



# به نام خالق علم

آموزر مقدماتی  
sql server

رانسگاه شریعت بررسی

نمایخانه رکورضوهای از

بهار ۱۴۰۱

# جلسہ راول

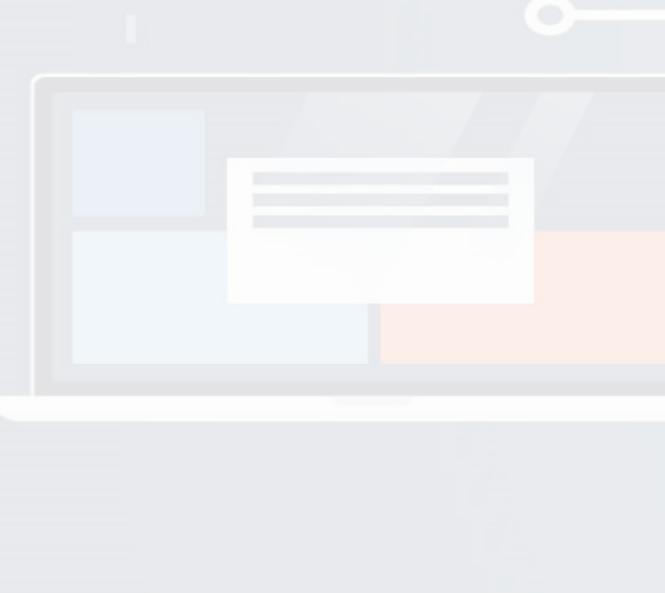
- شروع: مقدمات
- پیغام: رسم ER میں

# کاربردهای sql server

- علاقه مندان به بک اند وب
- پشتیبان نرم افزار
- مدیر بانک های اطلاعاتی
- متخصص داده (data scientist)
- متخصص هوش تجاری (BI)

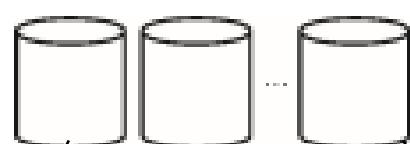
\*اين دوره پيشنياز خاصی ندارد اما باید به مفهوم برنامه نويسی آشنا باشيد\*

- داده: هر حقیقتی که به یک موجودیت نسبت دهیم را داده می‌گویند.
- اگر این داده‌ها با نظم مشخصی در کنارهم قرار بگیرند پایگاه داده را تشکیل می‌دهند.
- اگر داده‌ها در یک صفحه با یک نظم خاص قرار بگیرد نام flat file database دارد.
- اغلب در زمینه ثبت و نگهداری اطلاعات تراکنشی (Transaction) از پایگاه‌های داده هموار بهره می‌برند.
- پسوند CSV. معمولاً برای این داده‌ها استفاده می‌شود.



```
*Untitled - Notepad
File Edit Format View Help
"OrderID", "CustomerID", "OrderDate"
"01", "001", "06/06/2021"
"02", "369", "06/06/2021"
"03", "151", "06/06/2021"
"04", "014", "06/06/2021"
"05", "061", "06/06/2021"
"06", "220", "06/06/2021"
```

- به طور کلی ایجاد پایگاه اطلاعاتی یک محصول نرم افزاری شامل مراحل زیر است(در معماری ANSI):

دیدهای کاربران مختلف (view)	کاربر 1 ... کاربر 2 ... کاربر n	تصویر خارجی
کل بانک بدون توجه به مدل خاصی	ER مدل 	تصویر ادراکی عام (طراحی ادراکی)
کل بانک در قالب مدل انتخابی	مدل رابطه‌ای 	تصویر ادراکی خاص (طراحی منطقی)
کل بانک روی رسانه	SQL 	تصویر فیزیکی

1. پیاده سازی جدول DDL      2. پیاده سازی پرس و جو DML

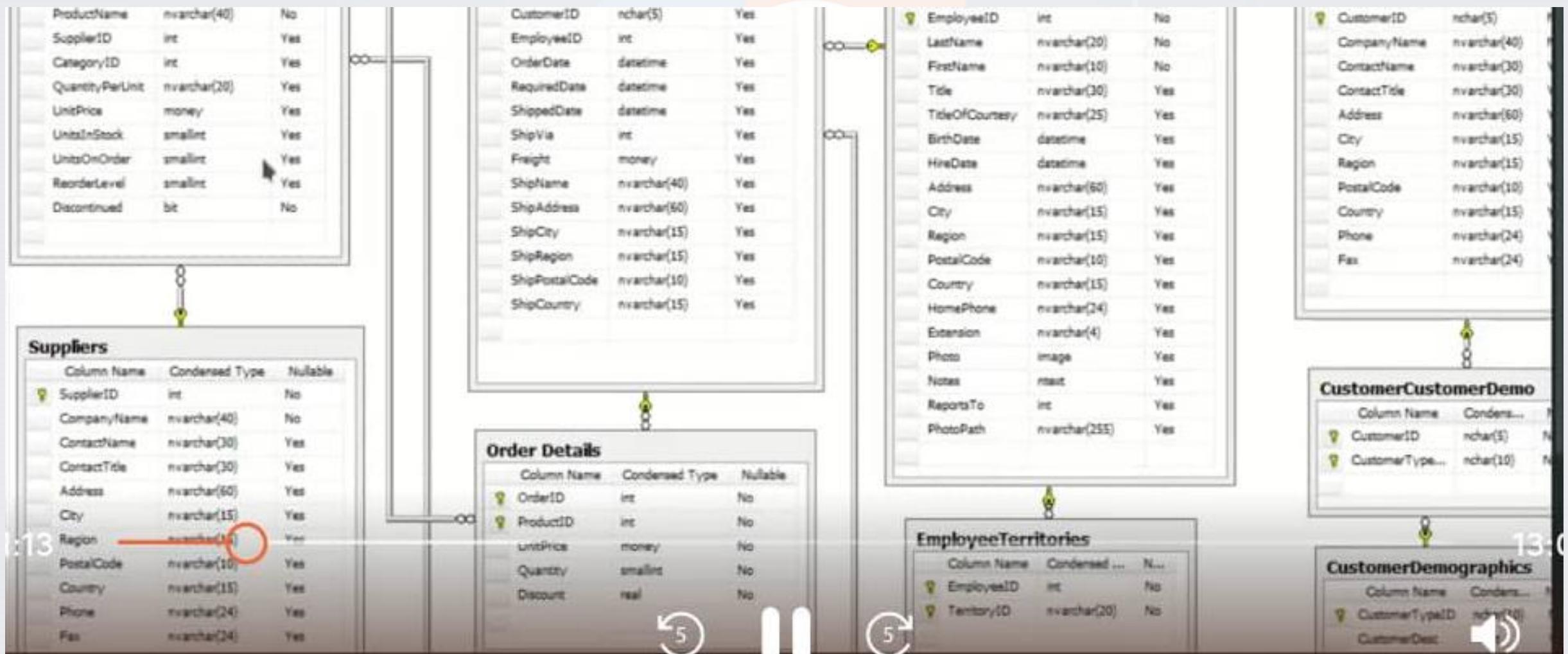
- انواع پایگاه داده ها:

1.1 = ساختار جدولی که با فیلد های مربوط به آن لیست می شود و ساختار بعد از طراحی قابل تغییر نیست.

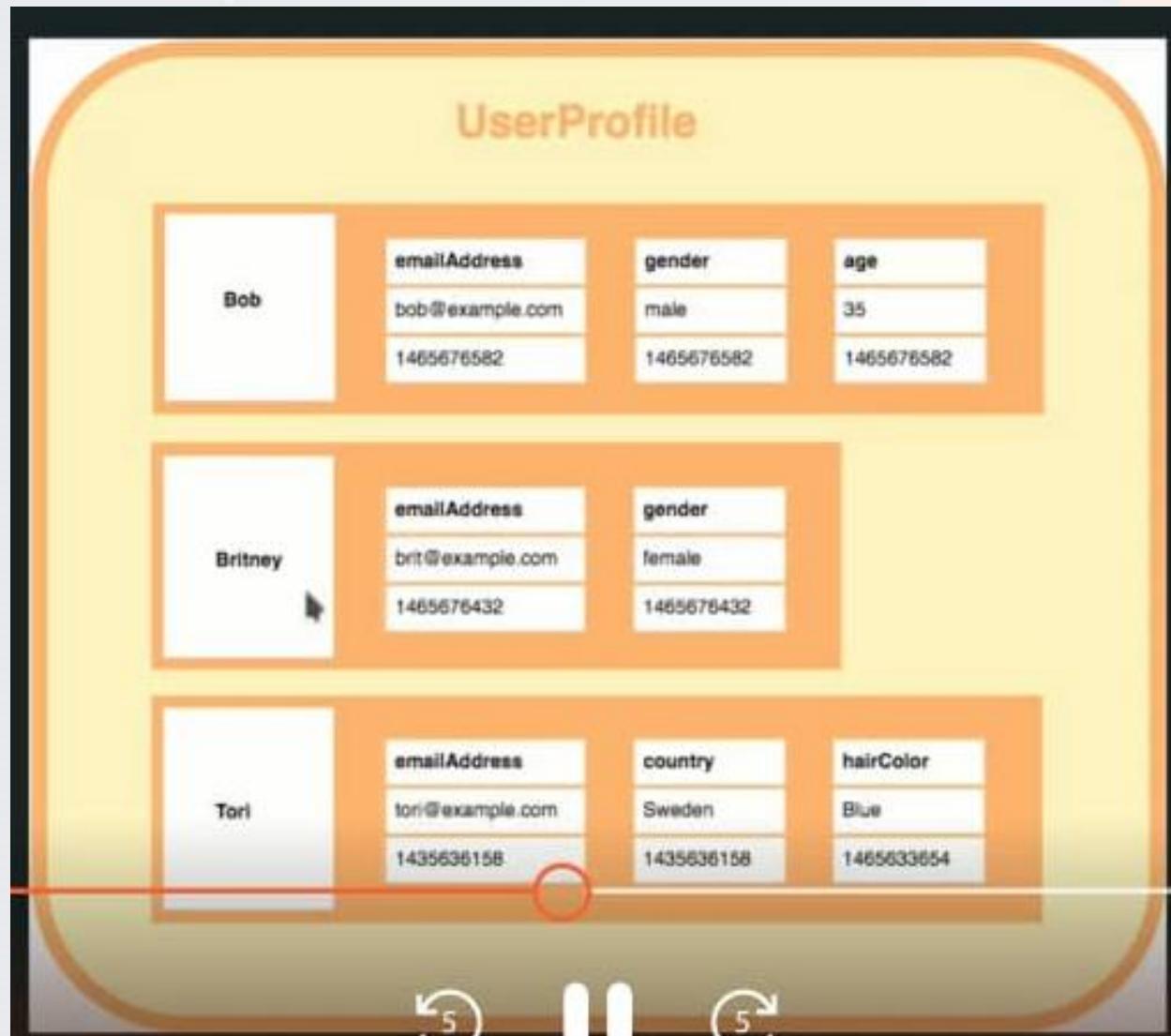
Sql server – mysql – postreqsql – oracle

1.1 = ساختار منعطفی دارد و نباید همه ی جزئیات طراحی از قبل مشخص شود.

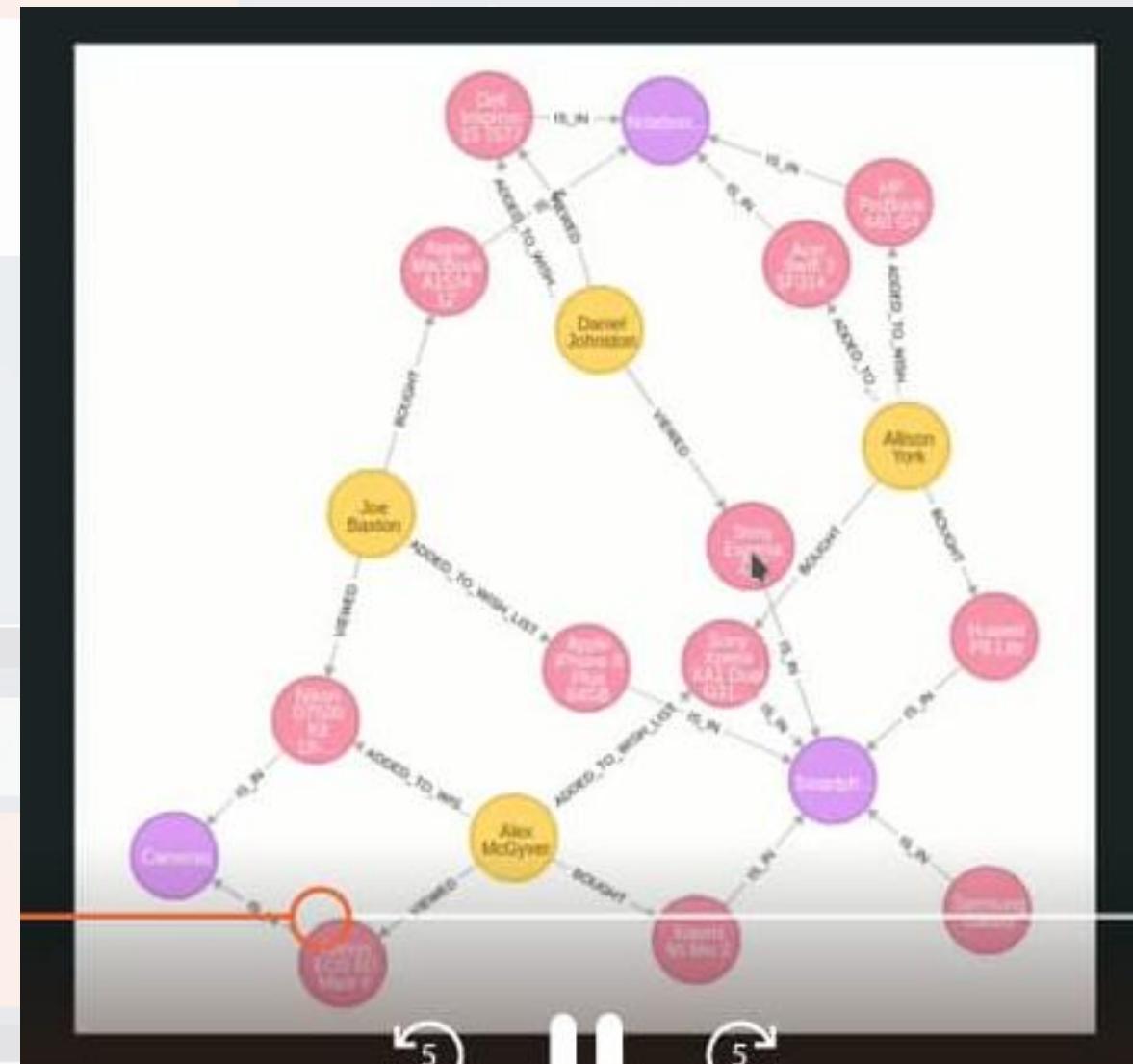
# sql



## Nosql : column



## Nosql : graph



## NoSql : key-value

Database

Collection

Collection

Collection

Document#1  
Key: Value

Document#2  
(Key: Value,  
Key: Value )

Document#3  
[Key: Value]

Document#4  
Key: {Key:  
Value, Key:  
Value}

# Nosql : document

## Document 1

```
{  
  "id": "1",  
  "name": "John Smith",  
  "isActive": true,  
  "dob": "1964-30-08"  
}
```

## Document 2

```
{  
  "id": "2",  
  "fullName": "Sarah Jones",  
  "isActive": false,  
  "dob": "2002-02-18"  
}
```

## Document 3

```
{  
  "id": "3",  
  "full Name":  
  {  
    "first": "Adam",  
    "last": "Stark"  
  },  
  "isActive": true,  
  "dob": "2015-04-19"  
}
```

- انواع دیتابیس Sql :



- Mysql

- Sql server

- Postreqsql

- oracle

## • انواع دیتابیس : Nosql



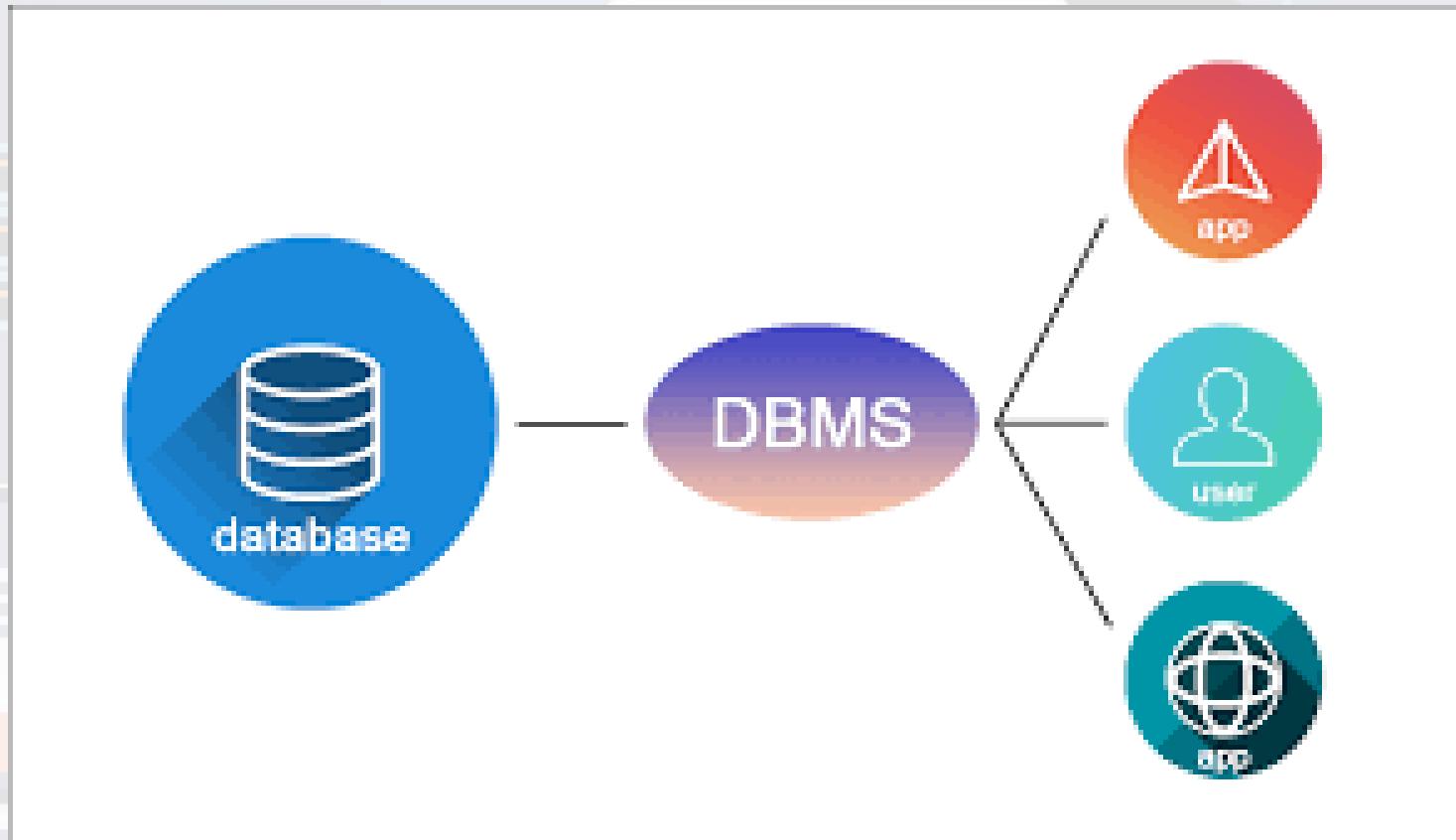
# Sql بهتر است یا Nosql ؟

- برای پایگاه هایی که اطلاعات مرتبط به دارند مثل یک بانک یا دانشگاه استفاده از پایگاه داده های رابطه ای مناسب تر است.

- اما اگر اطلاعات ما حجم وسیعی دارد و ارتباط معناداری نیز بین آن ها نیست Nosql انتخاب مناسب تری می باشد مثل شبکه های اجتماعی یا حوزه های مربوط به Big data

- پایگاه داده رابطه ای : مجموعه داده های مرتبط به هم، با حداقل افزونگی، به صورت مجتمع، یکپارچه و پایا برای استفاده‌ی چندین کاربر به صورت همزمان است.

# چیست؟ DBMS



# مزایای DBMS

- .1 مدیریت حافظه در حجم های بالای اطلاعات
- .2 محدود کردن یا افزایش میزان دسترسی ها به اطلاعات
- .3 دارا بودن یک زبان عام برای کدنویسی به جهت سهولت در مدیریت داده ها
- .4 افزایش سرعت مدیریت داده ها
- .5 این زبان در پایگاه داده های رابطه ای SQL نام دارد و هر پلتفرم آن را برای شرکت خود شخصی سازی کرده است و تفاوت های جزئی با هم دارند.

- Sql = structured query language

- پشتیانی شده توسط همهٔ پایگاه‌های رابطه‌ای
- query = تمام در خواست‌ها روی DBMS توسط این دستورات اجرا می‌شود.
- SQL یک زبان توصیفی است و مانند زبان‌های سطح بالا رویه‌ای نیست. به این معنا که نحوهٔ فرآخوانی را باید برای آن توضیح داد اما مثلاً در پایتون باید از مبدأ تا چگونه خواندن دیتا و تغییر در آن و مقصد دوباره ذخیره کردن داده را توضیح داد.
- T-SQL : زبان برنامه نویسی سمت بانک اطلاعاتی برای SQL SERVER است.

# ER = Entity Relationship Diagram

- هر نمودار er می تواند به راحتی به یک دیتابیس رابطه ای (sql) تبدیل شود.
- نمودار er = نوعی نمودار فلوچارت که به درک ارتباط بین موجودیت ها یا اشیا یا مفاهیم مختلف در یک سیستم کمک می کند.
- مدل er = یک مدل داده ای سطح بالا و یک رویکرد گرافیکی برای طراحی پایگاه داده که ارتباط بین اشیا را در سیستم مشخص می کند.
- هم در زمینه ای پایگاه داده و هم در زمینه ای مهندسی نرم افزار استفاده می شود.

# مزایای ER:

- تعریف ساختارهای ساده، لازم و کافی و غیر وابسته به پیاده سازی برای طراحی بانک اطلاعات که ویژگی های مدل ادراکی عام (model Conceptual) را به خوبی برآورده می کند.
- ۲-تعریف نمادهای مناسب برای ارائه یک طراحی قابل فهم و ساده

## مثالی از ER سیستم دانشگاه

صفت ها :

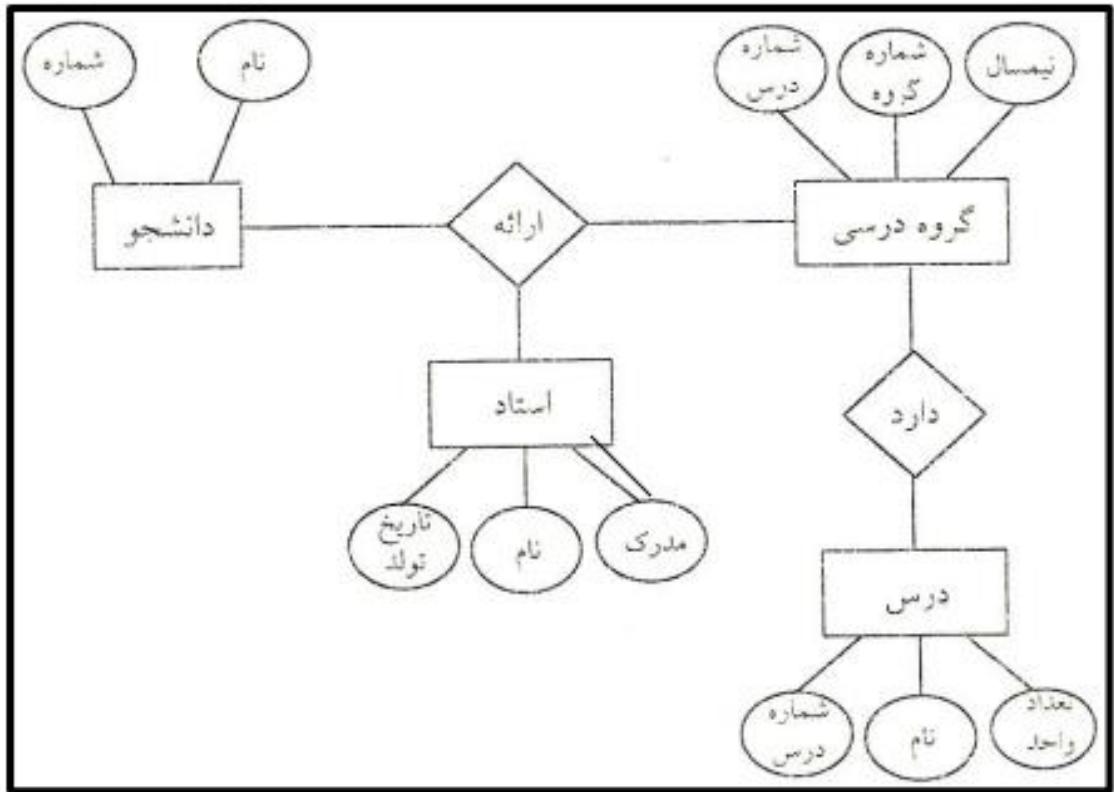
- نام دانشجو
- شماره دانشجویی
- نام استاد
- مدرک
- تاریخ تولد
- نیمسال
- شماره گروه
- شماره درس
- نام درس
- تعداد واحد
- شماره درس

موجودیتها :

- دانشجو
- استاد
- گروه درسی
- درس

ارتباط :

- ارائه
- دارد



# اجزای ER

1. موجودیت

موجودیت قوی

موجودیت ضعیف

اختیاری

اجباری

ضعیف

قوی

2. رابطه های بین موجودیت ها

3. صفات

- موجودیت: هر چیزی که بتواند با چند مشخصه توصیف شود موجودیت نام دارد مثل دانشجو یا ثبت‌نام.
- موجودیت قوی: کلید اصلی دارد و به هیچ موجودیت دیگری وابسته نیست.
- موجودیت ضعیف: کلید اصلی ندارد و به یک موجودیت قوی وابسته است و با حذف موجودیت قوی حذف می‌شود.

