودروی معمولی داریم که سالی چند هزار کشته و زخمی جاده و خیابان‌ها نداشته باشیم.

ببخشید که از موضوع خارج شدم.

نویسنده: محسن خان  
تاریخ: ۱۳۹۲/۰۹/۰۳ ۱۱:۱۶

این روحیه شما جستجوگری را از بین می‌برد. تفکر در مورد راه‌های مختلف را منع می‌کند. اقلان به روش‌های عهد عتیق طراحی را که الزاما بهینه نیستند، ترویج می‌کند. جستجوی در مورد راه‌های NoSQL الزاما به معنای استفاده از آن‌ها نیست ولی حداقل دید شخص را نسبت به الگوریتم‌ها و طرز تفکرهای مختلف موجود جهت حل مسایل باز می‌کند. خیلی‌ها مثلا جبهه می‌گیرند در مورد ORM. به این افراد باید گفت، اشکالی نداره. استفاده نکنید. حداقل طراحی اون‌ها رو مطالعه کنید که توسط بزرگان دنیا انجام شده و ارزش درس یاد بگیرید تا کدهای SQL Helper مشکل داری رو طراحی نکنید. برید SQL بنویسید بجای LINQ. اما حداقل یادگیری اونی که اومده لایه DAL جنریک درست کرده، طراحی‌اش چطوری بوده. دو تا نکته ارزش یاد بگیرید. نمی‌خواهین با MVC کار کنید، مهم نیست. حداقل طراحی‌اش رو بررسی کنید که چطور تونسته ViewState رو حذف کنه اما باز هم بعد از post back به سرور می‌تونه مقادیر وارد شده در فرم‌ها رو در صورت نیاز حفظ کنه.

مورد دوم اینکه اون تعداد کشته‌ها ربط مستقیم داره به میزان بی‌سوادی در کشور. [مطابق نظر معاون وزیر آموزش و پرورش](#) در سال قبل «در کشور نزدیک به 9 میلیون و 700 هزار نفر خواندن و نوشتن بلد نیستند و بیش از 10 میلیون نفر نیز تحصیلات حداکثر پنجم ابتدایی دارند.» خوب این‌ها مسلما مشکل‌زا هستند. همه چیز تقصیر ماشین و جاده نیست. این‌ها هم کسانی هستند که قانع هستند به آنچه که دارند و نیازی برای پیشرفت حس نمی‌کنند.

نویسنده: افشار محبی  
تاریخ: ۱۳۹۲/۰۹/۰۳ ۱۱:۴۱

منظور من این نیست که از روش‌ها و فناوری‌ها به روز و جدید استفاده نکنیم. درست برعکس. هر چه به روزتر باشیم بهتر است. نظر من این است که به کاربردهایی که در کشور داریم فکر کنیم نه چیزی که شاید فقط در آمریکا کاربرد داشته باشد. به عبارتی دیگر از روش‌ها و فناوری‌های روز استفاده کنیم منتها برای نرم‌افزارهایی که در داخل کشور کاربرد دارند نه فیسبوک و امثالهم.

اون قدر تمرکز نکنیم روی مدل دیتابیس فیسبوک که از دیتابیس‌های ساده مورد نیاز نرم‌افزارهای خودمان غافل شده و قدرت کار

بهینه روی آن را از دست بدهیم. مثال می‌زنم. در جمع‌های برنامه‌نویسی چه در دانشگاه‌ها، چه در شرکت‌ها چه در فضای اینترنت وقتی صحبت از دیتابیس می‌شود همه طوری حرف می‌زنند که انگار طراحی دیتابیس فیسبوک و امثال آن را فوت آب هستند اما وقتی پای عمل می‌رسد در عوض کردن دیتابیس یک نرم‌افزار کوچک از MS SQL به Express یا CE یا SQLite از زمین تا آسمان مشکل دارند و نرم‌افزاری که تولید کرده‌اند فقط روی یکی دو مدل ویندوز با یک سری config‌های خیلی محدود قابل اجرا هستند. بخشی از این مشکل به خاطر عدم تمرکز روی کاربردهای رایج و همه روزه از دیتابیس، کامپیوتر، نرم‌افزار، فناوری و غیره است.