- رابطه: ارتباطی که بین دو یا چند موجودیت وجود دارد.
- رابطه‌ی قوی: حاوی کلید اصلی والدین. رابطه‌ای که وجود موجودیت به موجودیت دیگری وابسته نباشد.
- رابطه‌ی ضعیف: به یک رابطه‌ی قوی وابسته است. کلید اصلی از موجودیت دیگری گرفته می‌شود.
- رابطه‌ی انعکاسی: یک موجودیت با خودش ارتباط داشته باشد مثل رابطه‌ی پیش‌نیاز که هر درس می‌تواند پیش‌نیاز درس دیگری باشد.

# انواع صفت:

- ساده - مرکب:

ساده: از یک جز تشکیل شود مثل نام - فامیل  
مرکب: از چند جز تشکیل شود مثل حقوق که شامل اضافه کاری، عیدی، تشویقی و ... است.



- تک مقداری - چند مقداری:

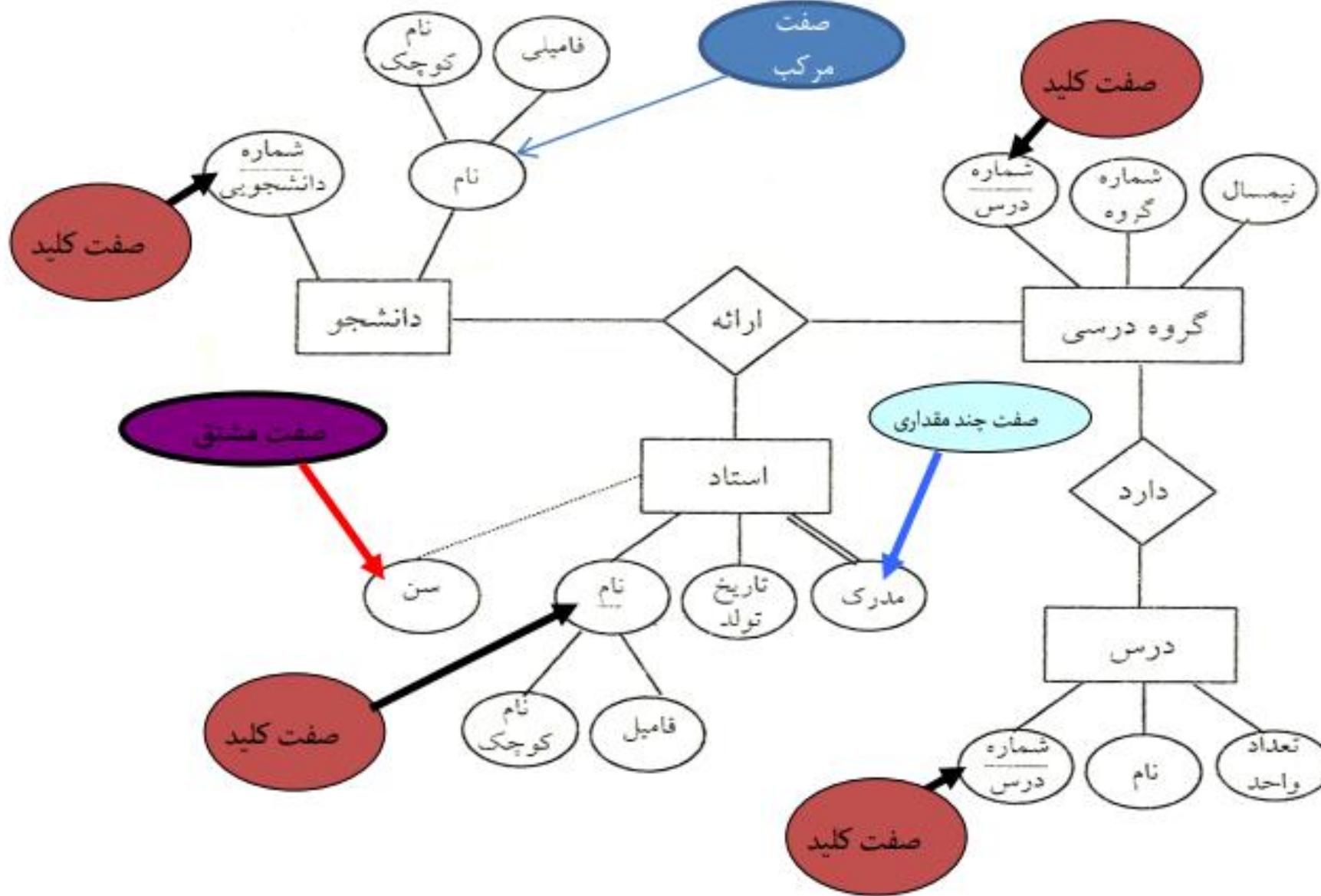
تک مقداری: یک جز است و یک مقدار دارد مثلاً افراد فقط یک فامیلی دارند یا تنها یک شماره ملی دارند.  
چند مقداری: یک جز است اما چند مقدار دارد مثل فردی که چند شماره حساب دارد یا تلفن که می‌تواند متعلق به محل کار یا منزل یا همراه باشد.

- صفت مشتق: 

مقداری که توسط کامپیوتر محاسبه می شود مثل معدل زیرا نمرات دانشجویان ثبت می شود اما معدل باید توسط خو کامپیوتر محاسبه شود.

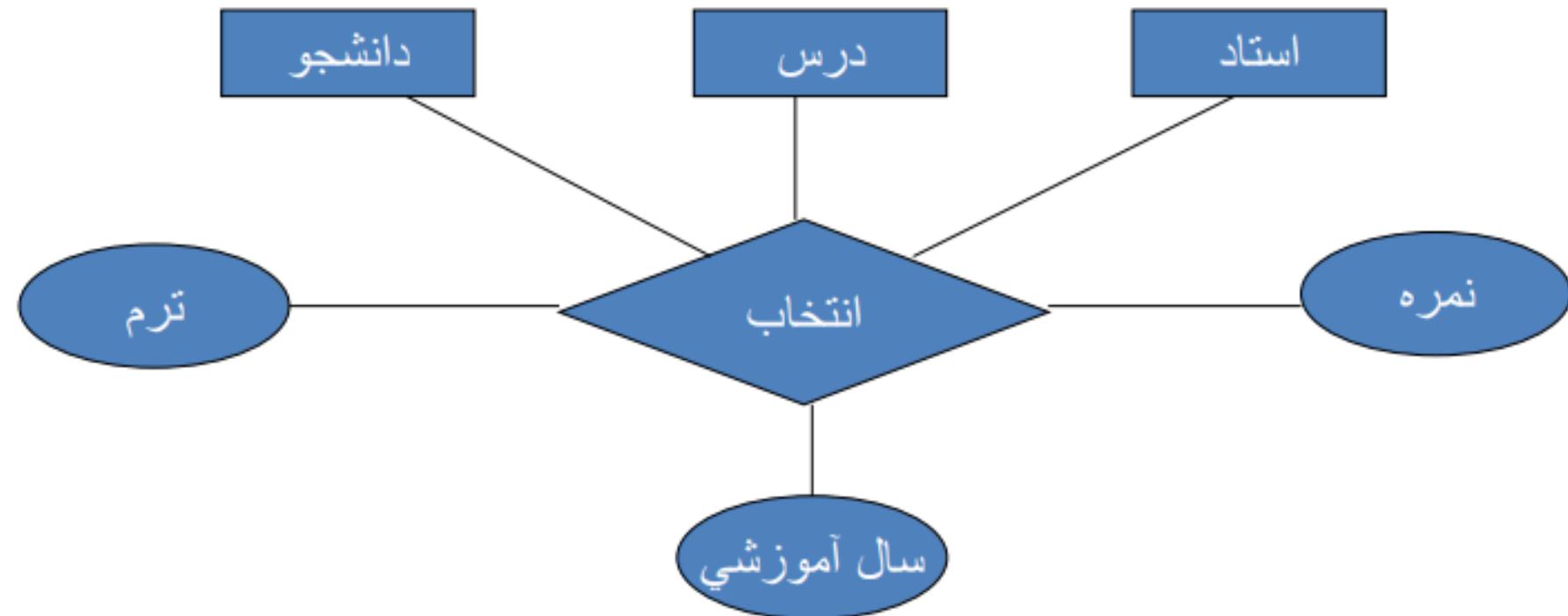
- صفت شناسه یا کلیدی:  
صفتی که برای موجودیت منحصر به فرد است مثل شماره ملی یا دانشجویی
- صفت کلیدی جزئی یا غیرشناسه: صفت اصلی در موجودیت ضعیف که با ترکیب صفت اصلی موجودیت قوی کلید اصلی خود را می سازد.

- \* برای ایجاد ارتباط منطقی باید صفات شناسه را کنارهم گذاشت.



# صفت در رابطه

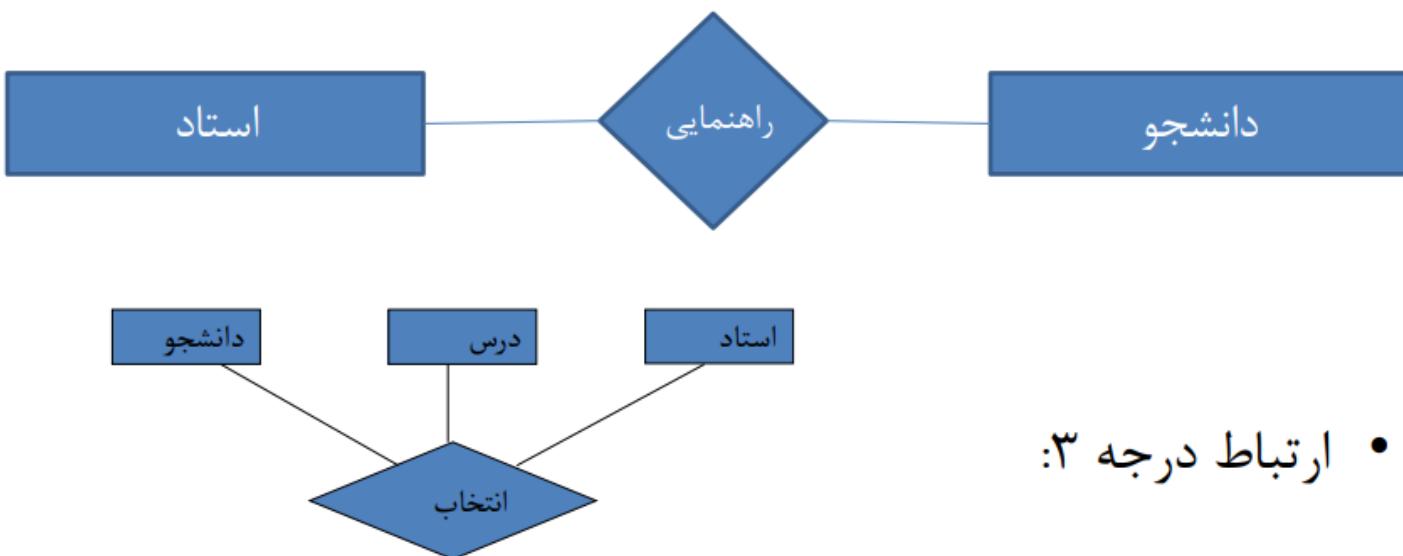
- رابطه ها نیز می توانند صفت بگیرند:



# درجه ارتباط

- تعداد موجودیت هایی که در یک رابطه شرکت دارند.
- درجه ارتباط در مدل ER عددی صحیح و کوچکتر از ۵ است و معمولاً تا درجه ۳ رسم می شود و درجه ۴ کمیاب و درجه ۵ به بالا غیرقابل رسم است.

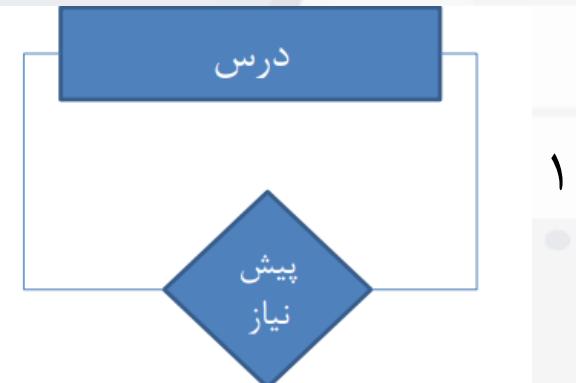
- ارتباط درجه ۲



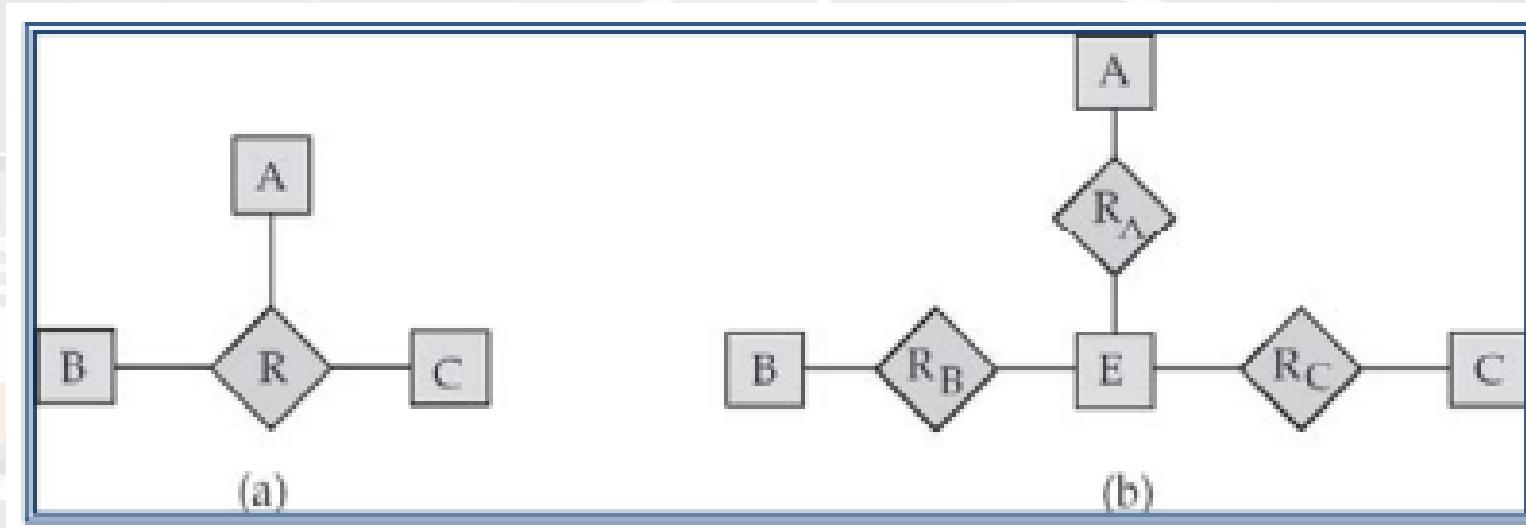
- ارتباط درجه ۳:

ارتباط بین سه موجودیت

- ارتباط درجه ۱

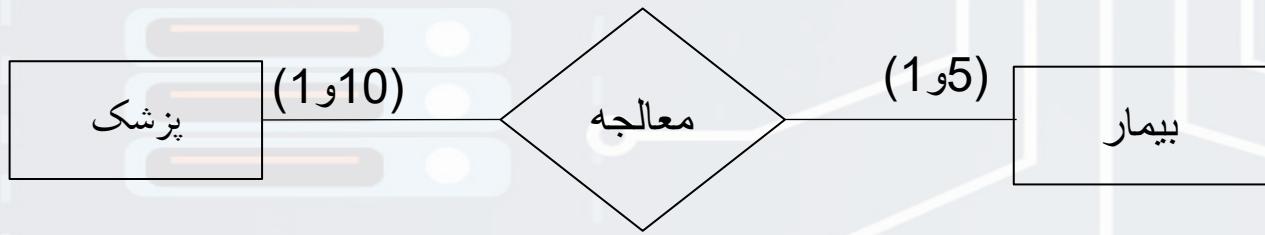


- به دلیل دشواری پیاده سازی ارتباط های بیش از دو موجودیتی، آن هارا با استفاده از یک موجودیت اضافه به ارتباط های باینری(دو موجودیتی) تبدیل می کنیم.



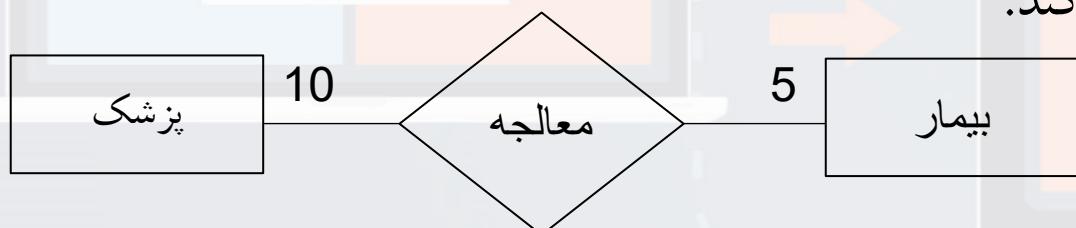
# حد (cardinality)

- کاردینالیتی در یک ارتباط تعداد ماقزیم و مینیم نمونه های یک موجودیت که در یک ارتباط مشارکت می کند را مشخص می کند.



► (حداکثر، حداقل)

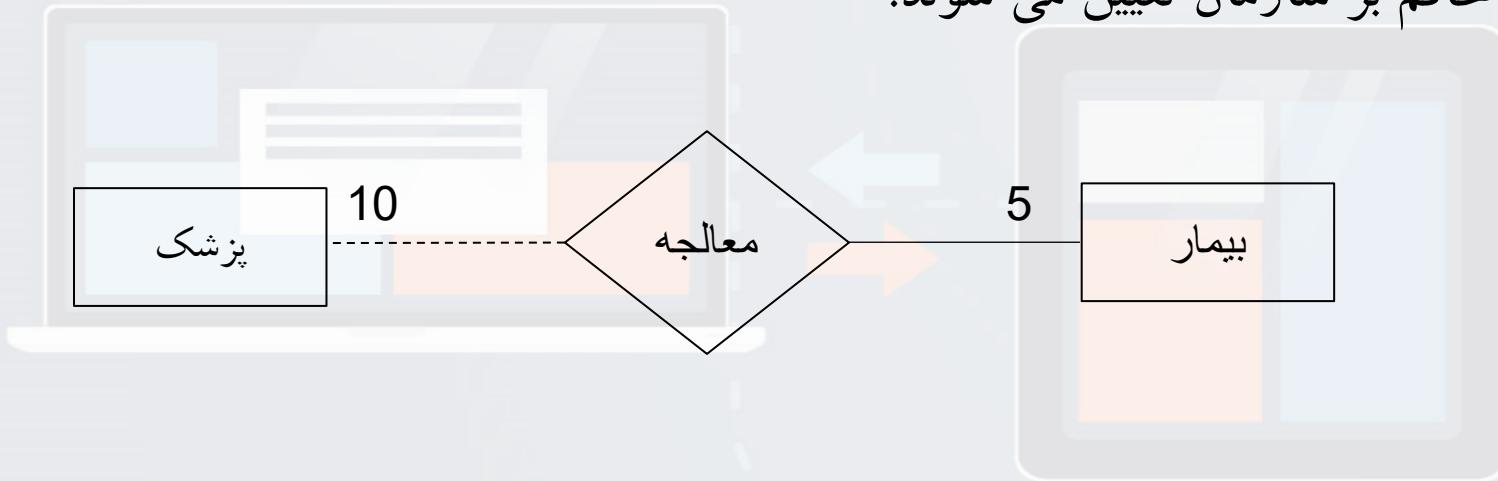
- هر بیمار می تواند حداقل یک و حداکثر ۵ پزشک معالج داشته باشد.
- هر پزشک می تواند حداقل یک و حداکثر ۱۰ بیمار معالجه کند.
- عموماً مقدار حداقل روی روابط نوشته نمی شود.



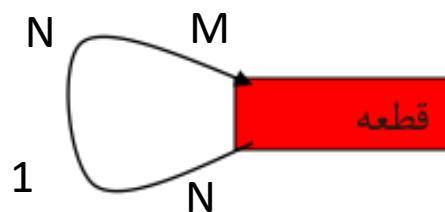
# روابط اختیاری و اجباری:

- فرض می کنیم پزشک های سیستم پایگاه داده می توانند هیچ بیماری برای معالجه نداشته باشند در این حالت حداقل کاردینالیتی مقدار صفر می گیرد و در این صورت ما رابطه ای اختیاری خواهیم داشت اما اگر در سناریوی سیستم شرطی مبنی بر اجبار دارابودن پزشک با حداقل یک بیمار وجود داشته باشد این ارتباط اجباری خواهد بود.

- نکته: کاردینالیتی ارتباط توسط قوانین حاکم بر سازمان تعیین می شوند.



## رابطه انعکاسی (self-recursive)



ممکن است یک موجودیت با خودش ارتباط داشته باشد.  
این بدین معناست که «یک قطعه از قطعه یا قطعات دیگر  
ساخته شده است».

اتصال (connectivity) یک ارتباط حالت‌های زیر را ممکن است داشته باشد:

۱. ارتباط یک به یک (1:1)

۲. ارتباط یک به چند (m:1)

۳. ارتباط چند به چند (m:n)

**۱. ارتباط یک به یک (one to one)**

ارتباط یک به یک (1:1) وقتی است که دقیقاً یک نمونه از موجودیت A مربوط به دقیقاً یک نمونه از موجودیت B شود.

مثال. به هر کارمند در یک شرکت یک دفتر اختصاص داده می‌شود. در هر دفتر فقط یک کارمند وجود دارد.

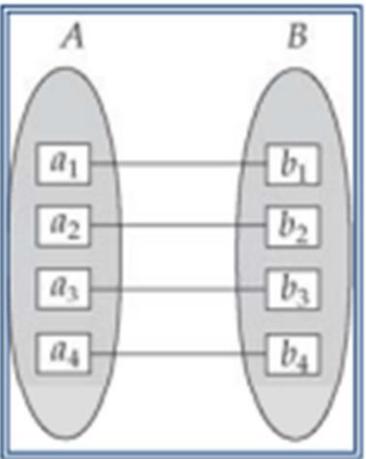
## 2. ارتباط یک به چند (one to many)

ارتباط یک به چند ( $1:n$ ) وقتی است که یک نمونه از موجودیت A به چندین نمونه از موجودیت B مربوط شود اما برای هر نمونه موجود در B تنها یک نمونه از A وجود داشته باشد.  
مثال: در یک رشته تعدادی دانشجو مطالعه می کند. هر دانشجو تنها می تواند در یک رشته درس بخواند.

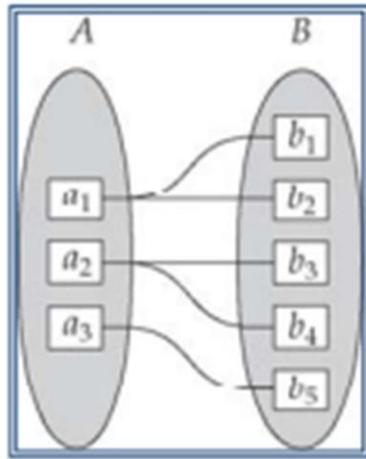
## 3. روابط چند به چند (many to many)

ارتباط چند به چند ( $m:n$ ) وقتی است که یک نمونه موجودیت A به چند نمونه موجودیت B مربوط شود و یک نمونه موجودیت در B به چند نمونه موجودیت در A مرتبط شوند.  
مثال: هر استاد چند درس را ارائه می دهد و هر درس می تواند توسط چند استاد ارائه شود.

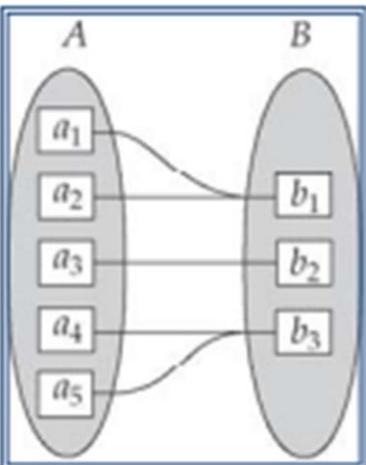
\* تقریبا همیشه وجود یک ارتباط چند به چند منعکس کننده یک موجودیت نهفته است به همین دلیل با مشخص شدن و اضافه کردن این موجودیت پنهان به مدل ارتباط چند به چند از دیاگرام حذف می شوند و موجودیت جدید به دو موجودیت قبلی با ارتباطات یک به چند مرتبط می شود.



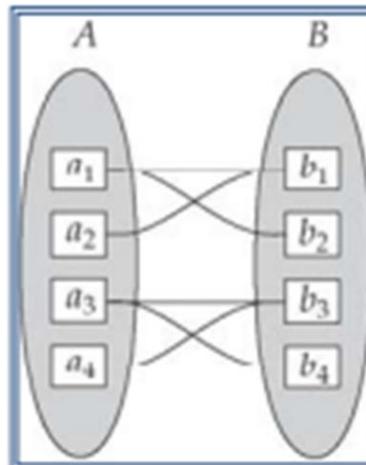
( 1 : 1 )



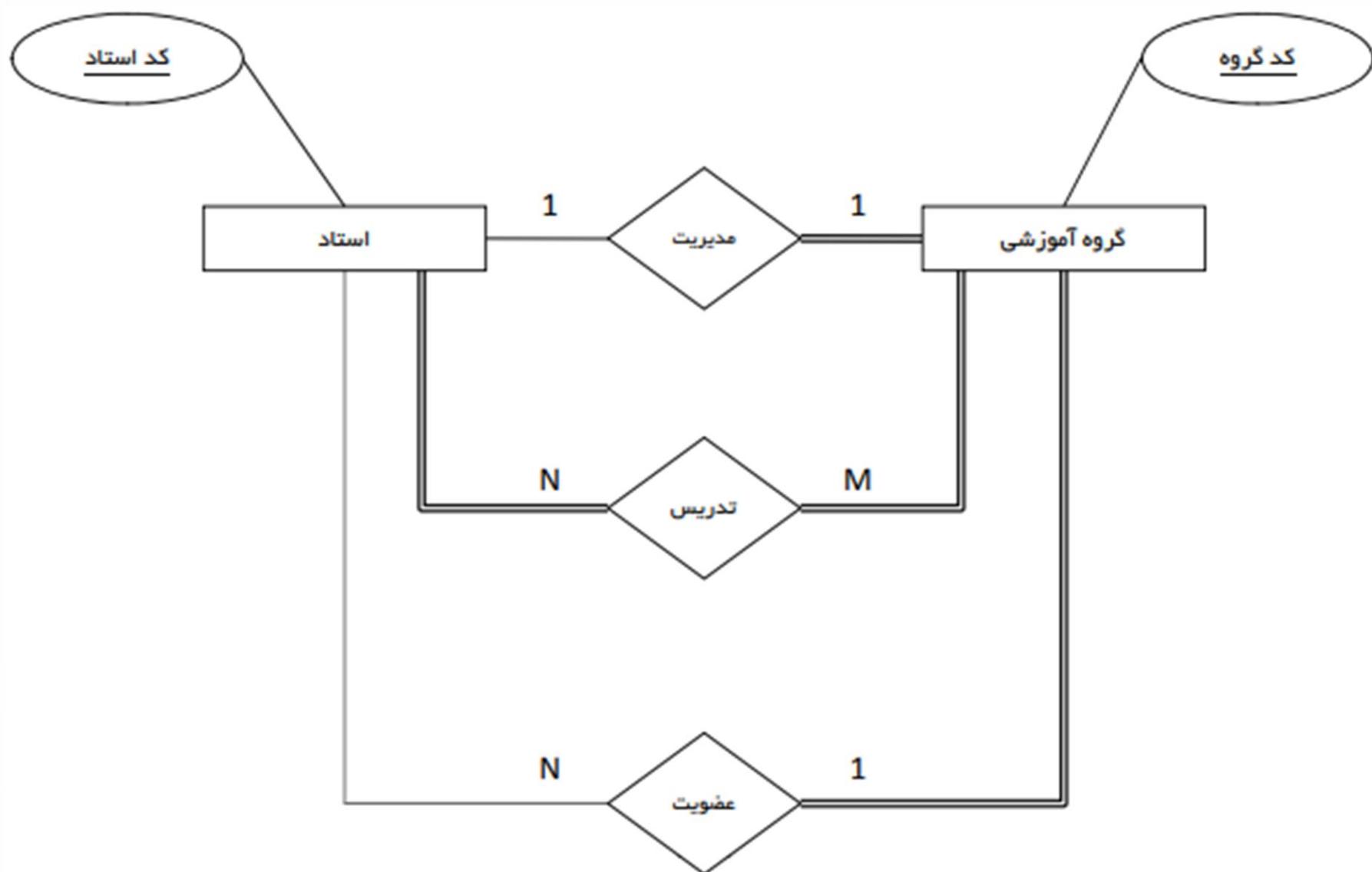
( 1 : N )

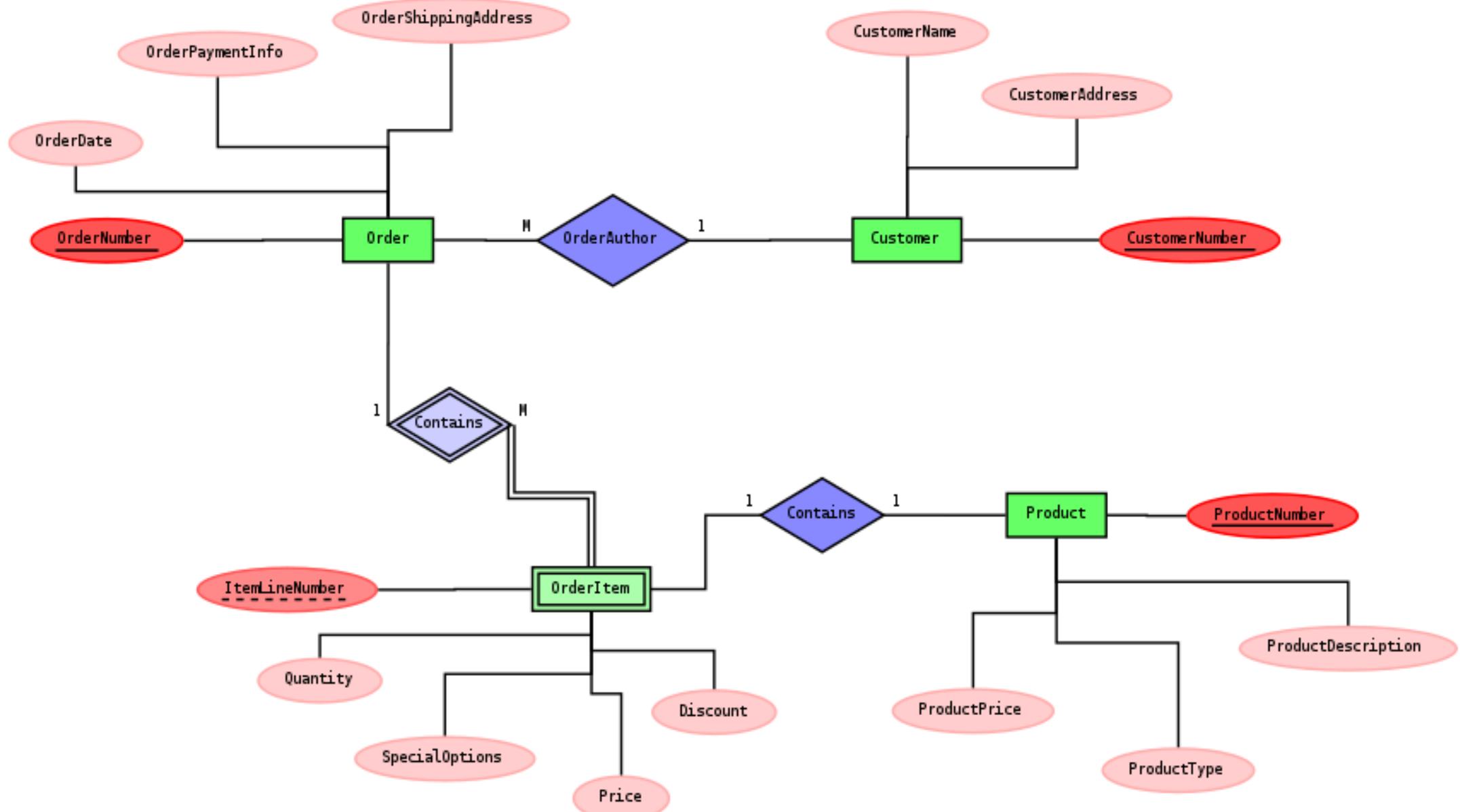


( N : 1 )



( N : M )





یک پایگاه داده شامل رکوردهای سفارشات مشتریان را در نظر بگیرید، که در آن یک سفارش شامل یک یا بیشتر مورد (آیتم) است که آن شرکت می‌فروشد. پایگاه داده شامل یک جدول است که در آن مشتری توسط شماره مشتری (کلید اصلی) شناسایی می‌شود؛ همچنین شامل یک جدول برای محصولات برای فروش است که توسط شماره محصول (کلید اصلی) شناسایی می‌شود؛ و همچنین شامل یک جفت جدول برای توصیف سفارش‌ها است.

یکی از جداول «سفارش‌ها» (**Orders**) نام دارد و آن جدول، یک شماره سفارش دارد (کلید اصلی) که با آن می‌توان این سفارش را شناسایی کرد، این جدول، یک شماره مشتری دارد (کلید بیرونی) تا تعیین کند که این محصولات به چه کسی فروخته می‌شود، بعلاوه شامل اطلاعات دیگری مثل تاریخ و زمانی است که سفارش در آن موقع انجام شده‌است، اینکه هزینه چگونه پرداخت شده‌است، نحوه ارسال چیست و غیره.

جدول دیگر «**OrderItem**» نام دارد؛ که نحوه شناسایی آن از طریق یک کلید ترکیبی شامل هم شماره سفارش (کلید بیرونی) و نیز یک عنصر شماره خط است. ویژگی‌های غیر کلید اصلی دیگری نیز مثل شماره محصول (کلید بیرونی) که سفارش شده، تعداد، قیمت، تخفیف، گزینه‌های اختصاصی دیگر و غیره در آن وجود دارد. ممکن است که صفر، یک یا تعداد بیشتر **OrderItem** متناظر با **Order** در آن باشد، اما اگر **OrderItem** متناظر موجود نباشد، هیچ **Order** ای نمی‌تواند وجود داشته باشد. (البته گاهی حالت **OrderItem** صفر به صورت موقت اعمال می‌شود، موقعی که سفارش اول وارد شده و قبل از آنکه اولین مورد سفارش شده ثبت گردد).

در اینجا جدول **OrderItem** دقیقاً یک «**موجودیت ضعیف**» است، زیرا یک **OrderItem** هیچ معنای مستقلی جدای از سفارش (Order) ندارد. ممکن است تصور شود که **OrderItem** خودش معنایی دارد، یعنی مواردی را ذخیره می‌کند که در خود رکورد شناسایی نمی‌شود، مثلاً تعداد یک محصول خاص در آن ذخیره شده‌است. این اطلاعات می‌تواند به خودی خود کاربردهایی داشته باشد، اما این یک «کاربرد محدود» است. برای مثال موقعی که بخواهیم در فروش یک محصول گرایش‌های فصلی یا مکانی را پیدا کنیم، ما به اطلاعاتی از رکورد مرتبط **Order** نیاز داریم.

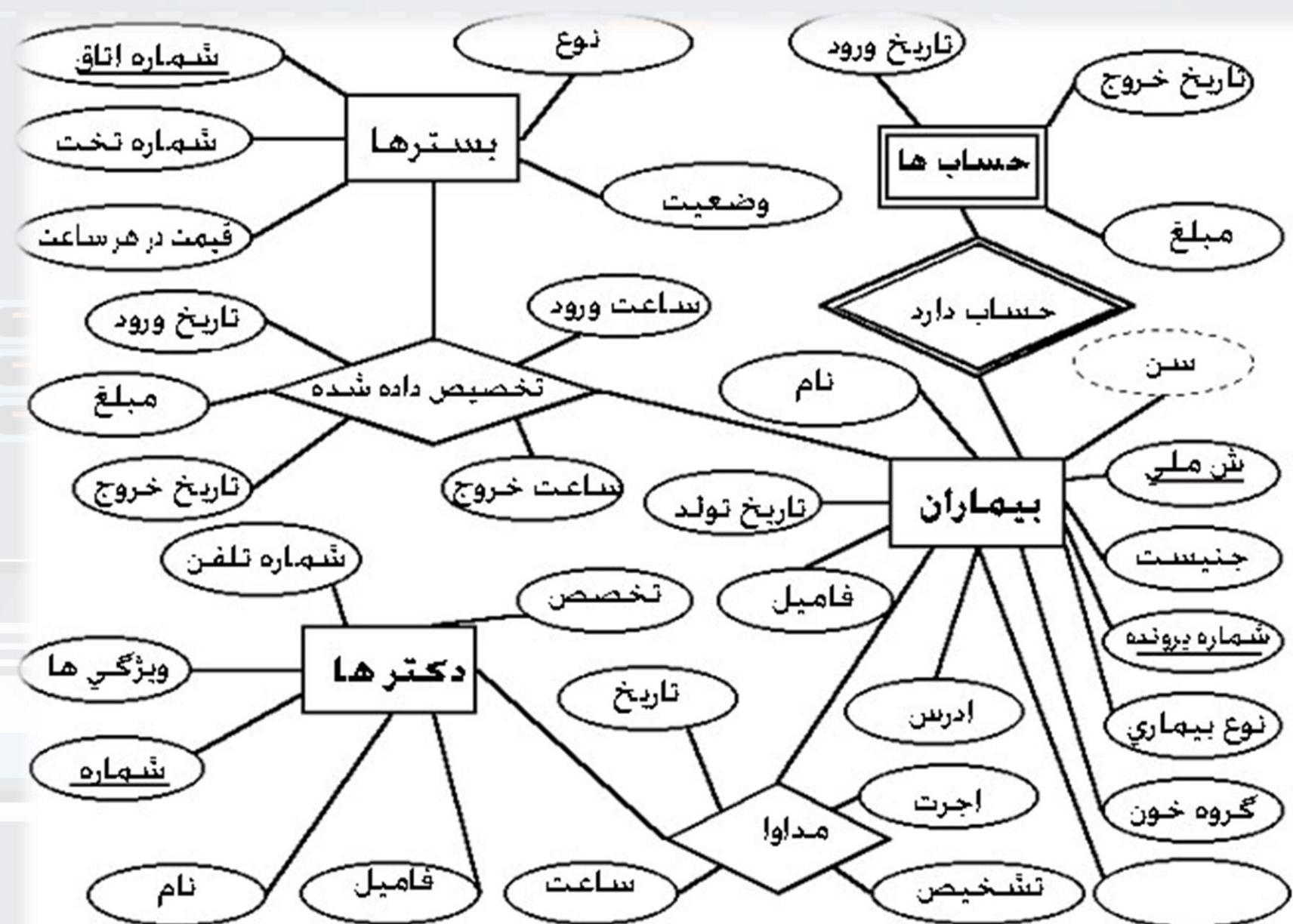
یک سفارش وجود ندارد مگر آنکه محصول و فردی که سفارش داده موجود باشد، بنابراین می‌توان بحث کرد که خود سفارش (Order) یک موجودیت ضعیف است و اینکه محصولات سفارش داده شده یک ویژگی چندمقداری سفارش است.

## تمرین:

### • ER مربوط به بیمارستان را براساس سناریوی زیر رسم کنید.

اینجا یک بیمارستان است. هر بیمار در بدو ورود باید فرمی شامل نام، فامیل، تاریخ تولد، شماره ملی، جنسیت، گروه خون و آدرس را پر کند. پرستار با دریافت فرم از بیمار یک پرونده برای او می سازد که این پرونده یک شماره دارد و این شماره برای هر بیمار منحصر به فرد خواهد بود. سن بیمار با توجه به تاریخ تولد او توسط سیستم قابل محاسبه است. هر دکتر در بیمارستان با مشخصاتی شامل نام، فامیل، تخصص، شماره تلفن و ویژگی استخدام شده است و یک شماره منحصر به فرد به آن شخص داده می شود. هر بیمار توسط یک پزشک مداوا می شود و تاریخ و ساعت مداوا، تشخیص بیماری و اجرت دکتر اطلاعاتی است که در هر دیدار پزشک و بیمار تولید می شود. هنگامی که نوع بیماری مشخص شود در پرونده بیمار مذکور درج خواهد شد. به هر بیمار یک بستر (تخت) اختصاص داده می شود که تاریخ و ساعت ورود و خروج و مبلغ این اختصاص با اهمیت است. ویژگی های هر بستر شامل هزینه برای هر ساعت، نوع، شماره اتاق، شماره تخت و وضعیت پر و خالی بودن است که قبل تخصیص دادن آن باید بررسی شود.

هر بیمار علاوه بر پرونده یک حساب هم در بیمارستان خواهد داشت که تاریخ ورود و خروج و مبلغی که باید پردازد در آن نگهداری می شود.



# جلسه کروم

- سروع: مسکلات ER
- پایان: آغاز SQL

# مشکلات ER:

از بزرگ ترین مشکلات ER وجود دام های پیوندی (Connection Trap) است که از درک نادرست بعضی از ارتباط ها ناشی می شود. مهم ترین این مشکلات پیوندی عبارت اند از:

## 1. دام حلقه ای:

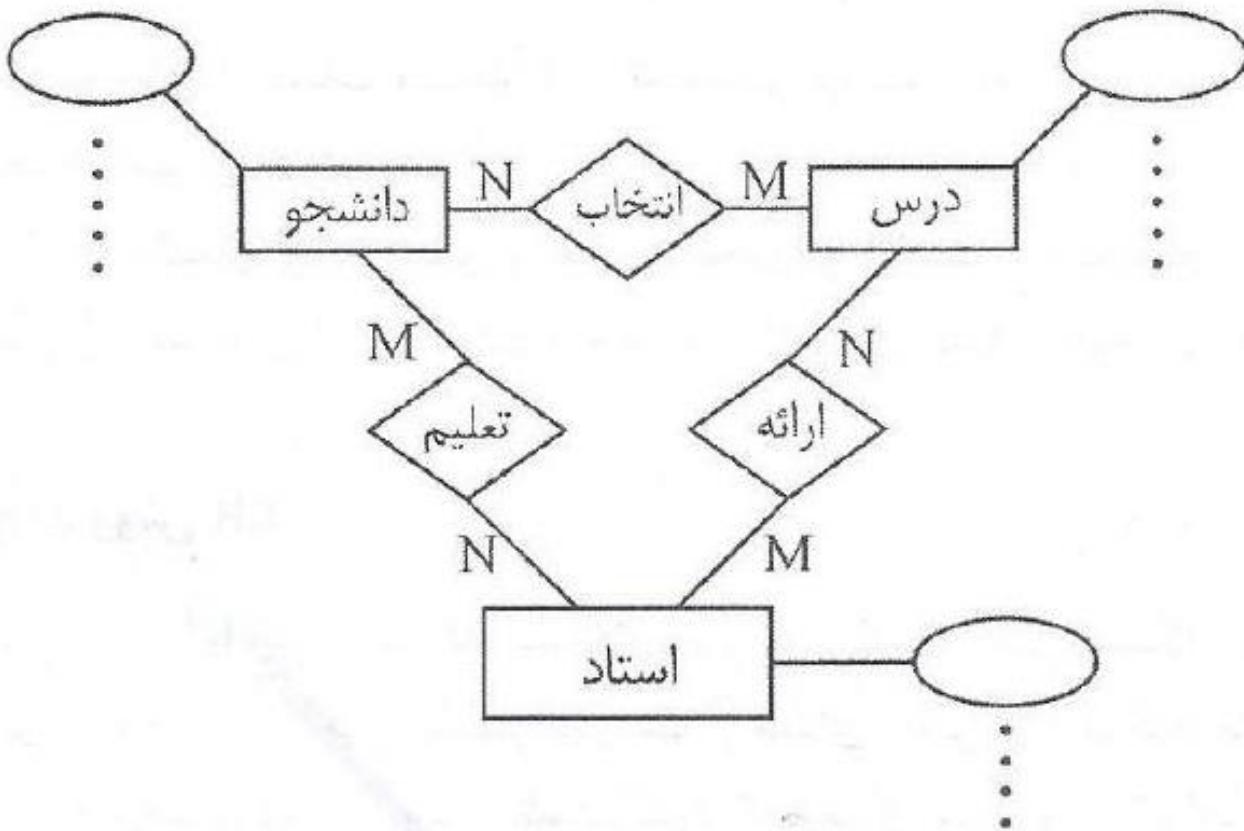
هنگامی ایجاد می شود که با داشتن سه ارتباط دو موجودیتی، وجود یک ارتباط سه موجودیتی را نتیجه بگیریم در وضعی که این استنتاج درست نباشد. در واقع این سه ارتباط نمی تواند به بعضی از سوالات ما پاسخ دهد برای مثال در شکل زیر برای سه پرسش اول پاسخ درست وجود دارد اما برای پرسش چهارم لزوماً پاسخی درست نداریم.

# ادامه‌ی دام حلقوی:

پرسش‌ها:

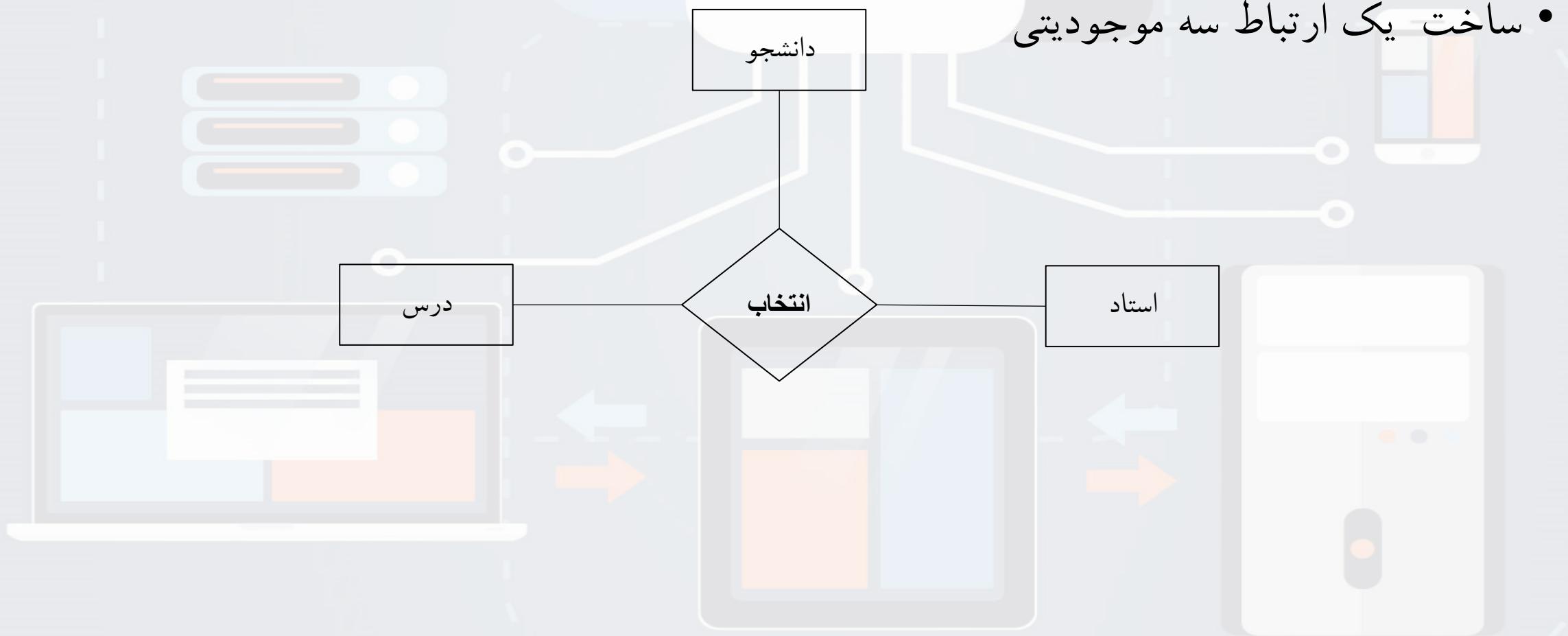
1. دروسی که استاد X تدریس می‌کند؟
2. دانشجویان درس Z چه کسانی هستند؟
3. دانشجوی Y چه درس‌هایی را دارد؟
4. کدام دانشجو درس Z را با استاد X برداشته است؟

نکته: با یک کوئیری نمی‌توان جمله‌ی چهار را به دست آورد.



# رفع دام حلقوی:

- ساخت یک ارتباط سه موجودیتی

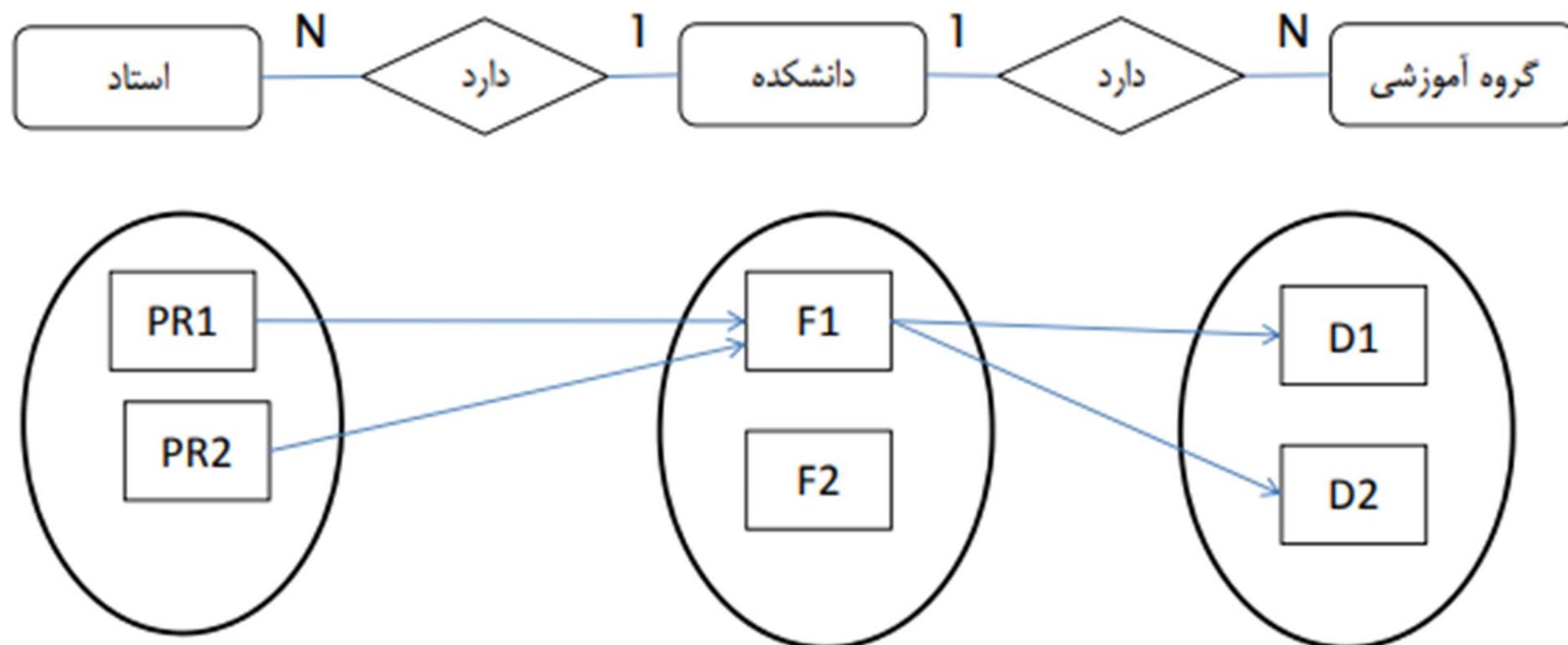


## ۲. دام چند شاخه (چتری-fan trap- یک چندی):

این نوع دام وقتی ایجاد می شود که بین یک نوع موجودیت E و هریک از دیگر انواع موجودیت F، G و ... ارتباط N:1 با مشارکت الزامی وجود داشته باشد، ولی ارتباط بین مثال F و G در مدلسازی دیده نشده باشد.