البته و صد البته مطالعه و بررسی مواردی که حتی در ایران قابلیت اجرا ندارند خیلی هم خوب است و دیدگاه‌های جدیدی به آدم می‌دهد. منتها اول باید به عنوان یک مهندس نرم‌افزار خوب به وظایفمان که تولید و راهبری سیستم‌های نرم‌افزاری در حیطه کشور است عمل کنیم سپس به سراغ چنین مباحثی برویم.

ذهن من پر مثال است. در حالی که استاندارد ساختمان سازی در ایران بسیار پایین است و عمر ساختمان در حد ۳۰ سال است (یک چندم استاندارد جهانی)، مقاومت زیادی در برابر زلزله ندارند و هزار ایراد دیگر، آن وقت میایم دست به ساختن برج میلاد می‌زنیم. اگر واقعاً توان مهندسی داریم اول آن را جاهایی که ضروری‌تر است مصرف کنیم سپس جاهایی که اسم گنده دارند و برایمان «رزومه» می‌شود.

نویسنده: Programmer  
تاریخ: ۱۵:۱۳ ۱۳۹۲/۰۹/۰۳

اگر امکانش هست یک مثال از کد SQL مربوطه بزارید ممنون میشم

نویسنده: محسن خان  
تاریخ: ۱۷:۱۴ ۱۳۹۲/۰۹/۰۳

فایل MAGENTO v1.0.19700 - Database Schema [SpacedFormat].sql داخل فایل zip پیوست شده هست.

نویسنده: Programmer  
تاریخ: ۱۰:۲۱ ۱۳۹۲/۰۹/۰۴

البته منظورم دستور SELECT همراه با INNER JOIN هستش. میخوام ببینم آیا میشه کل تولیدات رو با تمام خواص انتخاب کنم؟ بعد برای ستون بفرض رنگ که بعضی تولیدات چنین ویژگی ای ندارند چه چیزی میاد؟

نویسنده: محمد  
تاریخ: ۱۰:۲۶ ۱۳۹۲/۰۹/۰۶

سلام

این مدل طراحی من رو به یاد کتاب The Data Model Resource می‌اندازه به نظرم کتاب خوب و فوق العاده ای برای طراحی انواع پایگاه‌های داده برای پروژه‌های خاص البته vol 1,2,3 داره من فقط کتاب یک و دو رو دیدم . ممنون از مطلب خوبتون

نویسنده: سعید  
تاریخ: ۱۳:۵۳ ۱۳۹۲/۱۰/۱۳

سلام

سوال اولم اینه که چه نیازی به کد محصول در جدول مقادیرها میباشد ؟ مگه کد محصول تو جدول ویژگی‌ها نیست؟! و از اون طرفی هر ویژگی یه کد منحصر به فرد داره (attribute\_id) که این کد نیز در جدول ویژگی‌ها وجود دارد . پس با Join کردن موجودیت و ویژگی توسط کد موجودیت و Join کردن نتیجه با جدول مقدار توسط کد ویژگی میتوان به تمام مقادیر یک موجودیت دست یافت از اونجایی که فکر کردم سوالم مربوط به این مورد هست میپرسم . در نرم افزارهای حسابداری ما تفصیلی‌ها رو گروه بندی میکنیم مثل گروه اشخاص ، کالا ، سهامداران ، بانک و ...