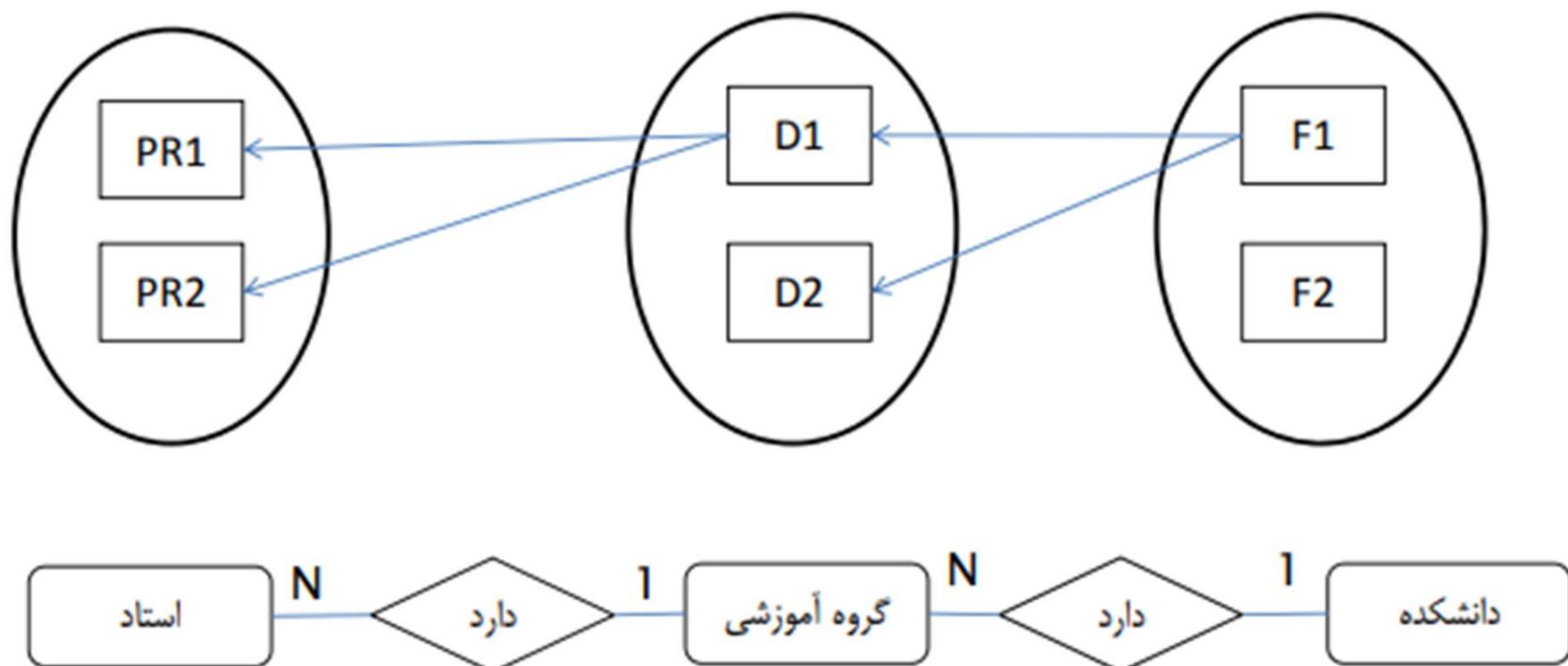
زمانی ایجاد میشود که ارتباطی بین چند نوع موجودیت وجود داشته باشد . ولی مسیر ارتباطی بین نمونه موجودیتهاي مشارکت کننده در ارتباط نامفهوم باشد . در واقع این مشکل زمانی ایجاد میشود که یک موجودیت از طریق ارتباط 1:n با دو (یا بیش از دو) موجودیت (که به طور مسقیم با هم در ارتباط نیستند) دیگر در ارتباط است.

## ادامه دام چندشاخه



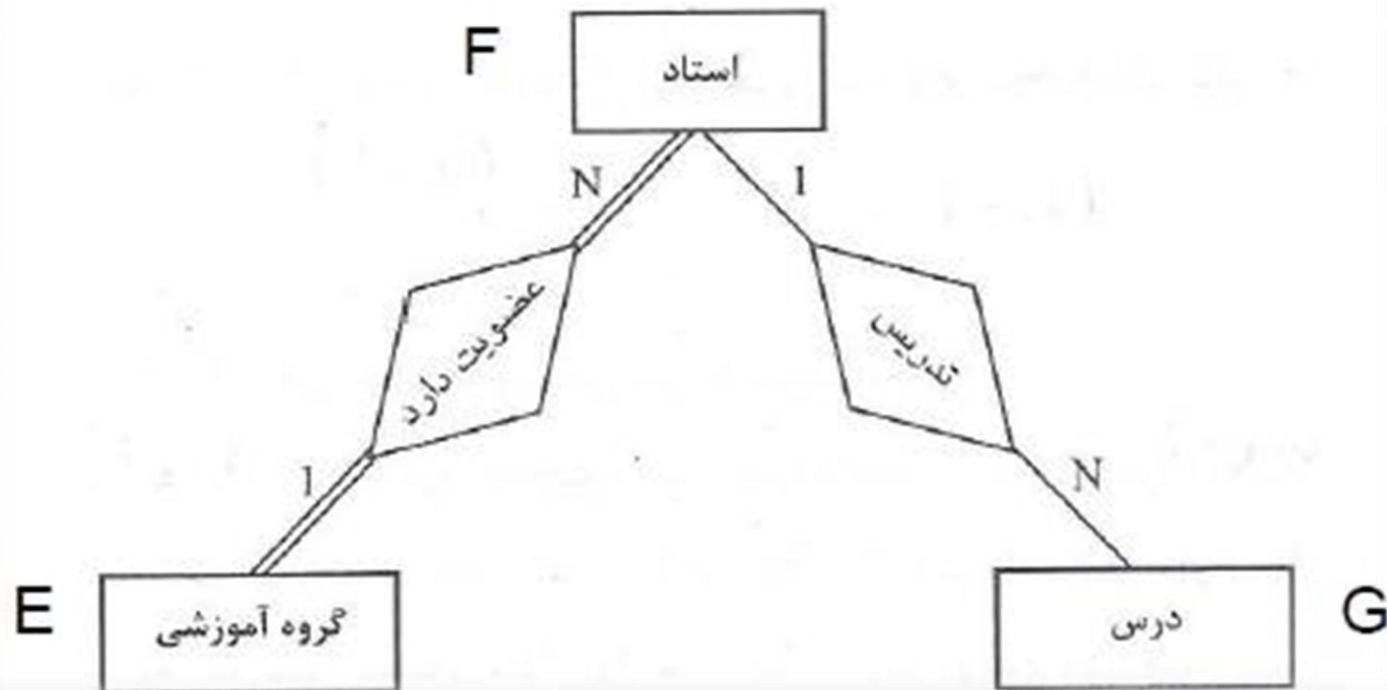
نمی توان به این سوال پاسخ داد که استاد در چه گروهی عضویت دارد ؟

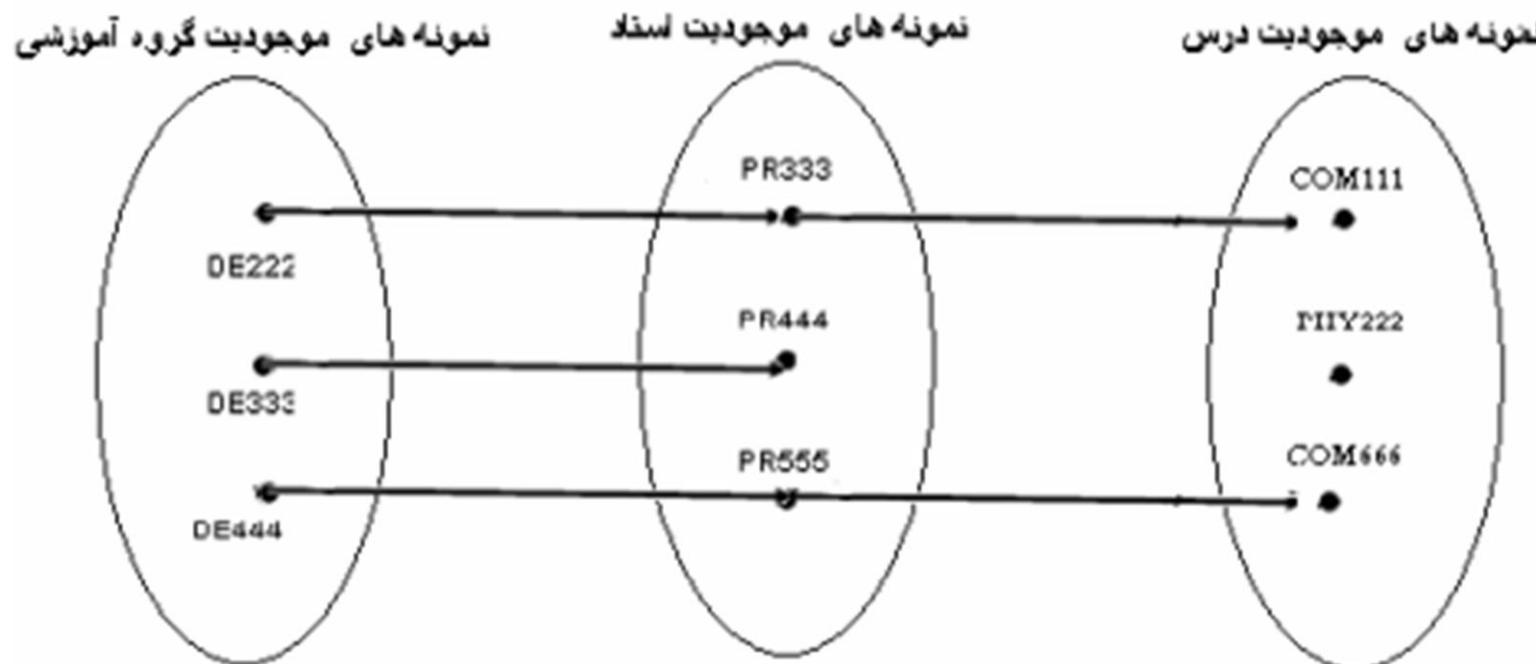
## رفع دام چند شاخه:



### ۳. دام گسل:

هنگامی ایجاد می شود که بین دو نوع موجودیت  $E$  و  $F$ ، یک ارتباط با چندی  $1:N$  و مشارکت الزامی وجود داشته باشد، ولی  $F$  خود با نوع موجودیت  $G$ ، ارتباط  $1:N$  با مشارکت غیر الزامی داشته باشد. به دلیل غیر الزامی بودن ارتباط بین  $F$  و  $G$ ، نمی توان همه اطلاعات دوم موجودیتی در مورد ارتباط بین نمونه های دو نوع موجودیت  $E$  و  $G$  را بدست آورد.





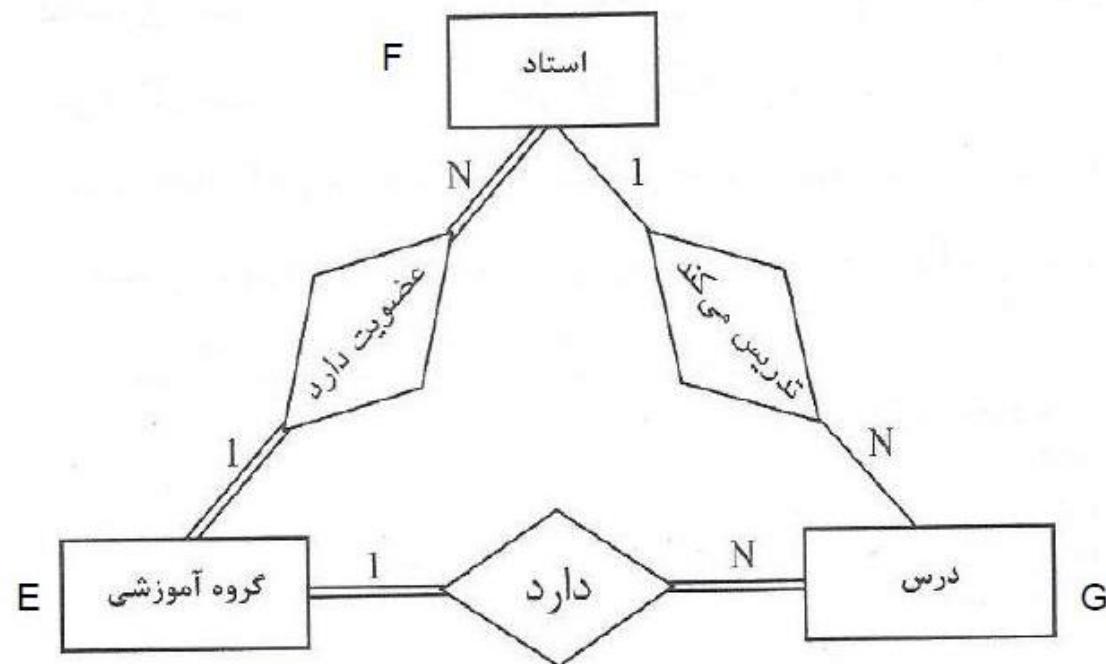
حال اگر این پرسش را مطرح کنیم :

درس PHY222 در کدام گروه آموزشی ارائه می شود؟

با مدلسازی انجام شده نمی توان به این پرسش پاسخ داد، زیرا استادی این درس را ارائه نمیکند . این مشکل از وجود دام شکاف ناشی می شود . به بیان دیگر الزامی نبودن مشارکت نوع موجودیت های عضو هیئت علمی و درس در ارتباط "تدريس" به این معناست که بعض اطلاعات در مورد گروه آموزشی ، مثلا درسها یی که ارائه می کند ، را نمی توان از طریق یک نوع موجودیت واسط ( در اینجا استاد ) بدست آورد.

# رفع دام گسل:

این مشکل زمانی بوجود می آید که یک موجودیت به عنوان پل ارتباطی دو موجودیت دیگر طراحی میشود ولی به خاطر مشارکت اختیاری اعضای آن موجودیت این ارتباط برقرار نمیشود . اگر مدلسازی به طرزی دیگرانجام شود ، این مشکل از بین می رود ، مثلا به صورت زیر :



# EER چیست؟

- Extended entity relationship

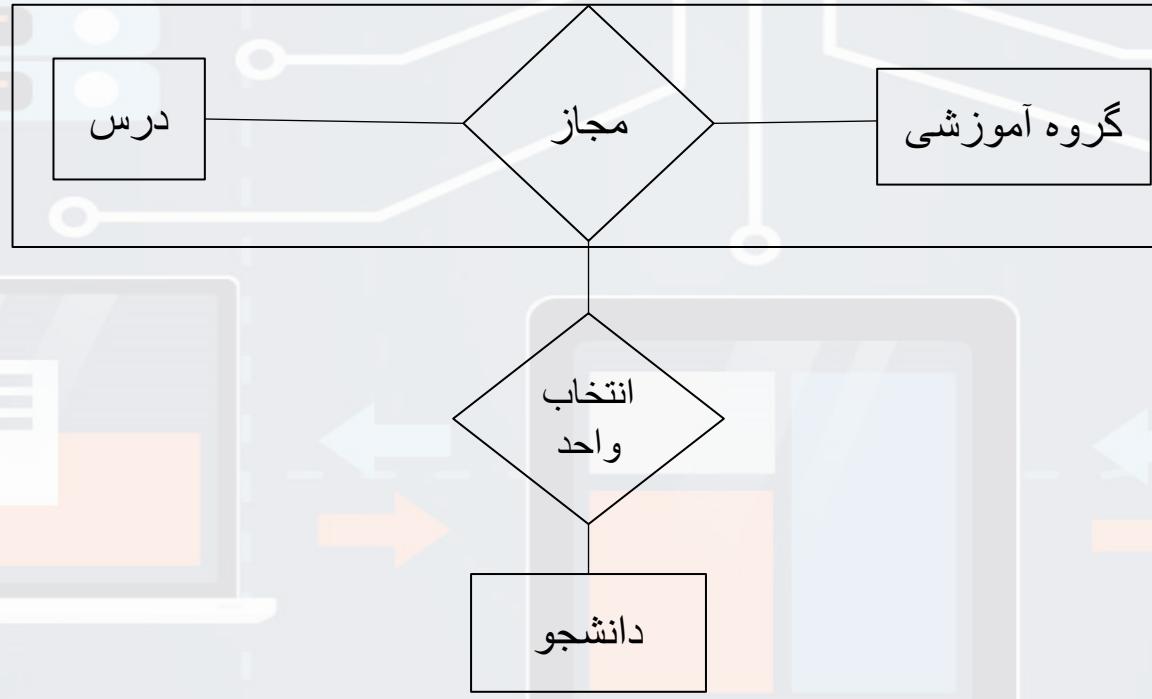
• برای رفع مشکلاتی که در رسم ER به وجود می‌آید حالت EXTENDED از ER ارائه شد که نام EER دارد.

<https://fa.tutorialcup.com/dbms/er-model-into-tables.htm>

# ویژگی ها:

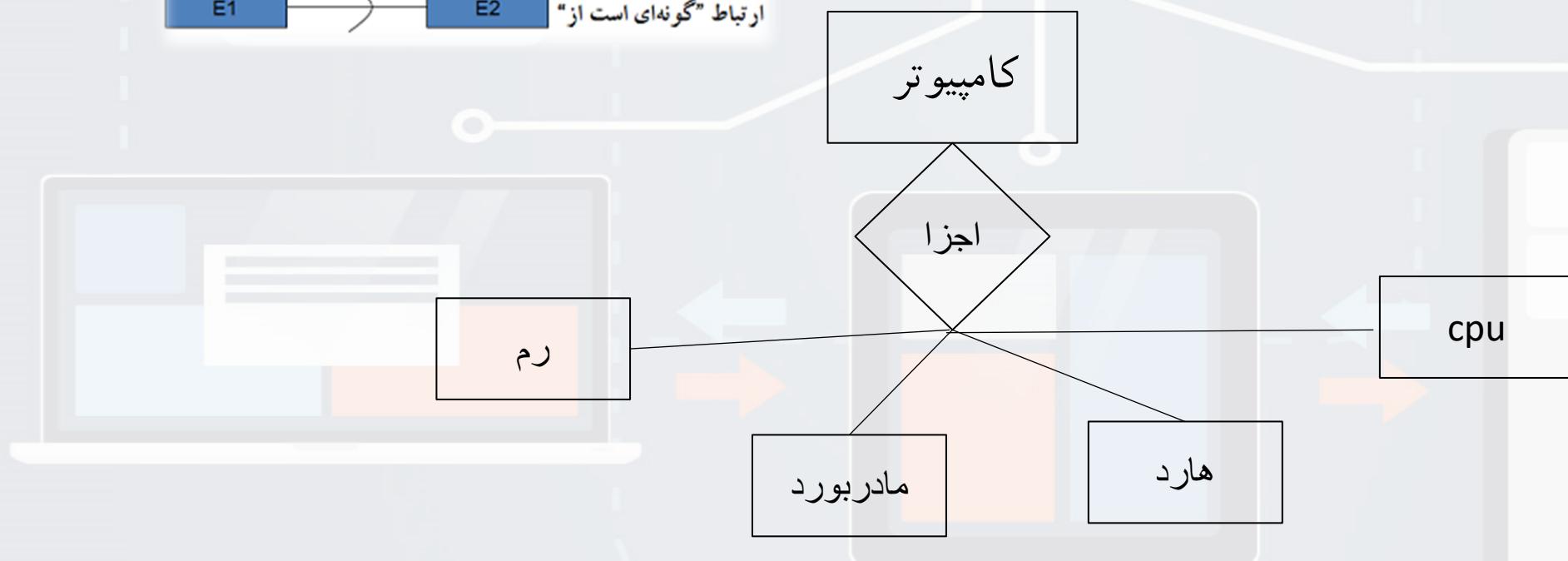
## ۱. تجمع (Aggregation)

میتوانیم یک ارتباط را به عنوان یک موجودیت در نظر بگیریم.



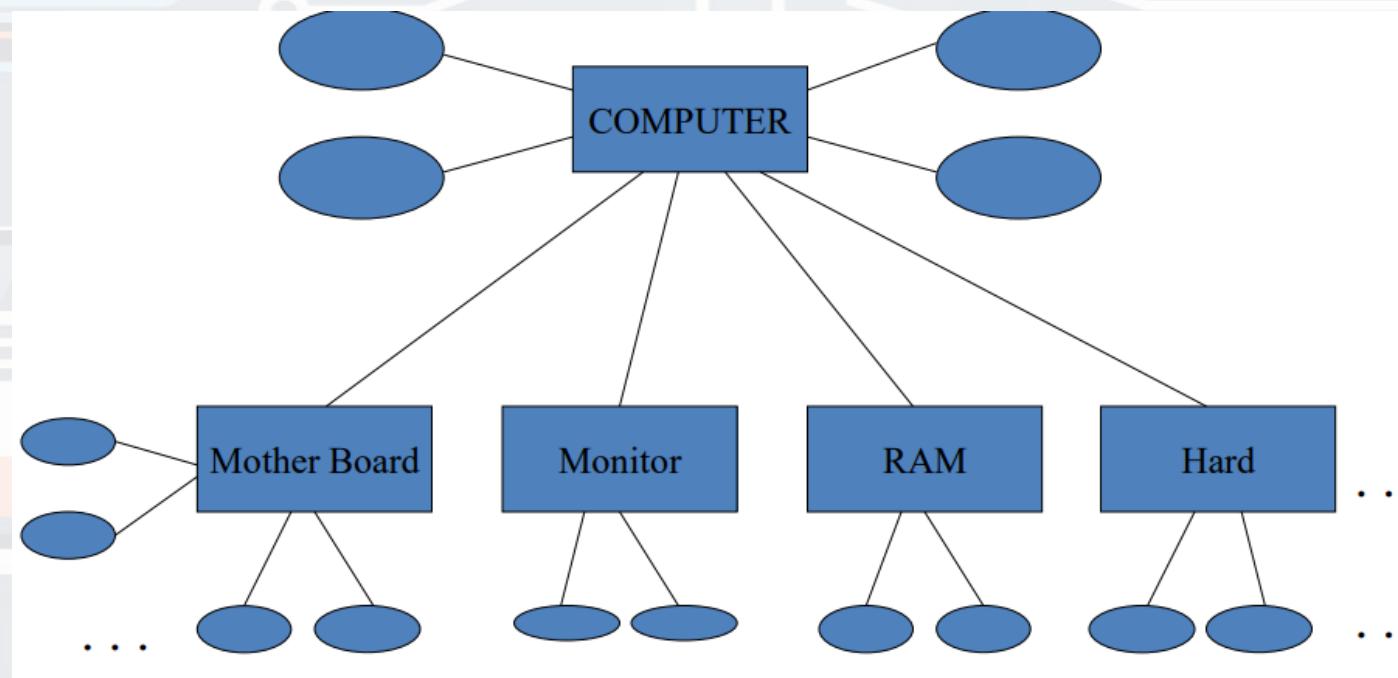
## 2. تجزیه (Decomposition)

تجزیه یا جداسازی یعنی یک شیء کل را به اجزاء تشکیل دهنده آن تقسیم کنیم. شیء کل صفات، ساختار و رفتار خود را دارد و هریک از اجزاء نیز صفات، ساختار و رفتار خاص خود را دارند. شیء کل شامل اجزاء خود است و بین شیء کل و اجزایش، ارتباط شمول وجود دارد. به این نوع ارتباط در EER، ارتباط "جزئی است از ..." گفته می‌شود.



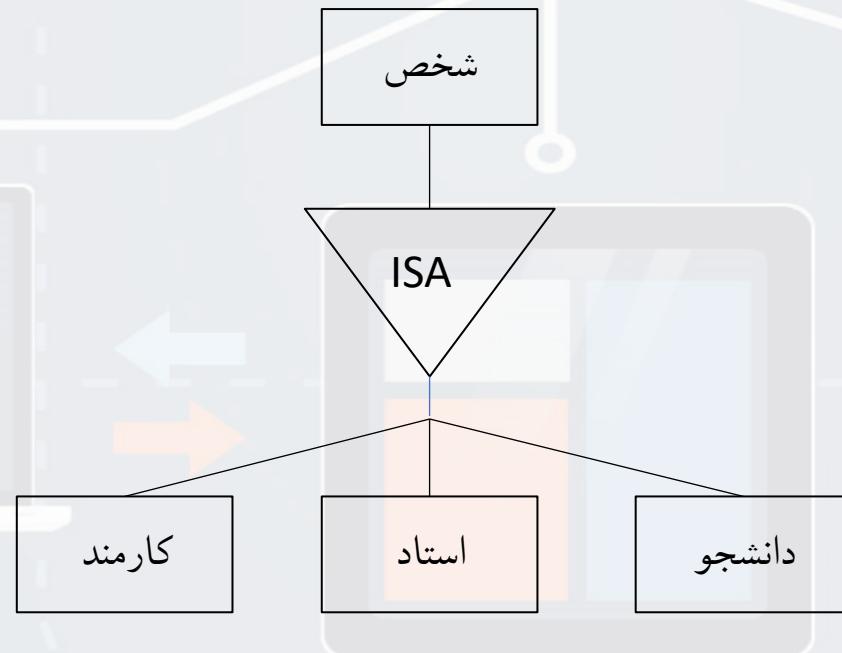
### 3. ترکیب(combination)

- ترکیب، عکس عمل تجزیه است و در این عمل، با داشتن ( $E_i = 1, 2, \dots$ ) یک نوع موجودیت  $E$  را بازشناسی می کنیم به نحوی که  $E_i$  ها اجزاء تشکیل دهنده آن باشند.



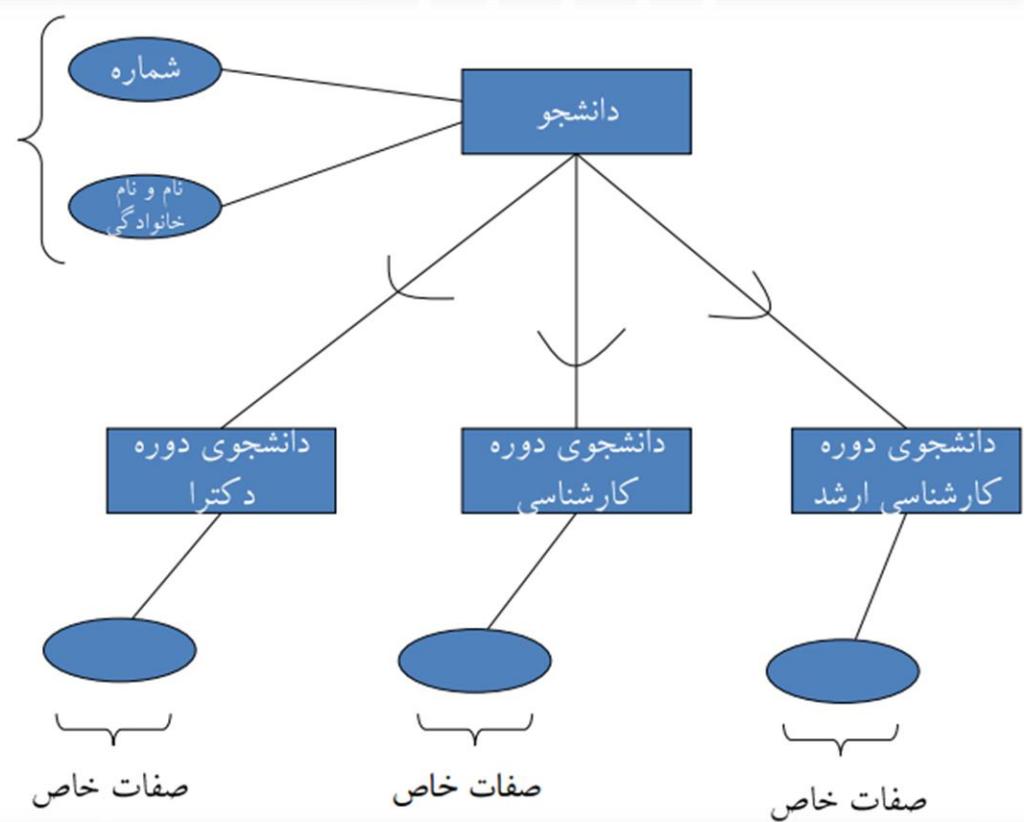
## 4. تخصیص (Specialization)

- تخصیص عبارتست از مشخص کردن گونه های خاص یک شیء براساس یک یا چند ضابطه مشخص، مثال اگر شیء موجود زنده را درنظر بگیریم، سه گونه خاص آن عبارتند از: انسان، حیوان و نبات.
- این ارتباط ارث بری است و اصطلاحا به آن رابطه (هست-یک) is-a می‌گویند و عموما به این شکل نشان داده می‌شود.



## ۵. تعمیم (generalization)

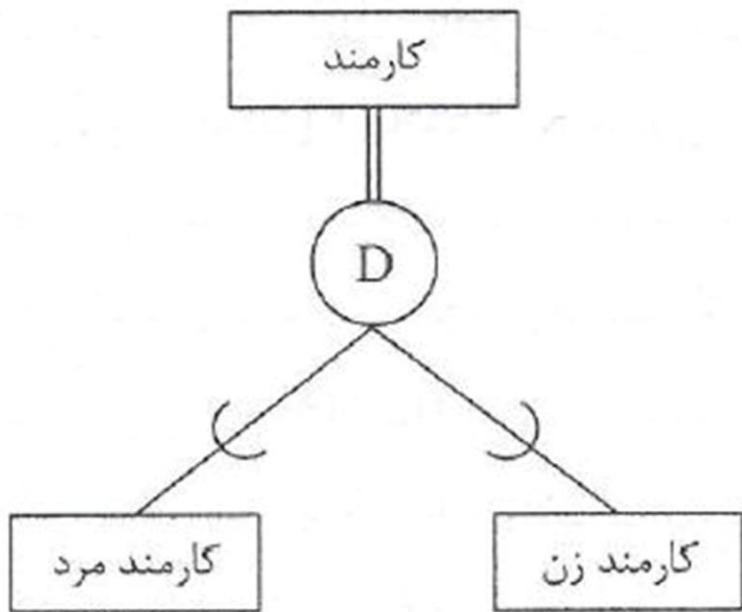
- تعمیم، عکس عمل تخصیص است، به این معنا که با داشتن زیرنوعهای خاص، صفات مشترک بین آنها را در یک مجموعه صفات برای یک زبرنوع موجودیت درنظر می‌گیریم.



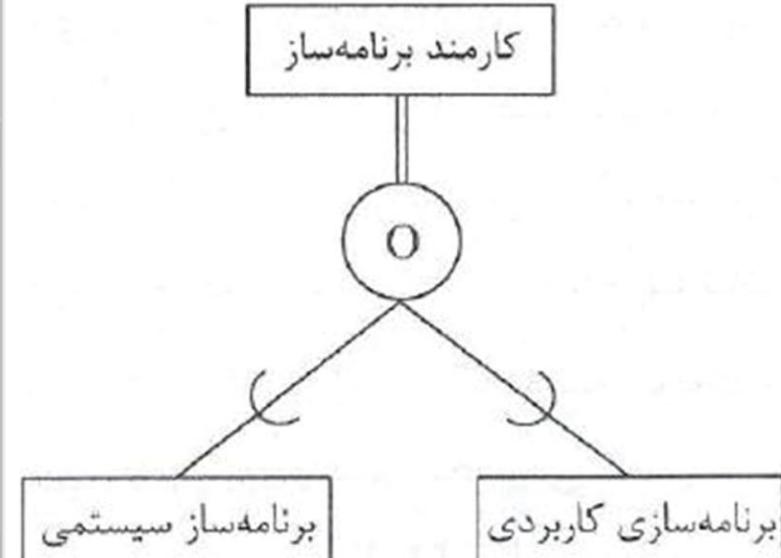
# انواع تخصیص (ISA)

در همپوشانی زیرنوع‌ها می‌توانند در یک لحظه مثل هم باشند یعنی کارمند برنامه ساز می‌تواند هم برنامه ساز کاربردی باشد و هم سیستمی اما کارمند زن همزمان مرد نمی‌تواند باشد.

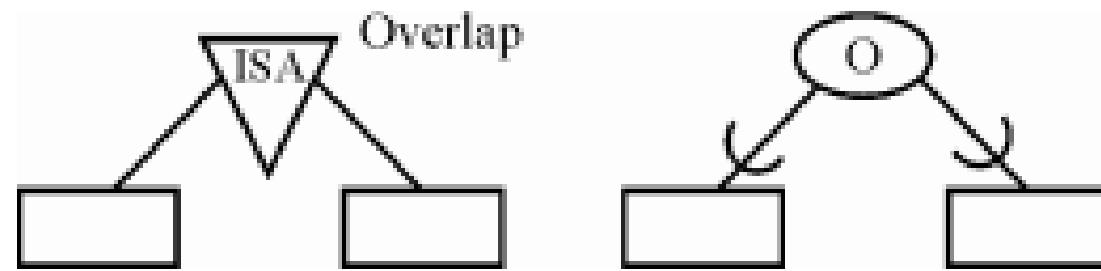
زیرنوعهای مجرزا



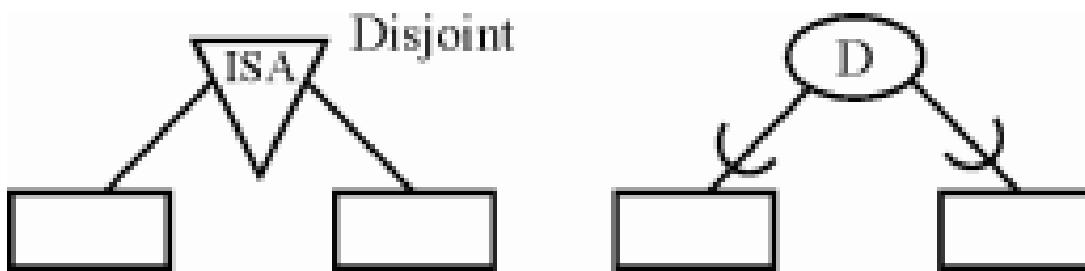
زیرنوعهای همپوشانی (با اشتراک)



رابطه پوشای تخصیص غیرمجزا (Overlap) مابین فرزندان و پدر به دو شیوه زیر نشان داده می‌شود:

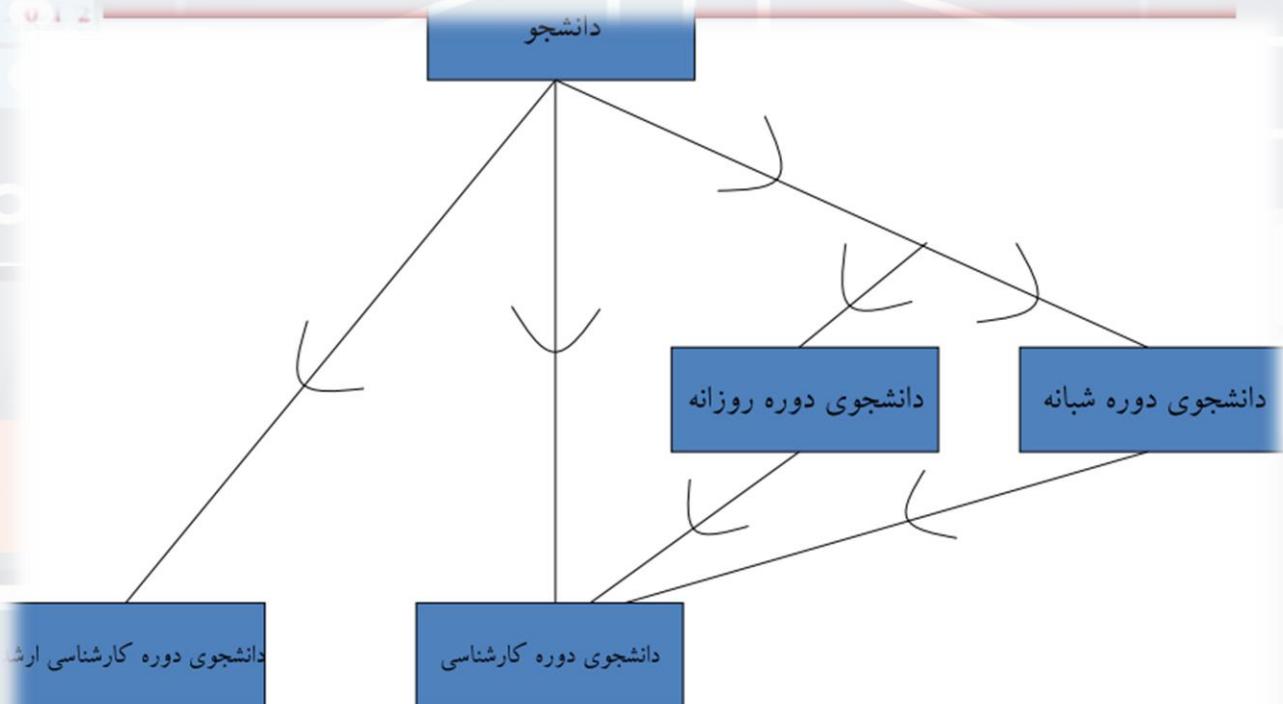


رابطه غیرپوشای تخصیص مجزا (Disjoint) مابین فرزندان و پدر به دو شیوه زیر نشان داده می‌شود:

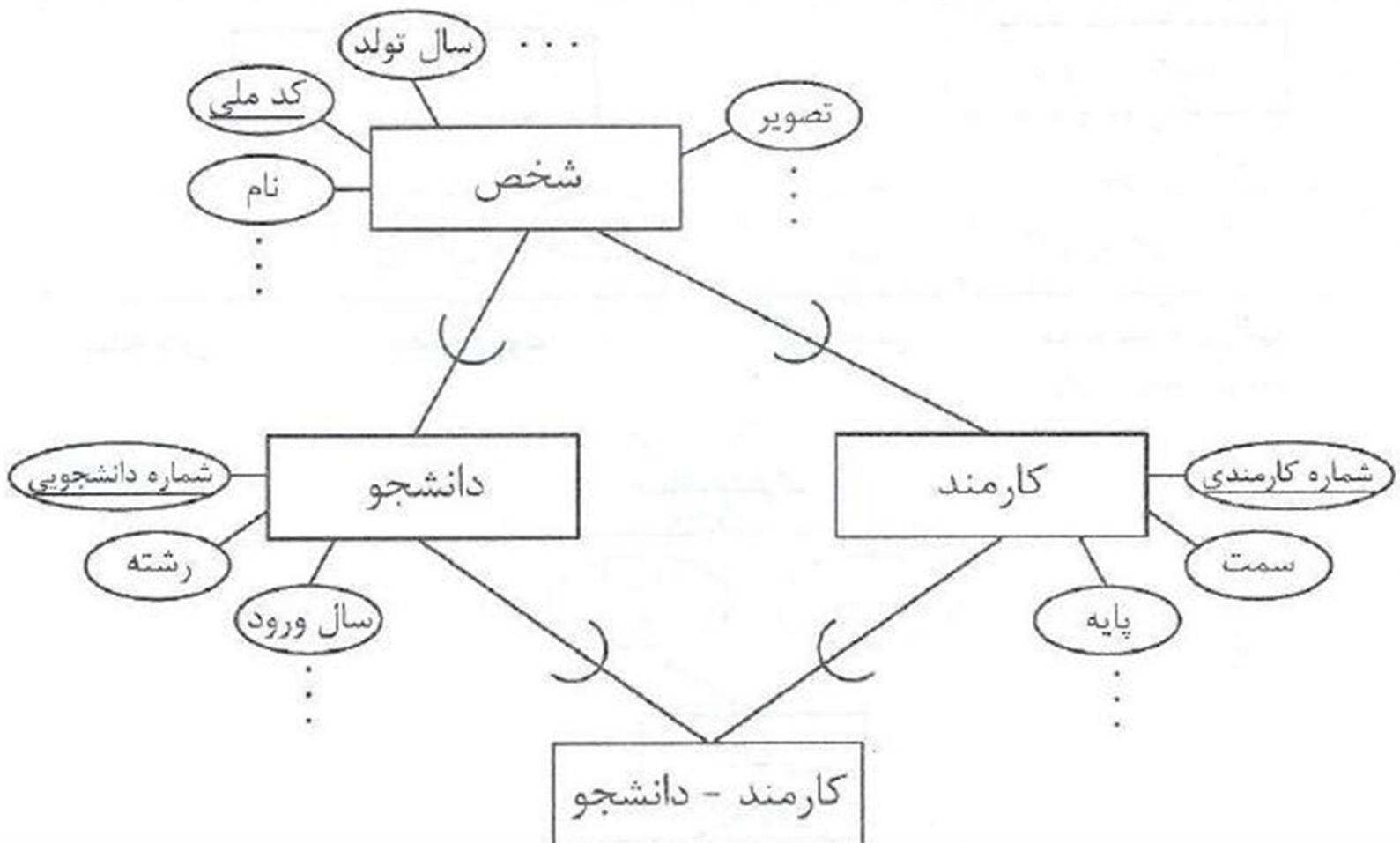


# وراثت چندگانه

- یک زیرنوع موجودیت، می‌تواند در عین حال زیرنوعیک زیرنوع موجودیت دیگر هم باشد. با این ترتیب می‌توان مفهوم وراثت چندگانه را در روش EER توضیح داد.

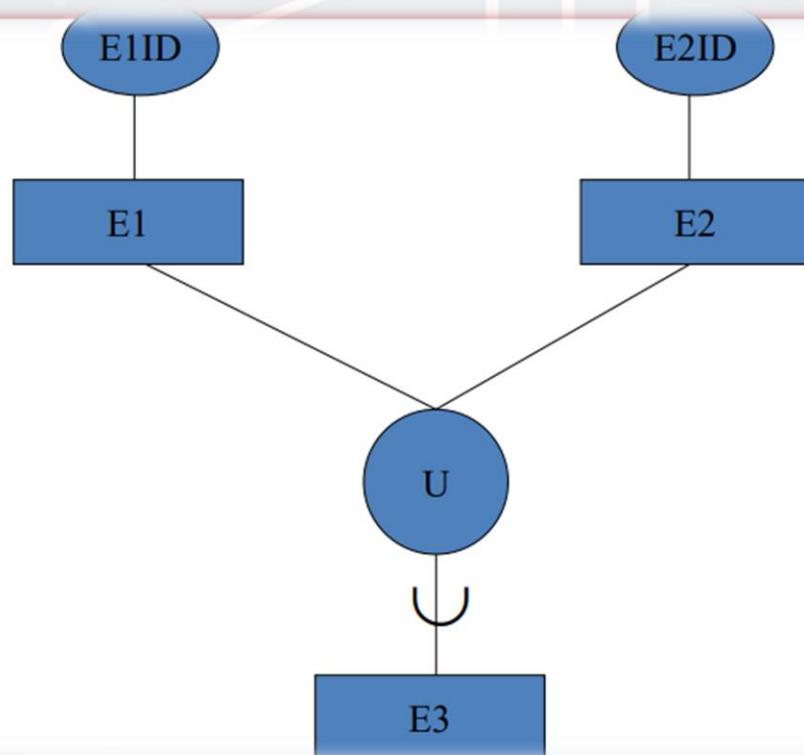


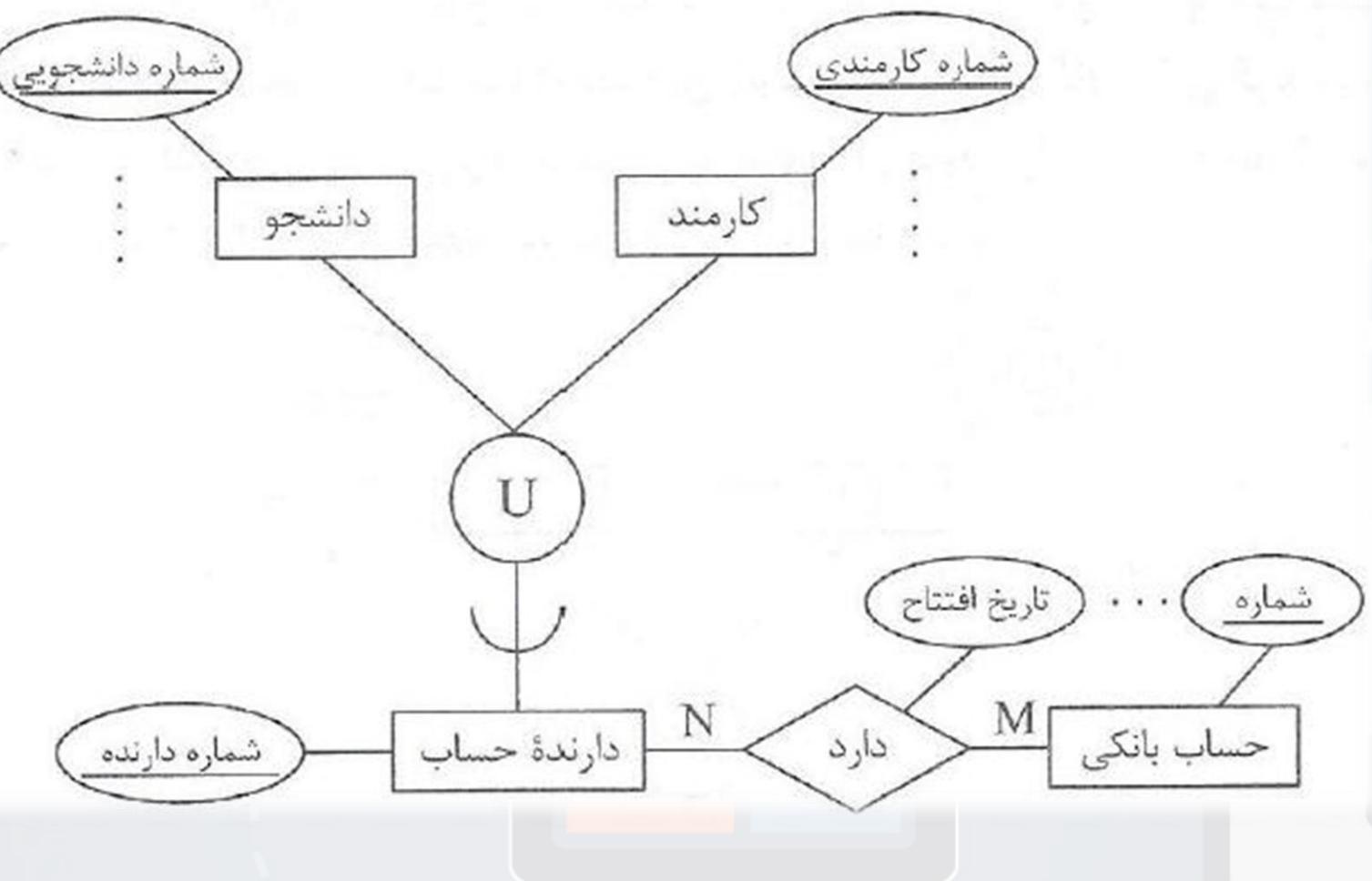
## مثال وراثت چندگانه



## دسته بندی

- یک زیرنوع می‌تواند زیرنوع بیش از یک زیرنوع باشد. ممکن است زیرنوعهای این زیرنوع، از یک نوع نباشند. به این زیرنوع اصطلاحاً دسته (طبقه) گویند. برای نمایش دسته، از نماد U استفاده می‌شود.





- ابزار آنلاین طراحی ER

- <https://editor.ponyorm.com/user/stare313/hospital/designer>
- <https://erdplus.com/edit-diagram/d3b66c0b-3f0a-44b5-b24b-837198defc53>

## تبدیل مدل ER به جدول:

- پس از اتمام طراحی نمودار ER، باید آن را در ساختار منطقی قرار دهیم که همان تبدیل به جداول است.
- با استفاده از نمودارهای ER می‌توان به راحتی مدل داده رابطه‌ای را ایجاد کرد که چیزی جز نمای منطقی پایگاه داده ندارد.
- برای تبدیل این نمودارها باید درستی از صفات موجودیت‌ها و چندی ارتباط داشته باشیم.
- اغلب به دو علت مهم موجودیت‌ها را در جدول‌های متفاوت ولی به همراه رابطه بینشان، ثبت می‌کنیم.
  1. حفظ امنیت داده‌ها در هر یک از جدول‌ها
  2. جلوگیری از افزونگی داده

- ارتباط جداول به یکی از دو صورت زیر امکان پذیر می باشد:
  - کلید خارجی: کلید اصلی یک جدول در جدول دیگری می آید
  - جدول ارتباط: جدولی که ارتباط بین دو یا چند جدول را مشخص می کند.

# انواع کلید

در مدل نهاد - رابطه پایگاه داده بعضی صفات با ویژگی های خاصی شناخته می شوند و هر کدام با توجه به خاصیتی که دارند کلید خاصی نام میگیرند. که عبارتند از:

1. ابر کلید **S.K Super Key**

2. کلید کاندید **C.K Candidate Key**

3. کلید اصلی در پایگاه داده **P.K Primary Key**

4. کلید فرعی **A.K Alternative Key**

5. کلید خارجی **F.K Foreign Key**

## • ابر کلید S.K Super Key

ابر کلید هر ترکیبی از صفت‌های کلید در پایگاه داده را دارند. این نوع کلید لزوماً کمینه نیست و میتواند دارای زیرمجموعه‌های مختلفی از کلیدها در پایگاه داده باشد. این ترکیب از صفات باید با هم یک موجودیت را از مجموعه موجودیت‌ها بصورت یکتا مشخص کند.

## • کلید کاندید C.K Candidate Key

- ابر کلیدی که هیچ زیرمجموعه‌ای از آن ابر کلید نباشد یک کلید کاندید است. برای مثال برای ابر کلید ویژگی کد ملی، که دارای زیرمجموعه جدیدی که خود یک ابر کلید باشد، نیست، میتواند یک کلید کاندید نیز به حساب می‌آید.

- فرض کنید در هر آدرس فقط یک نفر با نام مشخص زندگی کند پس دو ویژگی آدرس و نام، یک ابر کلید است چون هر فرد را بصورت منحصر به فردی نمایش میدهد. اما این مجموعه  $\{\text{آدرس و نام}\}$  دارای زیرمجموعه کوچکتری که خود یک ابر کلید باشد مثل  $\{\text{نام}\}$  یا  $\{\text{آدرس}\}$  نیست پس مجموعه  $\{\text{آدرس و نام}\}$  یک کلید کاندید است.

- هر رابطه حتماً حداقل یک کلید کاندید را دارد زیرا در بدترین حالت همه ویژگی‌ها یا صفات خاصه آن رابطه با هم کلید کاندید می‌شوند که به این رابطه، رابطه تمام کلید (All Key) گویند.

## • کلید اصلی در پایگاه داده P.K Primary Key

- هنگامیکه کلیدهای کاندید یک مجموعه رابطه مشخص شدند، آنگاه طراح بانک اطلاعاتی به سلیقه خود، یکی از آنها را عنوان کلید اصلی بر میگزیند. البته در انتخاب این نوع کلید در پایگاه داده دو نکته بسیار مهم است:
  - اول هر چه کوتاه تر بودن طول کلید اصلی از نظر طول رشته باشی و دوم اهمیت کلید اصلی نسبت به سایر کلیدهای کاندید در پاسخ گویی به پرس وجوهای کاربران. برای مثال در چند مثال قبل بهتر است که کد ملی عنوان کلید اصلی انتخاب شود تا مجموعه { آدرس و نام } چون اولاً طول یک ویژگی کمتر از طول دو ویژگی است و ثانیاً پرس وجوهای کاربران از طریق کد ملی انجام میشود و نه آدرس و نام. کلید اصلی **not null** است یعنی نمی تواند خالی بماند و مقداردهی آن اجباری است.

## • کلید فرعی A.K Alternative Key

هر کلید کاندیدی که عنوان کلید اصلی انتخاب نشده است یک کلید فرعی است و میتواند در موقع لزوم که به ندرت اتفاق می افتد به جای کلید اصلی مورد استفاده قرار گیرد. در انواع پایگاه داده بسیار کاربردی است و جایگاه ویژه ای دارد.

## ۰ کلید خارجی Foreign Key

اگر صفت خاصه A از رابطه R1 یک کلید اصلی باشد و همین صفت A در رابطه R2 نیز وجود داشته باشد صفت A در رابطه R2 یک کلید خارجی است که میتواند باعث ارجاع دو رابطه R1 و R2 نسبت به هم شود و در واقع این دو رابطه را به هم پیوند دهد. کلید خارجی تنها کلیدی است که میتواند مقدار Null را اختیار کند.

توجه کنید که، تنها راه ارتباط بین دو رابطه کلید خارجی نیست بلکه وجود هر صفت و ویژگی مشترک بین دو رابطه میتواند باعث ایجاد ارتباط بین آن روابط گردد. در روابط یک به چند، در طرف چند کلید خارجی وجود دارد و در طرف یک کلید اصلی و یا فرعی وجود دارد.

## نکات با اهمیت:

مراحل مختلفی در تبدیل ER به جدول و ستون وجود دارد. هر نوع موجودیت، ویژگی و رابطه در نمودار تصویر خاص خود را در اینجا نشان می‌دهد.