و هر کدام از این گروه‌ها دارای مقادیر خاصی هستند که در گروه دیگر وجود ندارد مثل کد فنی کالا و ...  
حالا سوالم اینه که به نظر شما برای این مورد از این روش مشه استفاده کرد ؟  
یعنی:

یک جدول برای تفصیلی ها  
یک جدول برای ویژگی‌های تفصیلی  
یک جدول برای مقادیر ویژگی ها  
یک جدول برای گروه‌های تفصیلی  
یک جدول هم برای ارتباط گروه تفصیلی با خود تفصیل (جهت مشخص کردن تفصیلی‌های موجود در یک گروه)  
با تشکر

نویسنده: حسین صفدری  
تاریخ: ۲۲:۴۵ ۱۳۹۲/۱۰/۱۳

سلام

در مورد سیستم حسابداری متاسفانه به صورت حرفه ایی بنده کار نکردم..  
این مدل معایبی هم دارد که selectهای پیچیده ایی باید نوشت، که روی عملکرد سیستم بسیار تاثیر می‌گذارد.  
بنده در این مقاله چون دیتابیس مجنتو را مطالعه کردم و با مدلی جدید آشنا شدم آن را با شما دوستان عزیز به اشتراک گذاشتم...

اعتماد و یا فقدان آن، عامل شماره یک مسدود کردن استفاده از نرم افزار به عنوان خدمات است. معماری پایگاه داده چند مستاجری برای رسیدگی به مشکل نرم افزار به عنوان سرویس (SaaS) که می‌تواند خدمات به تعدادی کلاینت ارائه کند استفاده می‌شود. معماری دیتابیس چند مستاجری وقتی مفید است که یک نمونه از دیتابیس به تعدادی کلاینت خدمات دهد. وقتی که نرم افزارهای محلی نصب می‌کنید نرم افزارهای به عنوان یک سرویس با مشتریان متمرکز، دسترسی به داده‌ها مبتنی بر شبکه با سربار کمتر را فراهم می‌کنند. اما به منظور برخورداری بیشتر از مزیت‌های یک نرم افزار سرویس، یک سازمان باید از سطحی از کنترل روی داده صرفنظر کند و به فروشنده نرم افزار جهت نگهداری و امنیت به دور از چشم آنها اعتماد کند.

برای به درست آوردن این اعتماد، یکی از بالاترین الویت‌ها، آینده نگری معماری نرم افزار و ساخت یک معماری داده است که باید هر دو قوی و به اندازه کافی ایمن باشد، این دو برای راضی کردن مستاجران و کلاینت‌هایی که علاقمند هستند کنترل داده‌های حیاتی تجارت خود را به شخص سومی واگذار نمایند، موثر است در حالی که برای اداره کردن و نگهداری مقرون به صرفه است.

## سه روش مدیریت چند مستاجری داده

دیتابیس‌های جداگانه برای هر مستاجر

دیتابیس مشترک و schema جداگانه برای هر مستاجر

دیتابیس مشترک و schema مشترک

انتخاب روش مناسب برای برنامه شما به عوامل زیر بستگی دارد :  
سایز دیتابیس هر مستاجر

تعداد مستاجران

تعداد کاربران هر مستاجر

نرخ رشد مستاجر

نرخ رشد دیتابیس مستاجر

امنیت

هزینه

## 1) دیتابیس‌های جداگانه برای هر مستاجر :

ذخیره سازی داده‌های مستاجران در دیتابیس‌های جداگانه ساده‌ترین روش است. در این روش هر مستاجر یک دیتابیس دارد. منابع و کدهای برنامه معمولاً در سرور بین همه مستاجران مشترک است اما هر مستاجر مجموعه ای از داده دارد که بطور منطقی از سایر مستاجران جدا شده است.

مزایا :

امنیت بیشتر

سهولت سفارشی سازی برای هر مستاجر

سهولت نگهداری ( Backup و Restore ) برای هر مستاجر

معایب:

برای نگهداری سخت افزار قوی مورد نیاز است

این روش هزینه بیشتری برای تجهیزات ( Backup و Restore ) برای هر مستاجر دارد

## 2) دیتابیس مشترک و schema جداگانه برای هر مستاجر :

خدمات دهی به چندین مستاجر در یک دیتابیس مشترک اما هر مستاجر یک مجموعه از جداول گروه بندی شده دارد که با Schema جدا شده است که برای هر مستاجر الزامی است.