- \* نکته ۱: هر جدول یک نام منحصر بفرد دارد.
- \* نکته ۲: در تمام گام‌ها، کاری به صفات مشتق شده نداریم.
- \* نکته ۳: اجزای صفات ترکیبی را در جدول قرار می‌دهیم.
- \* نکته ۴: دقیق شود که آدرس نمی‌آید و اجزای آن نمی‌آید و صفت مشتق شده هم نمی‌آید و با استفاده از کوئری‌های دیگر استنباط می‌شود.
- \* ویژگی اصلی در نمودار ER کلید اصلی جدول می‌شود.
- \* قانون جامعیت داده: مقدار کلید اصلی NULL و تکراری نمی‌باشد.
- \* قانون جامعیت ارجاعی: مقدار کلید خارجی یا NULL است یا برابر کلید اصلی جدول مرجع است.

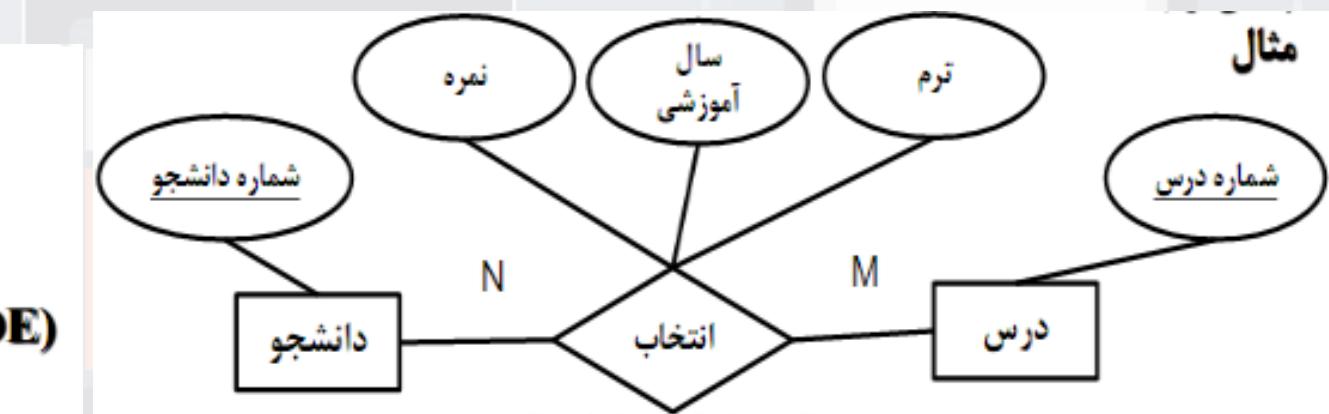
# قوانين اساسی تبدیل به جدول

## 1. چندی ارتباط N:M

برای این اتصال فقط حالت ارجاع متقابل استفاده می شود.

- در این حالت هر یک از موجودیت ها به یک جدول مستقل تبدیل خواهد شد.
- خود رابطه هم به یک جدول مستقل تبدیل خواهد شد.
- سپس کلید اصلی هر دو موجودیت در جدول رابطه به عنوان کلید خارجی در نظر گرفته میشود و هردو باهم به عنوان کلید اصلی در نظر گرفته میشوند.

- STT(STID,....)  
P.K.
- COT(COID,....)  
P.K.
- STCOT(STID, COID, **TR**, **YRYR**, **GRADE**)  
F.K.      F.K.  
P.K.



## ۲. چندی ارتباط N:1

برای این ارتباط دو حالت وجود دارد:

۱. حالت کلید خارجی: در این حالت هر موجودیت به یک جدول تبدیل خواهد شد سپس کلید اصلی طرف ۱ به جدول طرف N به عنوان کلید خارجی اضافه می‌شود.

۲. حالت ارجاع متقابل(cross-reffrence): جدول سومی شامل کلیدهای اصلی دو موجودیت.  
تذکر: صفت‌های ساده یا اجزاء ساده صفت‌های مرکب ارتباط یک به چند، به جدول جدید اضافه می‌گردند.  
مثال حالت اول:

• DEPT(DEID, DTITLE,.....)

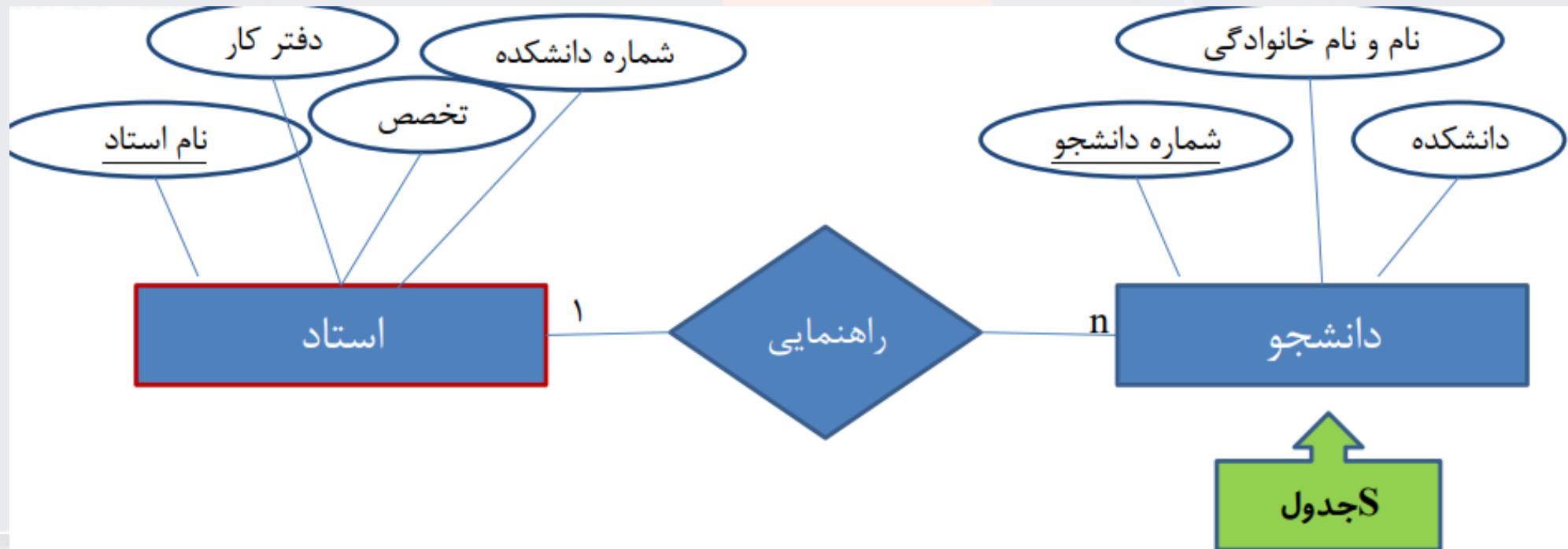
P.K.

• PROF(PRID, PRNAME, ..... ,DEID)

P.K.

F.K.





کلید خارجی

شماره دانشجو

نام و نام خانوادگی

نام استاد

دانشکده

ارجاع متقابل

نام استاد

شماره دانشجو

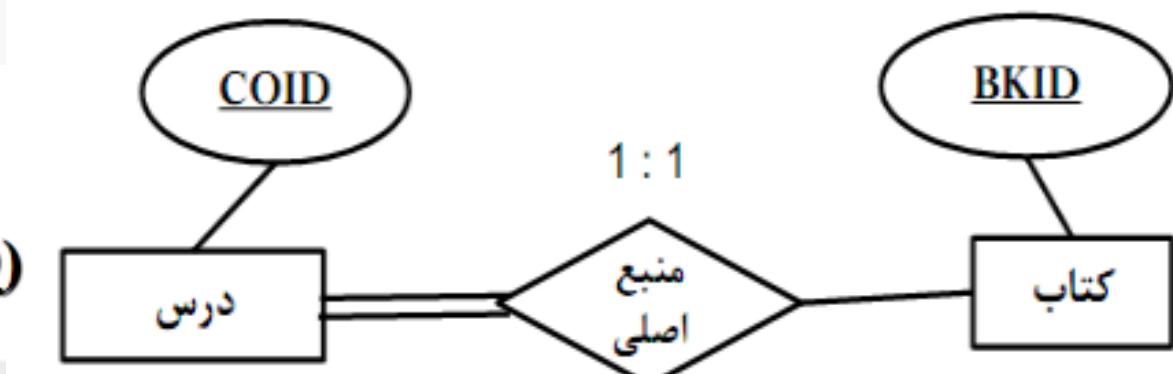
## ۳. چندی ارتباط ۱:۱

۳ حالت برای ارتباط درجه ۲ یک به یک داریم:

۱. حالت جدول ترکیب: هر دو موجودیت و ارتباط بین آن ها در یک جدول قرار بگیرد.(هردو طرف رابطه اجباری)
۲. حالت کلید خارجی:در این حالت هر یک از موجودیت ها به یک جدول تبدیل میشود و یکی از کلید های اصلی طرفین به طرف دیگر به عنوان کلید خارجی اضافه می شود.کلید خارجی را در جدولی قرار می دهیم که تعداد رکوردهای داشته باشد.(یک طرف رابطه اختیاری و طرف دیگر اجباری)
۳. حالت ارجاع متقابل(Cross- reference):جدول سومی با کلیدهای اصلی دو جدول موجودیت ها. (هردو طرف رابطه اختیاری)

مثال حالت دوم:

- **COT(COID, COTITLE,.....)**  
P.K.
- **BOOK(BKID, BKTITLE, ..... ,COID)**  
F.K.





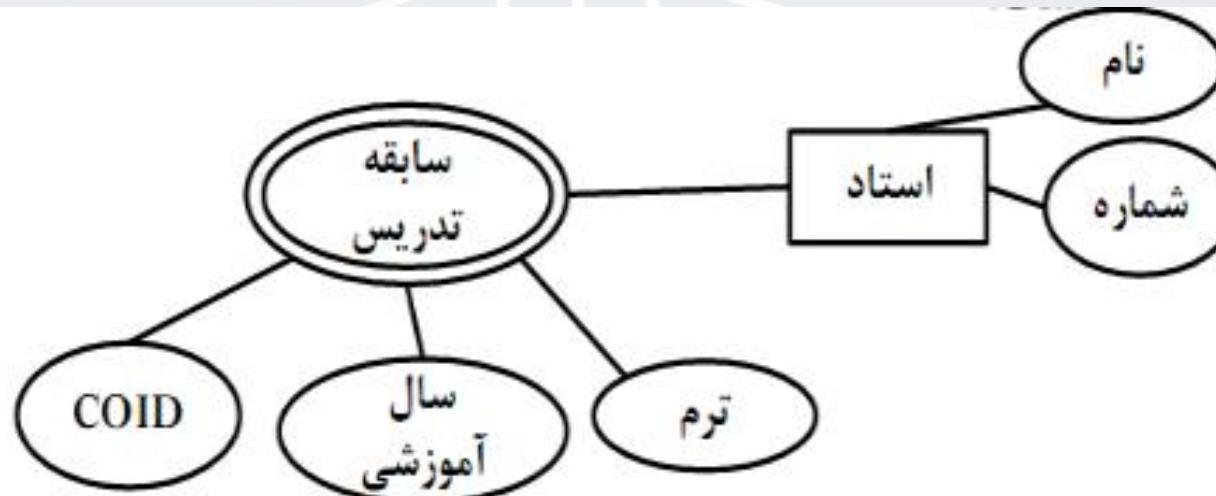
ترکیب جداول مشخصات شماره اموالی شماره اموالی نام استاد دفتر کار تخصص شماره دانشکده

کلید خارجی شماره اموالی شماره اموالی نام استاد دفتر کار تخصص شماره دانشکده

ارجاع متقابل نام استاد شماره اموالی

## ۴. صفت چند مقداری

\* به ازای هر صفت چند مقداره یک جدول جدید رسم می‌کنیم و کلید اصلی موجودیت و صفت چند مقداره را در جدول قرار می‌دهیم.  
ترکیب کلید اصلی و صفت چند مقداره کلید اصلی جدول جدید خواهد بود.



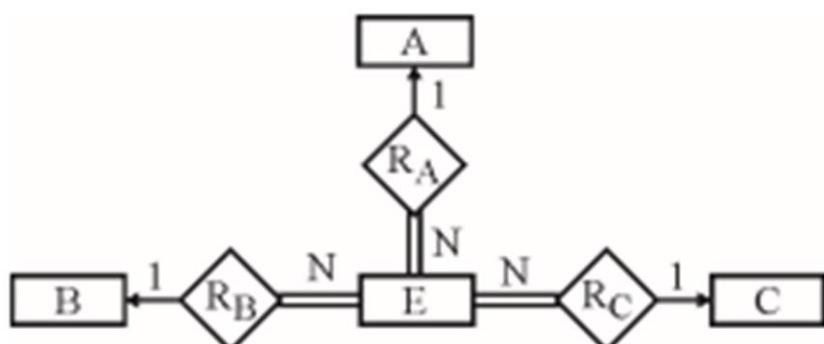
- **PROF(PRID, PRNAME, .....,  
P.K.)**
- **PRTEACHIS(PRID, COID, TR, YRYR)  
P.K.**

## ۵. نگاشت رابطه چند به چند با سه موجودیت

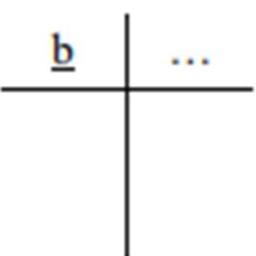
مدل تحلیل:



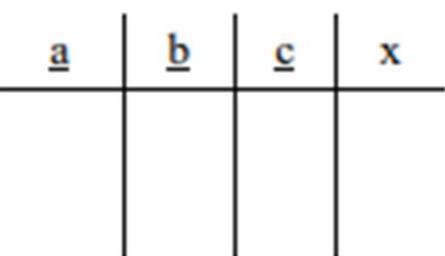
مدل طراحی:



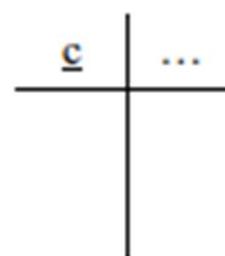
جدول A



جدول B



جدول ABC

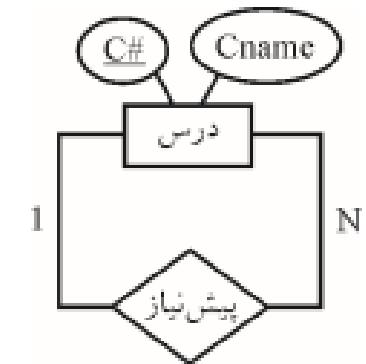
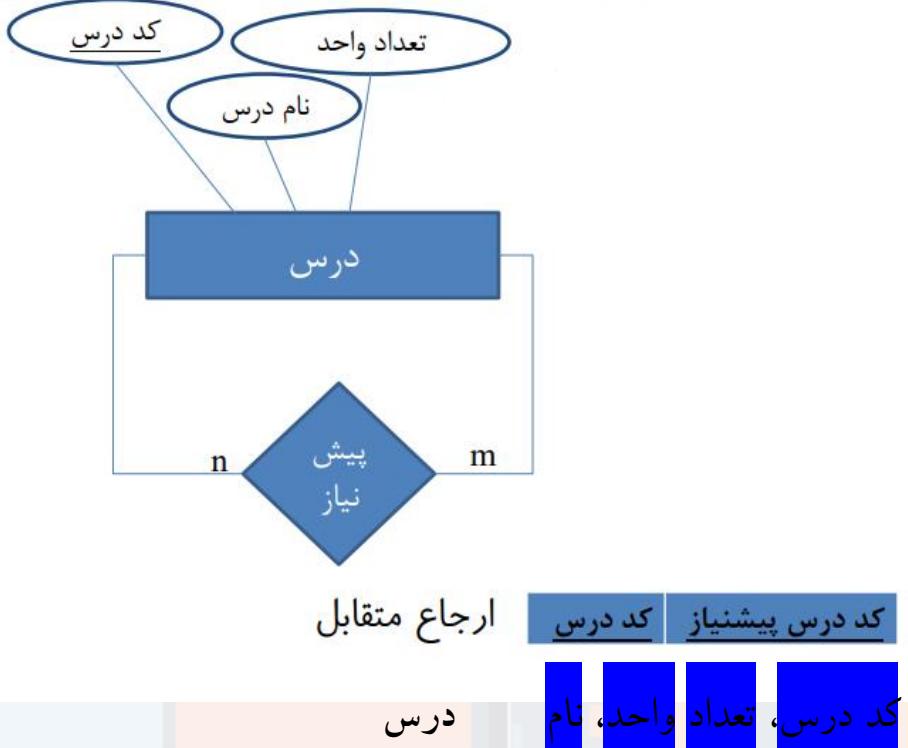


جدول C

## موارد خاص تبدیل به جدول

- نهاد ضعیف نیز به عنوان جدول نشان داده شده است. تمام صفات موجود ضعیف ستون جدول را تشکیل می‌دهد. اما ویژگی اصلی نشان داده شده در نمودار نمی‌تواند کلید اصلی این جدول را تشکیل دهد. ما باید یک ستون کلید خارجی اضافه کنیم، که ستون کلید اصلی موجودیت قوی آن باشد. این ستون کلید خارجی به همراه ستون ویژگی اصلی آن کلید اصلی جدول را تشکیل می‌دهد.
- هر ویژگی ترکیبی (مرکب) در همان جدول ستون‌های مختلف ادغام می‌شوند. برای مثال، آدرس یک ویژگی ترکیبی است. دارای درب #، خیابان، شهر، ایالت و پین است. این ویژگی‌ها به عنوان ستون‌های جداگانه در جدول دانشجو ادغام می‌شوند.
- در رابطه انعکاسی یک به چند، کد رابطه به عنوان کلید خارجی به جدول موجودیت اضافه می‌شود. در واقع کلید اصلی دوباره به عنوان کلید خارجی در جدول قرار می‌گیرد.
- در رابطه انعکاسی چند به چند، به دو جدول نیاز خواهیم داشت.
- در رابطه ارث بری باید کلید اصلی موجودیت پدر در همه‌ی فرزندان به صورت کلید خارجی قرار بگیرد.
- در تجزیه و ترکیب یک جدول اجزا داریم که شامل صفات کلید اصلی ترکیب، کلید خارجی (کد اجزا) و نوع هستند.
- در دسته بندی در جدول مشترک یک کلید خارجی و صفتِ نوع قرار می‌گیرد.

# رابطه انعکاسی



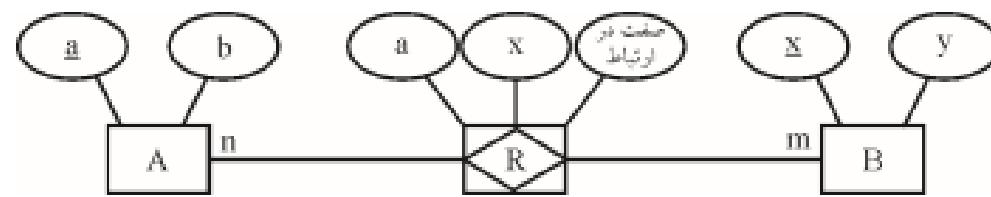
C#	Cname	CC#
C <sub>1</sub>	Cn <sub>1</sub>	NULL
C <sub>2</sub>	Cn <sub>2</sub>	C <sub>1</sub>
C <sub>3</sub>	Cn <sub>3</sub>	C <sub>1</sub>
C <sub>4</sub>	Cn <sub>4</sub>	C <sub>3</sub>
C <sub>5</sub>	Cn <sub>5</sub>	C <sub>2</sub>
C <sub>6</sub>	Cn <sub>6</sub>	C <sub>5</sub>

کلید خارجی (شماره درس)      کلید گاندید (پیش نیازها)

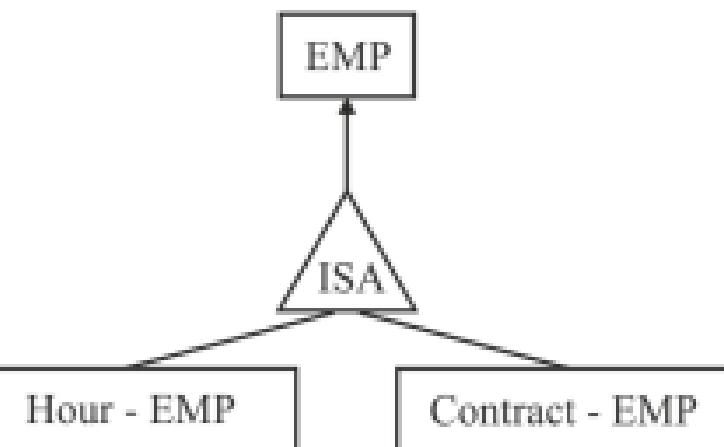
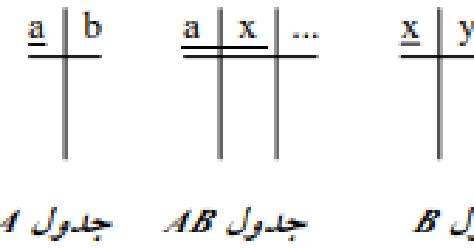


طوفه

مدل تحلیل:



مدل طراحی:

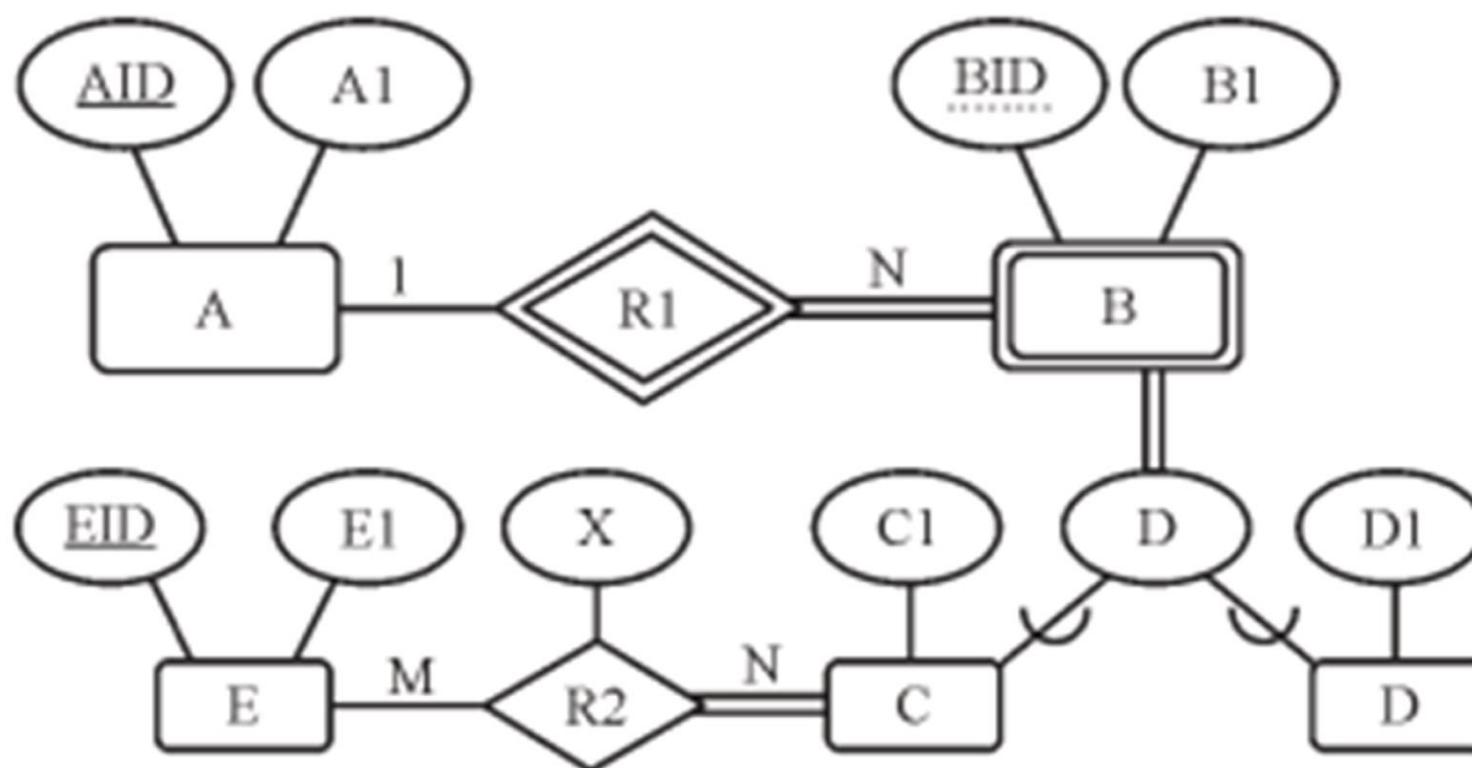


۲۶- با فرض وجود نمودار EER زیر کدام یک از رابطه‌های زیر به درستی طراحی نشده است؟

(مهندسی فناوری اطلاعات-دولتی ۹۵)

A (AID, A1)    B (BID, B1)    E (EID, E1, AID, BID)

C (AID, BID, BI, CI)    D (AID, BID, BI, DI)    R2 (AID, BID, EID, X)



A, B (۱)

B, E (۲)

C, D (۳)

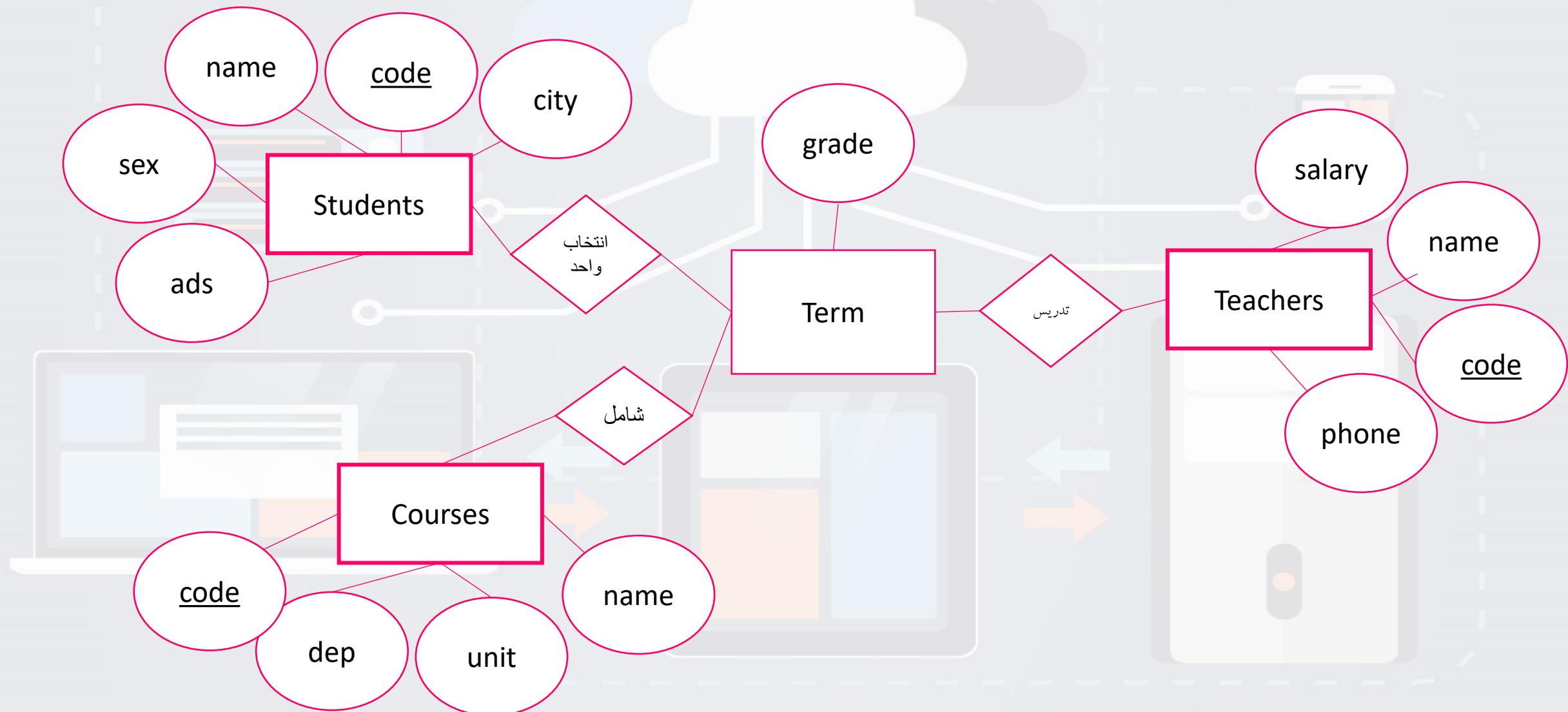
E, C (۴)

- B : (AID,BID,B1)
  - E : ( EID,E1)
  - A : ( AID ,A1)
- پاسخ صحیح گزینه دو است.