مزایا :

برای دیتابیس برنامه های کوچک مناسب است. وقتی تعداد جداول برای هر مشتری کم است

هزینه کمتری نسبت به روش اول دارد

برای مشتریانی که نگران امنیت هستند، سطح منطقی مناسبی برای جداسازی داده ه وجود دارد

معایب:

اطلاعات مستاجران در صورت بروز خطا به سختی restore می شود

مدیریت آن برای دیتابیس های بزرگ مشکل است

## 3) دیتابیس مشترک و schema مشترک :

این روش شامل یک دیتابیس و یک مجموعه از جداول برای چندین مستاجر است. داده های جدول می تواند شامل رکوردهای هر مستاجر باشد

مزایا :

در مقایسه با روش قبلی، کمترین هزینه سخت افزاری را دارد

می تواند مستاجران بیشتری را در هر سرور پشتیبانی کند

قابلیت بروز رسانی آسان در یک جا برای همه مستاجران

مدیریت آسان دیتابیس و خطا و Restore و Backup

معایب:

امنیت بیشتری مورد نیاز است تا مطمئن شوید هیچکس به اطلاعات سایر مستاجران دسترسی ندارد.

می تواند روی کارایی کوثری ها تاثیر بگذارد چون تعداد رکوردها زیاد است.

بروزرسانی و سفارشی کردن فقط برای یک مستاجر سخت است



منابع :

<http://msdn.microsoft.com/en-us/library/aa479086.aspx>

<http://www.codeproject.com/Articles/51334/Multi-Tenants-Database-Architecture>

## نظرات خوانندگان

نویسنده: XPlan

تاریخ: ۱۵:۲ ۱۳۹۲/۰۹/۱۲

با تشکر از شما  
در صورت امکان برای پیاده سازی با روش 2 و 3 یک مثال ساده بیان کنید.

نویسنده: محمد پهلوان

تاریخ: ۱۷:۵۷ ۱۳۹۲/۰۹/۱۲

در لینک MSDN که در قسمت منابع آمده روش 2 و 3 به صورت اجمالی و دستورات SQL نحوه کار با schema ذکر شده است.

ورود سیستم‌های ORM مانند EF تحولی عظیم در در مباحث کار و تغییرات بر روی داده‌ها یا Data Manipulation بود. به طور خلاصه اصلی‌ترین هدف یک ORM، ایجاد فرامین شیء گرا به جای فرامین رابطه‌ای است؛ ولی در این بین نکات دیگری هم مد نظر گرفته شده است که یکی از آن‌ها پشتیبانی از چندین دیتابیس هست تا توسعه گران از یک سیستم واحد جهت اتصال به همه‌ی دیتابیسی‌ها استفاده کنند و نیازی به دانش اضافه و سیستم جداگانه‌ای برای هر دیتابیس نباشد؛ مانند ADO که در دات نت به چندین دسته تقسیم شده بود و هم اینکه در صورتی که تمایلی به تغییر دیتابیس در آینده داشتید، کدها برای توسعه باز باشند و نیازی به تغییر سیستم نباشد.

ولی اگر کمی بیشتر به دنیای واقعی وارد شویم گاهی اوقات نیاز است که تنها بر روی یک دیتابیس فعالیت کنیم و یک دیتابیس نیاز است تا حد ممکن بهینه طراحی شود تا کارآیی بانک در حال حاضر و به خصوص در آینده تا حدی تضمین شود. من همیشه در مورد EF یک نظری داشتم و آن اینست که با اینکه یک ORM، یک هدف مهم را در نظر دارد و آن اینست که تا حد ممکن استانداردهایی را که بین تمامی دیتابیسی‌ها مشترک است، رعایت کند، ولی باز قابل قبول است اگر بگوییم که کاربران EF انتظار داشته باشند تا اطلاعات بیشتری در مورد sql server در آن نهفته باشد. از یک سو هر دو محصول میکروسافت هستند و از سوی دیگر مطمئناً توسعه گران محصولات دات نت بیش از هر چیزی به sql server نگاه ویژه‌تری دارند. پس میکروسافت در کنار حفظ آن ویژگی‌های مشترک، باید به حفظ استانداردهای جدایی برای sql server هم باشد.

تعدادی از برنامه نویسان در هنگام ایجاد Domain Model کم لطفی‌های زیادی را می‌کنند که یکی از آن‌ها عدم کنترل نوع داده‌های خود است. مثلاً برای رشته‌ها هیچ محدودیتی را در نظر نمی‌گیرند. شاید در سمت کلاینت اینکار را انجام می‌دهند؛ ولی نکته‌ی مهم در طرف دیتابیس است که چگونه تعریف می‌شود. در این حالت nvarchar(MAX) در نظر گرفته میشود که به معنی اشاره به منطقه دوگیگابایتی از اطلاعات است. در نکات بعدی، قصد داریم این مرحله را یک گام به جلوتر پیش ببریم و آن هم ایجاد نوع داده‌های بهینه‌تر در Sql Server است.

نکته مهم: بدیهی است که تغییرات زیر، ORM شما را تنها به sql server مقید می‌کند که بعدها در صورت تغییر دیتابیس نیاز به حذف موارد زیر را خواهید داشت؛ در غیر اینصورت به مشکل عدم ایجاد دیتابیس برخورد خواهید کرد.

**اولین مورد مهم بحث تاریخ و زمان است؛** وقتی ما یک نوع داده را تنها در DateTime در نظر بگیریم، در Sql Server هم همین نوع داده وجود دارد و انتخاب میشود. ولی اگر شما واقعاً نیازی به این نوع داده نداشته باشید چطور؟ در حال حاضر من بر روی یک برنامه‌ی کارخانه کار میکنم که بخش کارمندان و گارگران آن سه داده زمانی زیر را شامل می‌شود:

```
public DateTime BirthDate { get; set; }
public DateTime HireDate { get; set; }
public DateTime? LeaveDate { get; set; }
```

حال به جدول زیر نگاه کنید که هر نوع داده چه مقدار فضا را به خود اختصاص می‌دهد:

4 بایت	<a href="#">SmallDateTime</a>
8 بایت	<a href="#">DateTime</a>
6 تا 8 بایت	<a href="#">DateTime2</a>
8 تا 10 بایت	<a href="#">DateTimeOffset</a>
3 بایت	<a href="#">Date</a>

4 بایت	<a href="#">SmallDateTime</a>
3 تا 5 بایت	<a href="#">Time</a>

از این جدول چه می‌فهمید؟ با یک نگاه می‌توان فهمید که ساختار بالای من باید 24 بایت گرفته باشد؛ برای ساختاری که هم تاریخ و هم زمان (ساعت) را پشتیبانی می‌کند. ولی با نگاه دقیق‌تر به نام پراپرتی‌ها این نکته روشن می‌شود که ما یک گپ (Gap فضای بیهوده) داریم چون زمان تولد، استخدام و ترک سازمان اصلاً نیازی به ساعت ندارند و همان تاریخ کافی است. یعنی نوع Date با حجم کلی 9 بایت؛ که در نتیجه 15 بایت صرفه جویی در یک رکورد صورت خواهد گرفت.

پس کد بالا را به شکل زیر تغییر می‌دهم:

```
[Column(TypeName = "date")]
public DateTime BirthDate { get; set; }

[Column(TypeName = "date")]
public DateTime HireDate { get; set; }

[Column(TypeName = "date")]
public DateTime? LeaveDate { get; set; }
```

خصوصیت [Column](#) از نسخه 4.5 دات نت فریم ورک اضافه شده و در فضای نام `System.ComponentModel.DataAnnotations.Schema` قرار گرفته است.