.3



تمرین: ER: زیر را به ساختار منطقی تبدیل کنید.



Term	
StudentCode	*
CourseCode	*
TeacherCode	*
Grade	

Students	
StudentCode	*
StudentName	
Adrs	
Sex	
City	

Courses	
CourseCode	*
CourseName	
Dep	
Unit	

Teachers	
TeacherCode	*
TeacherName	
Phone	
Salary	

# انواع دستورات sql

## (Data Definition Language )DDL •

دستوراتی که از آنها برای تعریف ساختار پایگاه داده استفاده می شود.

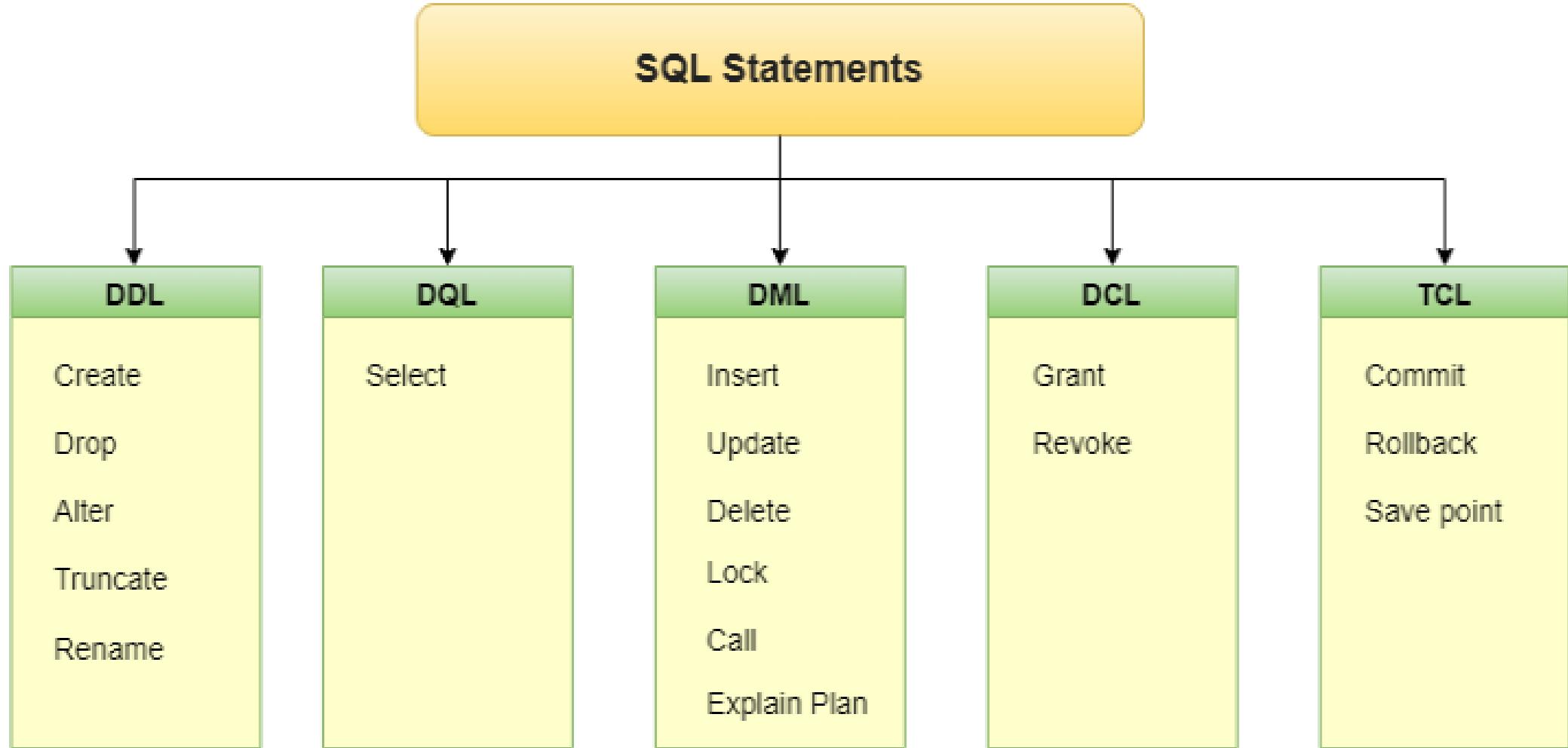
## (Data Manipulation Language )DML •

دستوراتی که از آنها برای تغییر داده های موجود در پایگاه داده استفاده می شود.

## (Data Control Language )DCL • کنترل سیستم پایگاه داده استفاده می شود.

## (Transaction Control Language )TCL •

دستوراتی که از آنها برای مدیریت تراکنش های (Transactions) پایگاه داده استفاده می شود.



# استاندارد نام گذاری



- `:camelCase` .۱. `upper camel case` یا پاسکال کیس
- `camel case` یا همون `lower camel case`.۲

عمل نوشتن ترکیبی کلمات یا عبارات است که در برنامه‌نویسی رایانه کاربرد دارد؛ به صورتی که هر کلمه به جز کلمه اول با حرف بزرگ آغاز می‌شود و بقیه حروف آن کلمه با حروف کوچک نوشته می‌شوند. مثل `iPhone`. در مستندات مایکروسافت در ویرایش شتری همیشه حرف اول با حروف کوچک نوشته می‌شود؛ مانند `backColor` اما در بین برنامه‌نویسان دات نت و حتی بسته به سلیقه برخی در خانواده `C` روش حرف اول بزرگ یا پاسکال کیس رایج است.

- `:snake case`

روش دیگر سرهمنویسی در مقابل شتری است که با زیرخط یا آندرلاین تفکیک می‌شود و در نام‌گذاری پایگاه‌های داده و همچنین در آدرس‌های وب رایج است. مثل `text_to_integer`

# اصول کدنویسی در sql server

## • اصل اول: نام گذاری اسامی

تنها از کarakترهای الفبایی، ارقام و خط زیر (**underline/underscore**) برای نام گذاری اسامی (متغیرها، جداول، ستونها و ...) استفاده شود. این زیر مجموعه از کarakترها برای نام گذاری اسامی در زبان های برنامه نویسی دیگر نیز بکار گرفته می شود و اگر بخواهیم از یک نام یکسان در Database و زبان برنامه نویسی استفاده کنیم مشکلی بوجود نمی آید.

## • اصل دوم: نامگذاری ستونها و متغیرها

تنها از حروف کوچک lower case برای نام گذاری ستونها، متغیرها و پارامترها استفاده شود. حروف کوچک نسبت به حروف بزرگ تر ساده تر و سریع تر خوانده می شوند به همین علت در روزنامه ها و کتب از حروف کوچک استفاده شده است. بطور نمونه "اولین همیار دانشجو" اگر نام یک ستون باشد طریقه‌ی صحیح نام گذاری آن "first\_student\_partner" خواهد بود (مستقل از ترتیب صحیح قرار گیری صفات). همانطور که شاهد هستید با کarakتر underscore کلمات را از همدیگر تفکیک کردیم تا از این اصل پیروی کرده باشیم. (snake case).

## • اصل سوم: نامگذاری جداول

اسامی جداول، ویوها و رویه های ذخیره شده Capitalize (stored procedure) را کنید به این معنا که حرف اول نام را بزرگ بنویسید. بطور مثال جدولی به نام "محصولات" به شکل "Products" نامگذاری شده و اسامی که شامل چند کلمه هستند را می توانیم به شکل پاسکال کیس بنویسیم؛ مثل **ProductSupplierAttribute**.

## • اصل چهارم: کلیدواژه ها با حروف بزرگ نوشته شده باشند

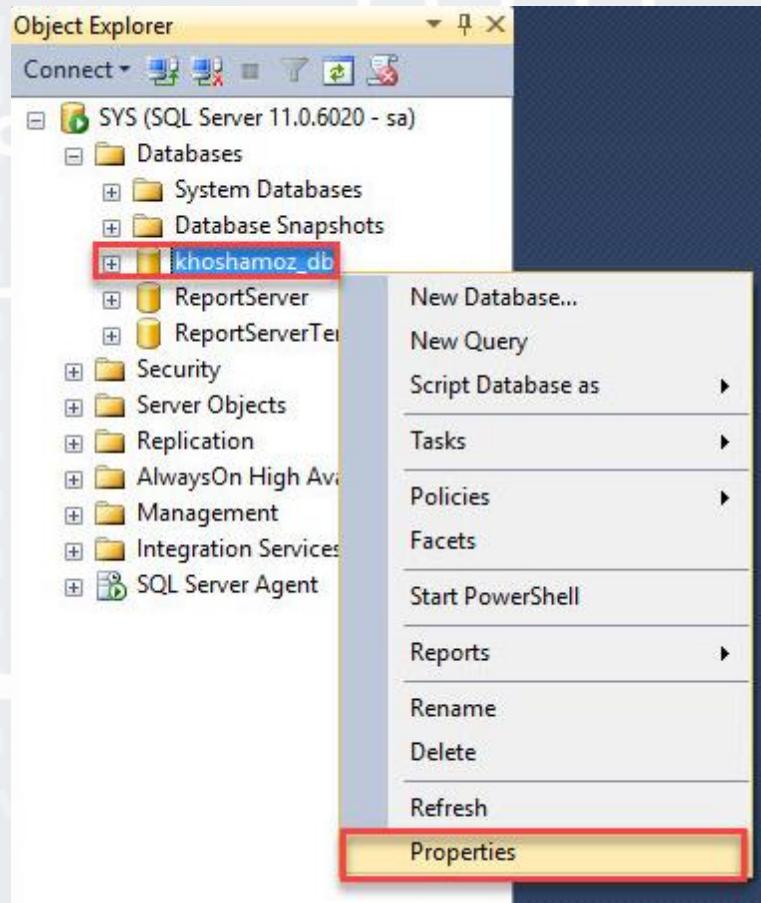
تمام کلید های رزرو شده مثل SELECT و ... را با حروف بزرگ بنویسید. با این کار به چشم کمک می کنید تا ماده های یک عبارت SELECT از هم تفکیک شده و بتوانیم به سادگی بین ماده های مختلف جابجا شویم.

بقیه اصول در لینک زیر موجود است:

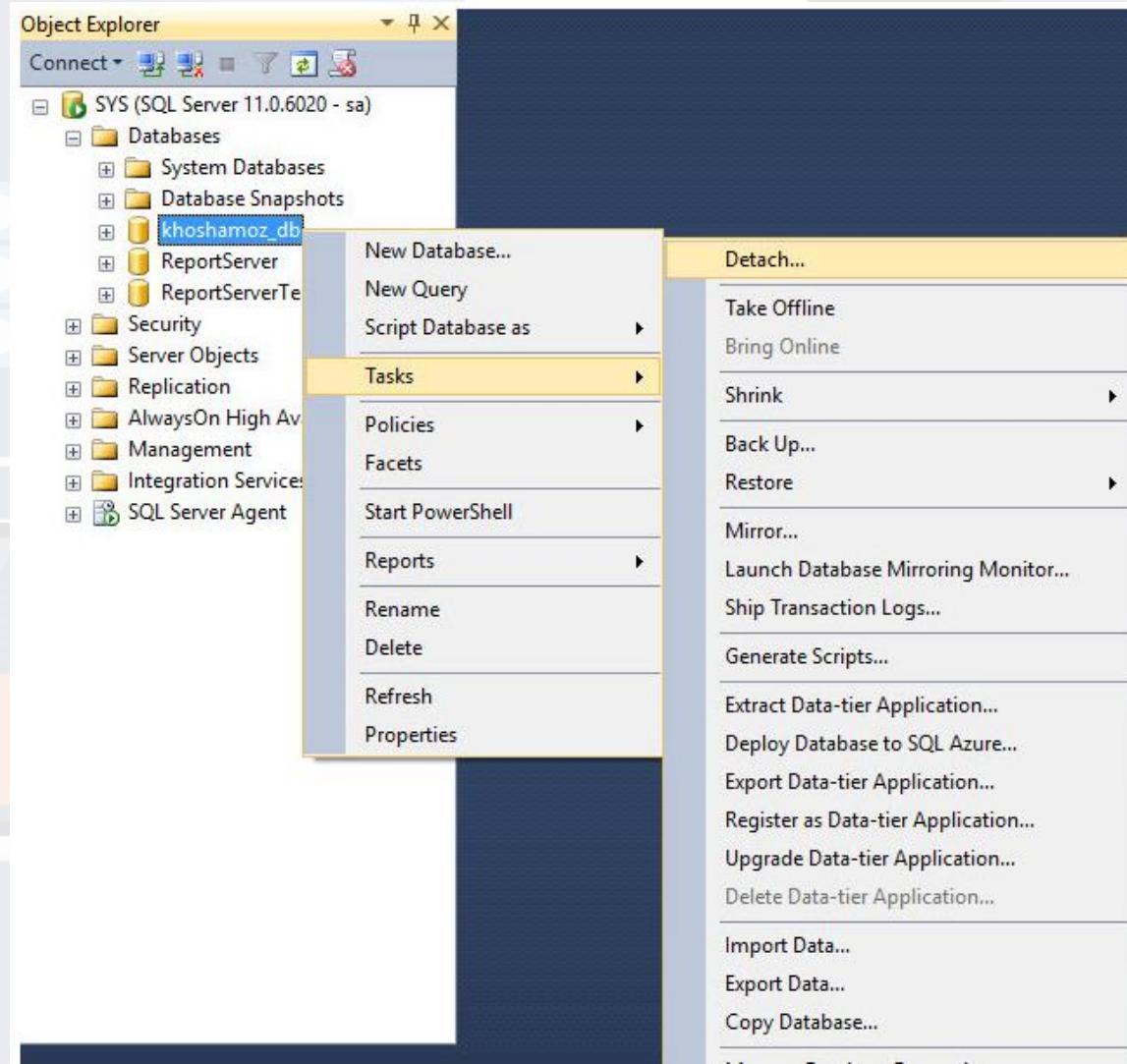
<http://www.30sharp.com/print/13/270/11/%D8%B3%D8%A8%DA%A9-%D9%88-%D8%B4%DB%8C%D9%88%D9%87-%D9%86%DA%AF%D8%A7%D8%B1%D8%B4-%DA%A9%D8%AF%D9%87%D8%A7%DB%8C-sql.aspx>

# انتقال فایل های دیتابیس

1. وارد کردن یک فایل mdf. به برنامه و استفاده از آن (attach)



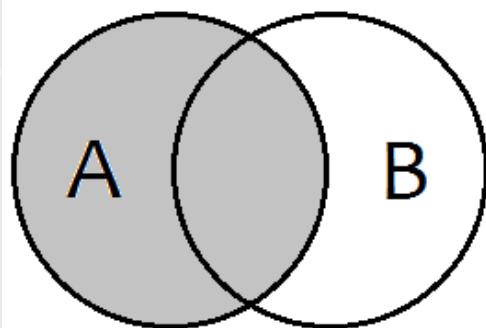
## 2. انتقال فایل دیتابیس که ما آن را ساخته ایم به سیستم یا درایو دیگری



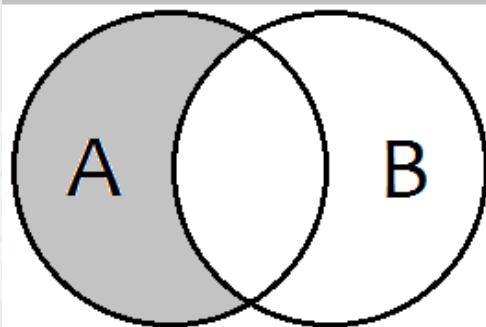
# ساختار نحوی دستور select

SELECT [ستون‌ها]  
FROM [جدول]  
[WHERE [شرایط]]  
[ORDER BY [شرایط]]  
[GROUP BY [ستون‌ها]]  
[HAVING [شرایط]]

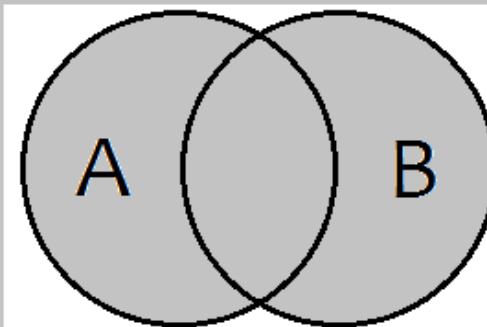
# SQL JOINS



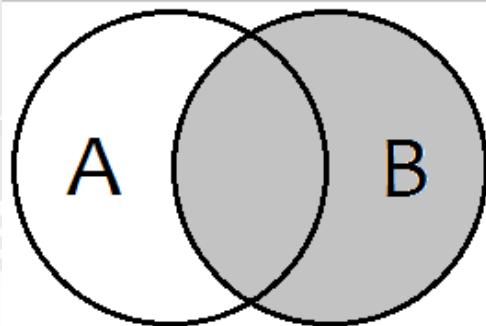
```
SELECT *  
FROM TableA a  
LEFT JOIN TableB b  
ON a.Key = b.Key
```



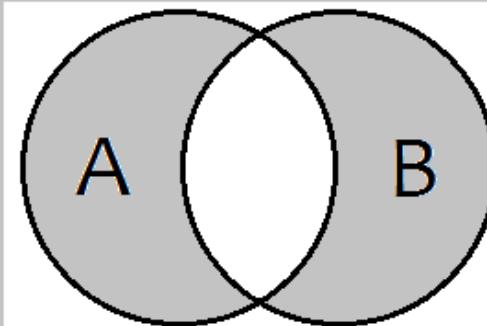
```
SELECT *  
FROM TableA a  
LEFT JOIN TableB b  
ON a.Key = b.Key  
WHERE b.Key IS NULL
```



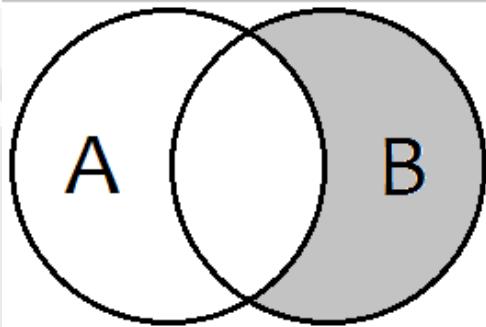
```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.Key = b.Key
```



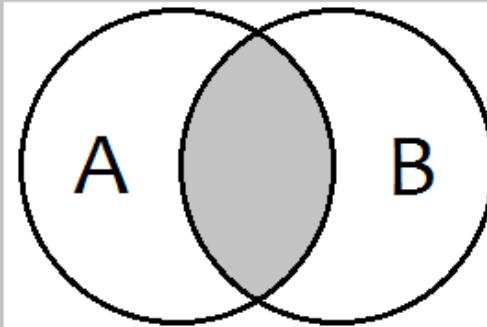
```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.Key = b.Key
```



```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.Key = b.Key  
WHERE a.Key IS NULL  
OR b.Key IS NULL
```



```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.Key = b.Key  
WHERE a.Key IS NULL
```



```
SELECT *  
FROM TableA a  
INNER JOIN TableB b  
ON a.Key = b.Key
```

## (ضرب دکارتی دارای شرط) **INNER JOIN**

این عملگر در واقع همان ضرب دکارتی است که می‌تواند همراه با شرط مورد نظر برای پیوند جداول مورد استفاده قرار بگیرد.

مثال اول:

```
select *  
from employee INNER JOIN department  
on employee.Dep# = department.Dep#
```

توجه: در **INNER JOIN** می‌توان از  $=$ ,  $>=$ ,  $<=$ ,  $>$  و  $<$  استفاده نمود.

**توجه:** در JOIN فقط = مورد استفاده قرار می‌گیرد و استفاده از > < و <= >= مجاز نیست.

**مثال دوم: (شیوه‌سازی)**

```
select *  
from employee, department  
where employee.Dep# = department.Dep#
```

1. **INNER JOIN** - fetches data if present in both the tables.

2. **OUTER JOIN** are of **3** types:

1. **LEFT OUTER JOIN** - fetches data if present in the **left table**.

2. **RIGHT OUTER JOIN** - fetches data if present in the **right table**.

3. **FULL OUTER JOIN** - fetches data if present in either of the **two tables**.

3. **CROSS JOIN**, as the name suggests, does  $[n \times m]$  that joins everything to everything.

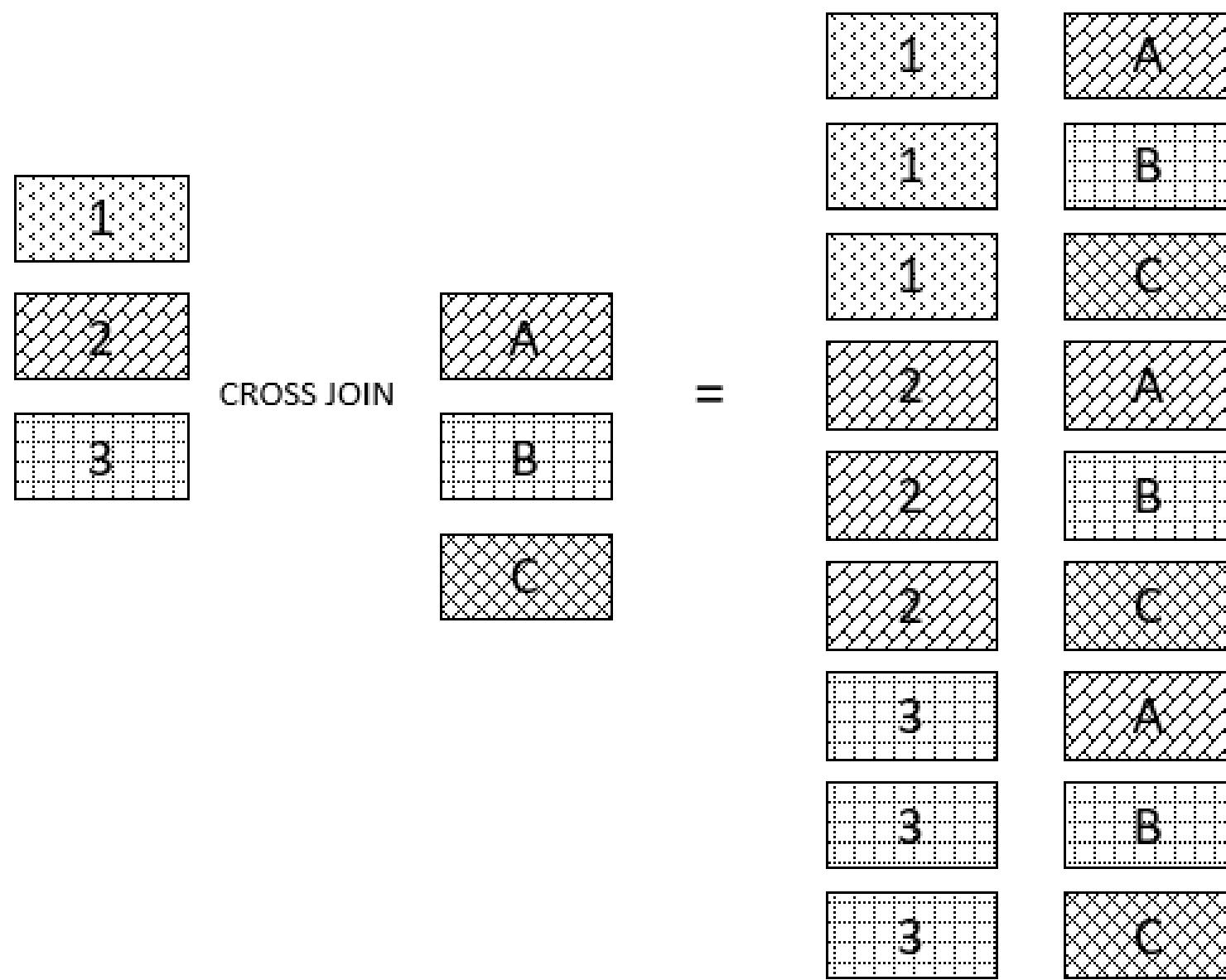
Similar to scenario where we simply lists the tables for joining (in the **FROM** clause of the **SELECT** statement), using commas to separate them.



What is the difference between left join and left outer join?