نوع‌هایی که در بالا با سایز متغیر هستند، به نسبت دقتی که برای آن تعیین می‌کنید، سایز می‌گیرند. مثل `time(0)` که 3 بایت از حافظه را می‌گیرد. در صورتی که `time` معرفی کنید، به جای اینکه از شیء `DateTime` استفاده کنید، از شیء `Timespan` استفاده کنید، تا در پشت صحنه از نوع داده `time` استفاده کند. در این حالت حداکثر حافظه یعنی 5 بایت را برخواهد داشت و بهترین حالت ممکن این هست که نیاز خود را بسنجید و خودتان دقت آن را مشخص کنید. دو شکل زیر نحوه‌ی تعریف نوع زمان را مشخص می‌کنند. یکی حالت پیش فرض و دیگری انتخاب دقت:

```
public class Testtypes
{
    public TimeSpan CloseTime { get; set; }
    public TimeSpan CloseTime2 { get; set; }
}

public class TestConfig : EntityTypeConfiguration<Testtypes>
{
    public TestConfig()
    {
        this.Property(x => x.CloseTime2).HasPrecision(3);
    }
}
```

در تکه کد بالا همه از نوع `time` تعریف می‌شوند ولی در خصوصیت شماره یک نهایت استفاده از نوع تایم یعنی `time(7)` مشخص می‌شود. ولی در خصوصیت بعدی چون در کانفیگ دقت آن را مشخص کرده‌ایم از نوع `time(3)` تعریف می‌شود.

#### مورد دوم در مورد داده‌های اعشاری است:

بسیاری از برنامه نویسان تا آنجا که دیده‌ام از نوع `float` و `single` و `double` برای اعداد اعشاری استفاده می‌کنند ولی باید دید که در آن سمت دیتابیس، برای این نوع داده‌ها چه اتفاقی می‌افتد. نوع `float` در دات نت، با نوع `single` برابری می‌کند؛ هر دو یک نام جدا دارند، ولی در واقع یکی هستند. عموماً برنامه نویسان به طور کلی بیشتر از همان `single` استفاده می‌کنند و برای انتساب برای این دو نوع هم حتماً باید حرف `f` را بعد از عدد نوشت:

```
float flExample=23.2f;
```

باید توجه کنید که اگر مثلاً `float` انتخاب کردید، تصور نکنید که همان `float` در دیتابیس خواهد بود. این دو متفاوت هستند تبدیلات به شکل زیر رخ می‌دهد:

```
//real
```

```
public float FloatData { get; set; }
//real
public Single SingleData { get; set; }
//float
public double DoubleData { get; set; }
```

همه نوع‌های بالا اعداد اعشاری هستند که به صورت تقریبی و به صورت نماد اعشاری ذخیره می‌گردند و برای به دست آوردن مقدار ذخیره شده، هیچ تضمینی نیست همان عددی که وارد شده است بازگردانده شود. اگر تا به حال در برنامه هایتان به چنین مشکلی برخوردید دلیلش اعداد اعشاری کوچک بوده است. ولی با بزرگتر شدن عدد، این تفاوت به خوبی دیده می‌شود. حالا اگر بخواهیم اعداد اعشاری را به طور دقیق ذخیره کنیم، مجبور به استفاده از نوع decimal هستیم. در دات نت آنچنان محدودیتی بر سر استفاده‌ی از آن نداریم. ولی در سمت سرور داده‌ها بهتر هست برای آن تدابیری اندیشیده شود. هر عدد دسیمال از دقت و مقیاس تشکیل می‌شود. دقت آن تعداد ارقامی است که در عدد وجود دارد و مقیاس آن تعداد ارقام اعشاری است. به عنوان مثال عدد زیر دقتش 7 و مقیاسش 3 است:

```
4235.254
```

در صورتی که عدد اعشاری را به دسیمال نسبت دهیم باید حرف m را بعد از عدد وارد کنیم:

```
decimal d1=4545.112m;
```

برای اعداد صحیح نیازی نیست. برای تعیین نوع دسیمال از fluent api استفاده می‌کنیم:

```
modelBuilder.Entity<Class>().Property(object => object.property).HasPrecision(7, 3);
```

کد زیر برای خصوصیت شماره یک، دقت 18 و مقیاس 2 را در نظر می‌گیرد و دومین خصوصیت طبق آنچه که برایش تعریف کرده ایم دقت 7 و مقیاس 3 است:

```
public class Testtypes
{
    public Decimal Decimal1 { get; set; }
    public Decimal Decimal2 { get; set; }
}

public class TestConfig : EntityTypeConfiguration<Testtypes>
{
    public TestConfig()
    {
        this.Property(x => x.Decimal2).HasPrecision(7, 3);
    }
}
```

**مورد سوم مبحث رشته هاست :**

کدهای زیر را مطالعه فرمایید:

```
[StringLength(25)]
public string FirstName { get; set; }

[StringLength(30)]
[Column(TypeName = "varchar")]
public string EnProductTitle { get; set; }

public string ArticleContent { get; set; }
[Column(TypeName = "varchar(max)")]
public string ArticleContentEn { get; set; }
```

اولین رشته بالا (نام) را به محدوده‌ای از کاراکترها محدود کرده‌ایم. به طور پیش فرض تمامی رشته‌ها به صورت nvarchar در نظر گرفته می‌شوند. بدین ترتیب در رشته نام کوچک (nvarchar(25) در نظر گرفته خواهد شد. حال اگر بخواهیم فقط حروف انگلیسی

پشتیبانی شوند، مثلاً نام فنی کالا را بخواهید وارد کنید، بهتر هست که نوع آن به طرز صحیحی تعریف شود که در کد بالا با استفاده از خصوصیت Column نوع varchar را معرفی می‌کنم. بدین ترتیب تعریف نهایی نوع به شکل varchar(30) خواهد بود. استفاده از fluentApi هم در این رابطه به شکل زیر است:

```
this.Property(e => e.EnProductTitle).HasColumnType("VARCHAR").HasMaxLength(30);
```

برای مواردی که محدوده‌ای تعریف نشود (nvarchar(MAX) در نظر گرفته می‌شود مانند پراپرتی ArticleContent بالا). ولی اگر قصد دارید فقط حروف اسکی پشتیبانی گردند، مثلاً متن انگلیسی مقاله را نیز نگه می‌دارید بهتر هست که نوع آن به‌هیته‌ترین حالت در نظر گرفته شود که برای پراپرتی ArticleContentEn نوع varchar(MAX) تعریف کرده‌ایم. همانطور که گفتیم پیش فرض رشته‌ها nvarchar است، در صورتی که دوست دارید این پیش فرض را تغییر دهید روش زیر را دنبال کنید:

```
modelBuilder.Properties<string>().Configure(c => c.HasColumnType("varchar"));
```

//===== یا

```
modelBuilder.Properties<string>().Configure(c => c.IsUnicode(false));
```

جهت تکمیل بحث نیز هر کدام از متغیرهای عددی در سی شارپ معادل نوع‌های زیر در Sql Server هستند:

```
//tinyInt
public byte Age { get; set; }

//smallInt
public Int16 OldInt { get; set; }

//int
public int Int32 { get; set; }

//Bigint
public Int64 HighNumbers { get; set; }
```

## نظرات خوانندگان

نویسنده: مرتضی ریسی  
تاریخ: ۲۰:۴۴ ۱۳۹۴/۰۵/۰۴

سلام. ممنون.  
میشه بفرمائید برای مقادیر مالی به ریال و تومان بهترین نوع داده ای چیست؟  
من اینچنین استفاده میکنم:

```
public virtual decimal CostPrice { set; get; }
```

و در کانفیگ:

```
this.Property(x => x.CostPrice)  
    .HasColumnType("money")  
    .IsRequired();
```

همین نوع و همین اندازه تنظیم کافیه؟ آیا تنظیم بیشتری نیاز دارد؟

نویسنده: علی یگانه مقدم  
تاریخ: ۲۲:۲ ۱۳۹۴/۰۵/۰۴

توصیه می‌کنم برای واحد پولی ایران از این جنس استفاده نکنید. از همان واحدهای عددی دقیق استفاده کنید.  
اگر واحد مالی ما مانند کشورهای خارجی به صورت اعشار بیان میشد بله بهینه بود ولی در حال حاضر خیر.  
من خودم تا به الان از Int استفاده کردم و می‌توان برای واحدهای بزرگتر BigInt را مورد استفاده قرار داد. هر چند int تا میلیارد را به خوبی پشتیبانی می‌کند