**Nothing.** `LEFT JOIN` and `LEFT OUTER JOIN` are equivalent.

# Cross join



## CROSS JOIN (ضرب دکارتی)

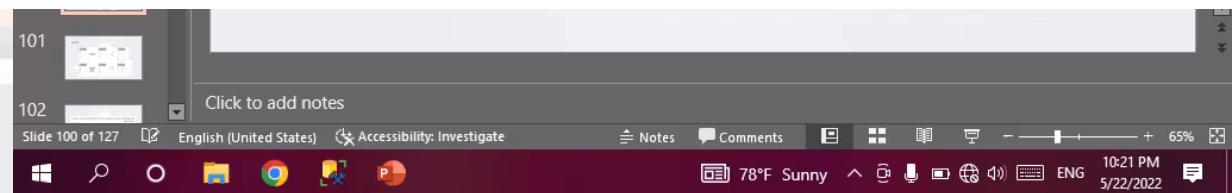
مثال اول:

```
select *  
from employee CROSS JOIN department
```

مثال دوم:

```
select *  
from employee, department
```

توجه: در CROSS JOIN همه سطرهای دو جدول در کنار هم قرار می‌گیرند.

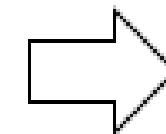


# UNION

<b>id</b>
1
2
3

**UNION**

<b>id</b>
2
3
4

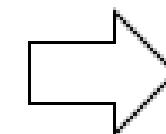


<b>id</b>
1
2
3
4

<b>id</b>
1
2
3

**INNER  
JOIN**

<b>id</b>
2
3
4

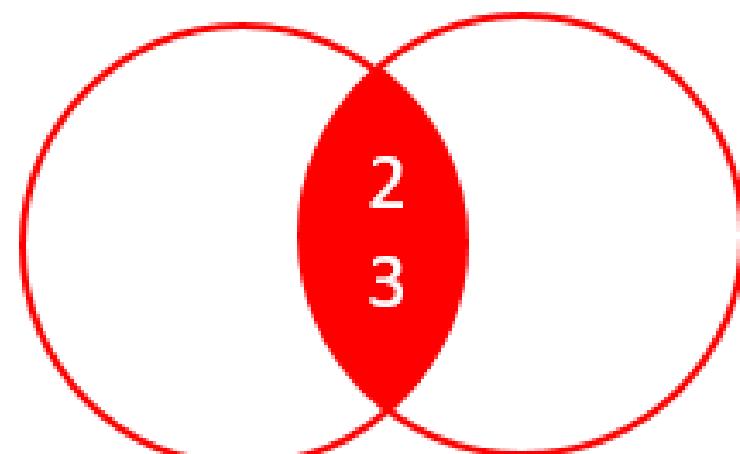
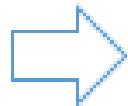
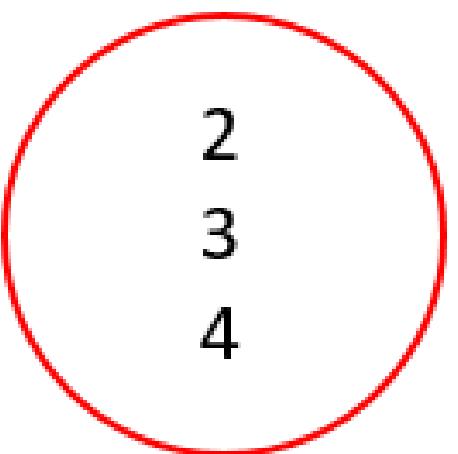
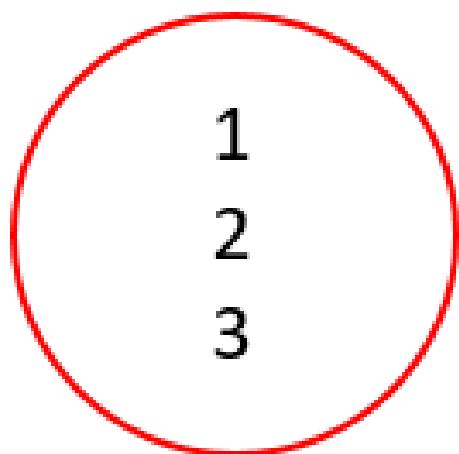


<b>id</b>	<b>id</b>
2	2
3	3

مثال: شماره قطعاتی را بباید که یا وزن آنها بیش از ۱۶ باشد یا توسط S2 تهیه شده باشند یا هر دو شرط را دارا باشند.

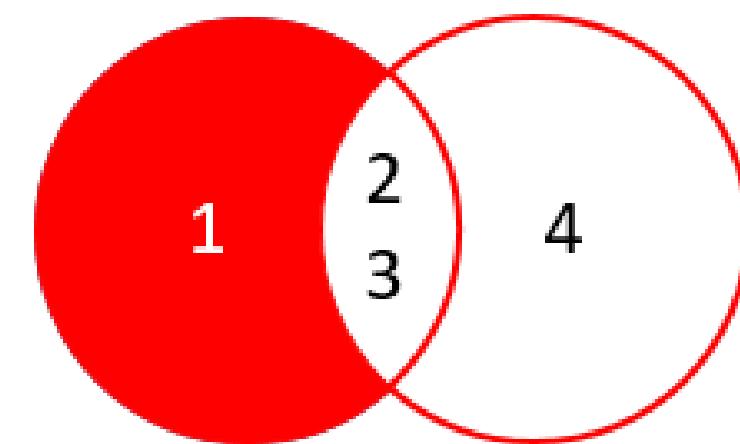
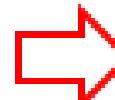
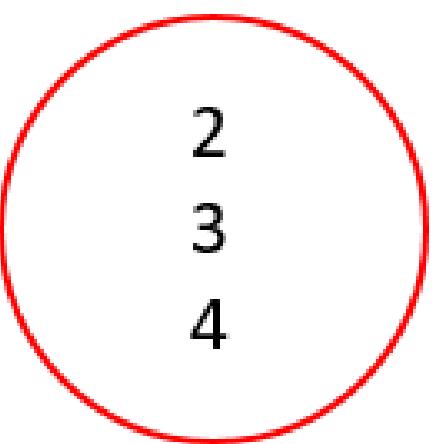
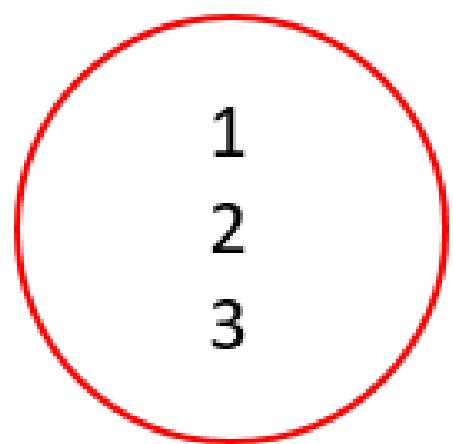
```
SELECT P#
FROM P
WHERE weigh>۱۶
UNION
SELECT P#
FROM SP
WHERE S#=S2
```

# INTERSECT



T1 INTERSECT T2

**EXCEPT**



S#	Sname	City
S1	Sn1	C1
S2	Sn2	C2
S3	Sn3	C2

خالد

S#	P#	QTY
S1	P1	10
S1	P2	20
S2	P1	30

جذور

**مثال اول:** نام تولید کنندگانی که حداقل یک قطعه تولید کرده‌اند:

```
select Sname  
from S
```

where exists (select \*

from SP

where S.S# = SP.S#

خواجہ  $\Rightarrow \frac{\text{Sname}}{\text{Snl}}$

جدول Department و Employee را در نظر بگیرید:

Emp#	EmpName	Salary	Dep#
101	Alavi	2	12
102	Hassani	4	12
103	Hosseini	6	13
104	Sajjadi	8	13
105	Asgari	10	14
106	Akbari	12	14

جدول Employee

Dep#	DepName
12	حسابداری
13	فروش
14	مهندسی

جدول Department

مثال دوم: مشخصات کارمندانی که حقوقشان، از متوسط حقوق، کارمندان بخشنده بیشتر است:

```
select *
from employee empl
where salary > (select AVG (salary)
from employee emp2
where empl.Dep# = emp2.Dep#)
```

۱۸- دستوراتی که به عنوان عکس العمل یک عمل در پایگاه داده به فرم خود کار اجرا می شود  
(مهندسی کامپیوترا- آناد QP نامیده می شود)

Assertion (۲)

Grant ()

Stored procedure (۴)

Trigger (۳)

# index

سیستم‌های پایگاه داده به دلیل سرعت کم دیسک سخت، نمی‌توانند با سرعت بالا کار جستجو و مرتب‌سازی اطلاعات را انجام دهند.

این مشکل به ویژه هنگامی که حجم اطلاعات بسیار زیاد است، باعث غیرقابل استفاده شدن سیستم و افزایش فشار روی سخت‌افزار و افزایش هزینه‌های سخت‌افزاری می‌شود.

ایندکس یک کپی جداگانه از اطلاعات است که به ترتیب دلخواه مرتب شده است. جستجو روی یک جدول مرتب شده با استفاده از روش دودویی بسیار سریع‌تر از پیمایش کلی اطلاعات است.

ایک جدول کپی از اطلاعات است و فضای پایگاه داده را اشغال می‌کند. علاوه بر این، **Index** گذاری کاری پر هزینه است و در نتیجه نمی‌توان آن را روی تمامی جدول‌ها و تمام اطلاعات به انجام رساند.

در عوض این برنامه‌نویس است که می‌بایست تشخیص دهد، روی کدام جدول و کدام ستون‌ها **Index** گذاشته شود. انتخاب دقیق و کارآمد ستون‌ها برای ایندکس گذاری سبب می‌شود تا کار جستجو روی اطلاعات بسیار سریع انجام شود و در نتیجه عملکرد کلی سیستم نیز بهبود پیدا می‌کند.

جستجوها در پایگاه داده همیشه به صورت صریح و توسط برنامه‌نویس به انجام نمی‌رسند. به عبارت دیگر، دستور **WHERE** تنها دستوری نیست که کار جستجو را انجام می‌دهد.

دستورات دیگری مثل **JOIN**‌ها نیز برای یافتن ردیف‌های متناظر می‌بایست کار جستجو را انجام دهند. حتی توجه به این نکته ضروری است که **JOIN**‌ها به دلیل سر و کار داشتن با تعداد زیاد ردیف‌ها، فشار بسیار زیادی به پایگاه داده وارد می‌کنند.

بنابراین هنگامی که از **JOIN**‌ها روی ستون‌های مورد نظر استفاده می‌کنید، لازم است تا با دقت بررسی کنید که آیا ستون‌های مورد مقایسه و جستجو دارای **Index** هستند یا خیر.

# انواع index

B-Tree Index Structure

Root Level

1  
5000

Intermediate Level

1  
2500

5000  
7500

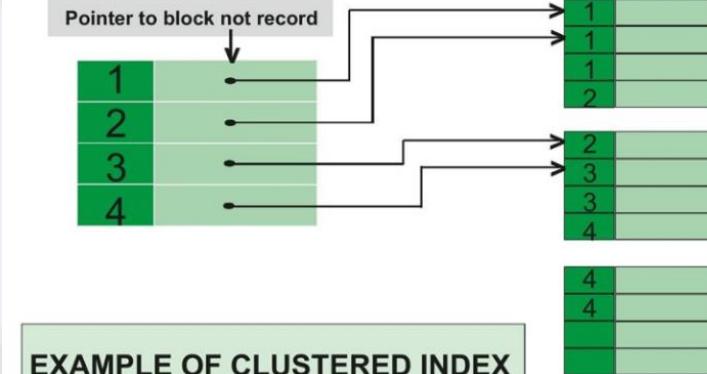
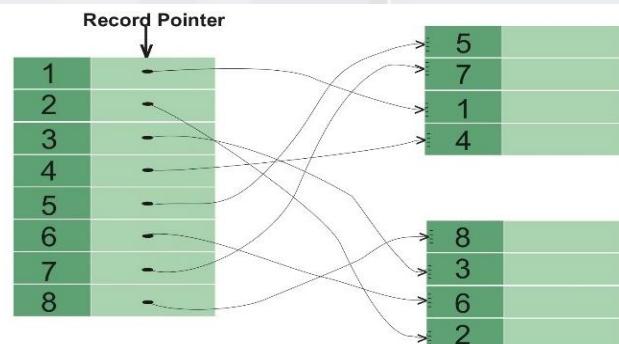
Leaf Level

1  
...  
2499

2500  
...  
4999

5000  
...  
7499

7500  
...  
10000



# مقایسه‌ی روش‌های Clustered و Nonclustered

به اختصار و برای جمع‌بندی چند تفاوت بین دو روش خوش‌های و غیرخوش‌های را بررسی می‌کنیم :

- ایندکس‌های Nonclustered اختیاری و به نظر طراح پایگاه داده وابسته است ولی ایندکس‌های clustered اجباری هستند و به صورت اتوماتیک روی جدول در پایگاه داده اعمال می‌شود.
- برای خواندن اطلاعات روش nonclustered و برای ویرایش و به روزرسانی داده‌ها روش clustered کارآمدتر است.
- در یک جدول می‌توان چند ایندکس clustered داشت ولی ایندکس nonclustered تنها یکی قابل استفاده است.
- روش clustered اطلاعات را بر اساس کلید اصلی مرتب می‌کند ولی روش nonclustered در نظم داده‌ها دخالتی ندارد.
- سرعت دسترسی به اطلاعات در ایندکس‌های clustered بیشتر است و در مقابل ایندکس‌های nonclustered سرعت ویرایش و ذخیره سازی اطلاعات بیشتری دارند.

# view

- view ها برای مقاصد امنیتی در پایگاه داده مورد استفاده قرار میگیرند ، و view ها کاربر رو در دسترسی و دیدن سطرها و ستون ها محدود می کند که این بدین معنی است که ، ما بوسیله view ها میتوانیم محدودیت هایی رو برای دسترسی کابرها متفاوت ، به سطر ها و ستون های خاصی ، اعمال کنیم . view ها فقط داده هایی که در Qurey زیر ذکر شدند ، را نمایش میدهد . بنابراین فقط داده هایی که توسط Query برگردانده میشود را نمایش میدهد که این امر در زمان ساخت view تعریف میشود . ما میتوانیم جداول استفاده شده توسط view رو DROP کنیم ، اگر این جداول را حذف کنیم view در پایگاه داده باقی میماند.

# function

جایگزین های مناسب دیدها : می توان از روال ها و مخصوصاً توابع به عنوان جایگزین های مناسبی برای دیدها استفاده کرد. یکی از بهترین دلایل استفاده از توابع به جای دیدها این است که توابع مانند دیدها به یک دستور `Select` محدود نیستند و می توانند هر تعداد دستور را اجرا کنند. مجوزهای دسترسی که در دیدها مطرح می شوند در توابع هم قابل پیاده سازی هستند .

توابع مورد استفاده در SQL به دو بخش تقسیم می شوند :

۱- توابع سیستمی : - توابع رشته ای - توابع تاریخ - توابع ریاضی - سایر توابع

۲- توابعی که توسط کاربر تعریف می شوند UDF

این توابع به سه حالت می توانند نوشته شوند:

توابع اسکالر (scalar)

- توابع جدولی تک خطی (inline)

- توابع جدولی چند دستوری (multi-statement)

مثال: با استفاده از این روش تابعی می نویسیم که شماره یک دانشجو و شماره یک ترم وی را در یافت و در صورتی که معدل کل دانشجو بالاتر از ۱۲ است شماره درس هایی را که دانشجو در آن ترم ثبت نام کرده است را برابر می گرداند. در غیر این صورت حدول تهی برگرداند.

```
Create function Fn_crsnames(@S# int ,@TrmNo CHAR(4))
```

```
Returns @crsnames table(c# CHAR(7) not null)
```

```
As
```

```
Begin
```

```
Declare @mygpa dec(5,2)
```

```
select @mygpa=Gpa from STD where S# = @S#
```

```
If @mygpa>12
```

```
Begin
```

```
insert @crsnames
```

```
Select c# from REG
```

```
Where ( (S# = @S#) and (Trmno = @Trmno))
```

```
End
```

```
return
```

```
End
```

```
Select * from Fn_crsnames(8008093,'3802')
```

تابع زیر با دریافت شماره درس و شماره ترم لیست نام و جنسیت دانشجویان آن درس را در ترم مذکور بر می‌گرداند.

```
Create function fn_myreg(@C# char(7),@TrmNo char(4))
```

Returns table

As

```
Return (select STD.Name,STD.Family,CODEFILE.[Desc] From STD,CODEFILE
```

```
Where CODEFILE.Field = 'sex'
```

And

```
CODEFILE.Type = STD.Sex
```

And

```
exists(select * from REG
```

```
Where (REG.C# = @C#
```

```
and STD.S#=REG.S#
```

```
and REG.Trmno=@Trmno) )
```

```
)
```

```
Select * from fn_myreg (8006534,'3821')
```

# CURSOR

- cursorها به ما کمک می‌دهند که تا بتوانیم سطر به سطر روی جداول حرکت کنیم و اگر می‌خواهیم مرحله به مرحله کدهایمان را اجرا کنیم. cursorها در واقع یک اشاره گر هستند که کنار سطرهای یک جدول یا یک محتوا قرار می‌گیرند و یکی یکی روی آن‌ها می‌توانند حرکت کنند و روی هر سطربالوی که قرار بگیرند می‌توانیم به محتویات آن سطر دسترسی داشته باشیم. دو مدل scroll cursor و cursor داریم :
- در این مثال می‌خواهیم روی سطرهای جدول customers حرکت کنیم. برای اینکار سه متغیر برای فیلدهایی که می‌خواهیم اطلاعات آن را واکشی کنیم تعریف می‌کنیم و بعد یک cursor با نام CR تعریف می‌کنیم و با دستور for مشخص می‌کنیم که روی چه جدول و فیلدهایی می‌خواهیم حرکت کنیم. بعد از مشخص کردن جدول باید cursor را open و در انتهای close کنیم و عملیاتی را که می‌خواهیم انجام دهیم بین دستور open و close قرار می‌دهیم و در انتهای با دستور Deallocate اشاره گری را که تعریف کردہ‌ایم از حافظه پاک می‌کنیم. با دستور fetch هم اطلاعات سطرهای جدول را واکشی می‌کنیم.

```
Declare @fname nvarchar(100),@lname nvarchar(100),@stateCode tinyint  
Declare cr Cursor  
For(  
    Select [FName],[LName],[StateCode]  
    From Customers  
)  
Open cr  
    fetch next from cr into @fname,@lname,@stateCode  
    Print(@fname + ' '+ @lname + ' '+Cast(@stateCode as nvarchar(20)) )  
Close cr  
Deallocate cr
```

# Store procedure

- رویه ها نیز مانند توابع ابزارهای مناسبی برای دسته بندی دستورات پر کاربرد هستند. رویه ها حتی می توانند مقادیر خروجی داشته باشند ولی تفاوت عمده آن با تابع در نحوه احضار آن ها برای گرفتن مقدار بازگشتی است . احضار توابع صریح است بدین معنی که در احضار توابع می توان مقداری را مساوی با تابع قرار داد و بعد از احضار متغیر با مقدار بازگشتی تابع مقدار دهی می شود. ولی برای گرفتن خروجی از رویه ها باید یک یا چندین پارامتر را به عنوان مقادیر خروجی معرفی کنیم و رویه با مقدار گذاری و تغییر این پارامتر ها مقدار خروجی را SET می کند.
- می توان از عبارت **with recompile** در جایی که یک رویه احضار می شود استفاده کرد. در این صورت رویه دوباره کامپایل و بهینه سازی می شود .

می خواهیم رویه ای بنویسیم که دروس یک دانشجو را که بدون رعایت هم نیازی در ترم جدید ثبت نام کرده ، حذف نماید .

```
create procedure pr_delete_coreq_reg (@st# int)
as
begin
delete from reg where
    reg.s# = @st#
    and
    reg.c# in (select coreq .c# from coreq
                where
                    coreq.c# = reg.c#
                    and
                    reg.s# = @st#
                    and
                    coreq.cp# not in (select reg. c# from reg
                                    where
                                        reg.s# = @st#    ))
```

```
execute pr_delete_coreq_reg 8004367
```

# CTE

- تکنولوژی CTE از نسخه SQL Server 2005 رسمیت یافته است و شامل یک result set موقتی [۱] است که دارای نام مشخص بوده و می‌توان از آن در دستورات SELECT, INSERT, UPDATE, DELETE استفاده کرد. همچنین از CTE می‌توان در دستور MERGE و دستور SELECT مربوط به آن استفاده کرد. در نسخه SQL Server 2008 نیز امکان استفاده از CTE در دستور VIEW فراهم شده است.  
در SQL Server از دو نوع CTE بازگشتی و غیر بازگشتی پشتیبانی می‌شود.
- یکی از کاربردهای CTE ایجاد قطعات کوچکی است که امکان استفاده مجدد را به شما داده و می‌تواند سبب خواناتر شدن کدهای پیچیده شود. یکی دیگر از کاربردهای CTE آنجایی است که شما نمی‌خواهید یک شی View ی عومومی تعریف کنید و در عین حال می‌خواهید از مزایای View‌ها بهره‌مند شوید.  
و همچنین از کاربردهای دیگر CTE تعریف جدول موقت و استفاده از آن جدول به صورت همزمان در یک دستور است.

# Sequence

• یک دنباله یا مجموعه‌ای از اعداد است که پشت سرهم ایجاد شده و به یک ستون از جدول Sequence تخصیص داده می‌شود. sequence در واقع عملکردی مشابه ستون شناسه (identity column) دارد، با این تفاوت که در صورت نیاز به یک identity باایستی آن را در یک ستون از جدول مورد نظر ایجاد کنید و سپس زمانی که به آن identity در ستون جدول دیگری نیاز پیدا می‌کنید، باز باید identity را در ستون جدول غیر ایجاد نمایید. در حالی که sequence یک شی برنامه نویسی شده، مانند یک تابع، می‌باشد که شما در سطح پایگاه داده ایجاد می‌کنید و بعد آن را به هر جدولی که می‌خواهید اعمال می‌نمایید.

# Trigger

- نوعی رویه ذخیره شده است که پس از یک رویداد اجرا می شود ، برخلاف Stored Procedure که حتما باید فراخوانی شود Trigger ها قابل فراخوانی نبوده و به صورت خودکار در واکنش به اعمال Insert , Update , Delete قبل یا بعد از تراکنش اجرا می شوند .

- واضح است که چنین رویه برخلاف SP نمیتواند پارامتر ورود و خروجی بپذیرد و اطلاعات خود را از رکورد مورد نظر (رکوردي که بر روی آن صورت گرفته که اصطلاحا Inserted , Updated و Deleted خوانده می شود ميگيرد).

- مهمترین کاربر Trigger ها به شرح زیر است :
- جلوگیری از ورود داده های نا معتبر
- لاغ نمودن تراکنش ها
- حذف یا به روز رسانی رکورد ها در جداول مرتبط بالا بردن کارایی سیستم
- تغییر و جایگزینی یک تراکنش

# DCL

## Grant •

این دستور برای اعطای مجوز به کاربران (یا نقش های کاربری) برای انجام کارهای مشخص بکار می رود. شکل کلی دستور به صورت زیر می باشد.

GRANT [امتیاز] ON [object] TO [کاربر]

## Revoke •

از این دستور برای لغو کردن یا سلب مجوزهای و امتیازات اعطای شده به کاربر استفاده می شود و میتوانیم یکسری از مجوزهای دسترسی را از کاربران بگیریم.

REVOKE [اختیار دادن برای] [مجوز] ON [object] FROM [user] [CASCADE]

## DENY •

• DENY [مجوز] ON [object] TO [کاربر]

برای گرفتن دسترسی از یک نفر در یک گروه

# مجوز ها:

Privilege	Description
<b>SELECT</b>	<i>select statement on tables</i>
<b>INSERT</b>	<i>insert statement on the table</i>
<b>DELETE</b>	<i>delete statement on the table</i>
<b>INDEX</b>	<i>Create an index on an existing table</i>
<b>CREATE</b>	<i>Create table statements</i>
<b>ALTER</b>	<i>Ability to perform ALTER TABLE to change the table definition</i>
<b>DROP</b>	<i>Drop table statements</i>
<b>ALL</b>	<i>Grant all permissions except GRANT OPTION</i>
<b>UPDATE</b>	<i>Update statements on the table</i>
<b>GRANT</b>	<i>Allows to grant the privilege that</i>

# TCL

تراکنش ها عملیاتی هستند که باید یا همه ی دستورات باهم اجرا شوند و یا هیچ کدام اجرا نشود. هر تراکنش باید ۴ اصل ACID را داشته باشد:

1. Atomicity: اگر در اجرا جایی مشکلی پیش آمد تمامی تغییرات به قبل باز خواهند گشت.
2. Consistency: (ثبات و دوام) این خاصیت تضمین میکند که هر تراکنش دیتابیس را از یک وضعیت صحیح و معتبر به وضعیت معتبر دیگری می برد. یعنی هر دیتابیس نوشه میشود باستی بر طبق قواعد و constraints ها و تریگر ها معتبر باشد.
3. Isolation: تغییرات ایجاد شده توسط یک تراکنش باستی از تغییرات ناشی از تراکنش های همزمان دیگر ایزوله و مجزا باشد.
4. Durability: پایداری و دوام. هنگامی که یک تراکنش کامل شد تغییرات ناشی از آن بطور دائمی در دیتابیس نوشته میشوند . یعنی هنگامی که یک تراکنش بطور کامل انجام شد. حتی از اس کیوال سرور دچار مشکل شود و سرویس restart شود و اطمینان داریم که تغییرات ناشی از تراکنش در اطلاعات جداول لحاظ خواهند شد .

# انواع تراکنش ها:

- Implicit Transactions

هنگامی که کانکشن به اس کیوال سرور به این مد باشد اس کیوال سرور بطور اتوماتیک پس از اتمام هر تراکنش (commit , rollback) تراکنش جدیدی را شروع خواهد کرد . شما شروع تراکنش را معین نخواهید کرد ولی می توانید commit شده و یا rollback شدن آنرا تعیین کنید . بنابراین یک زنجیره پیوسته از تراکنش ها خواهیم داشت.

- Explicit Transactions

بطور صريح شروع و پایان آنرا مشخص میکنیم .

# TCL

از دستورات زیر برای کنترل transaction استفاده می شود :

- COMMIT - برای ذخیره تغییرات استفاده می شود.
- ROLLBACK - برای بازگرداندن تغییرات مورد استفاده می گردد.
- SAVEPOINT - ایجاد نقاط درون گروهی transaction در ROLLBACK.

دستورات TCL فقط با دستورهای INSERT, UPDATE و DELETE استفاده می شود.

