به نام آنکه جان را فکرت آموخت

معرفی درس:

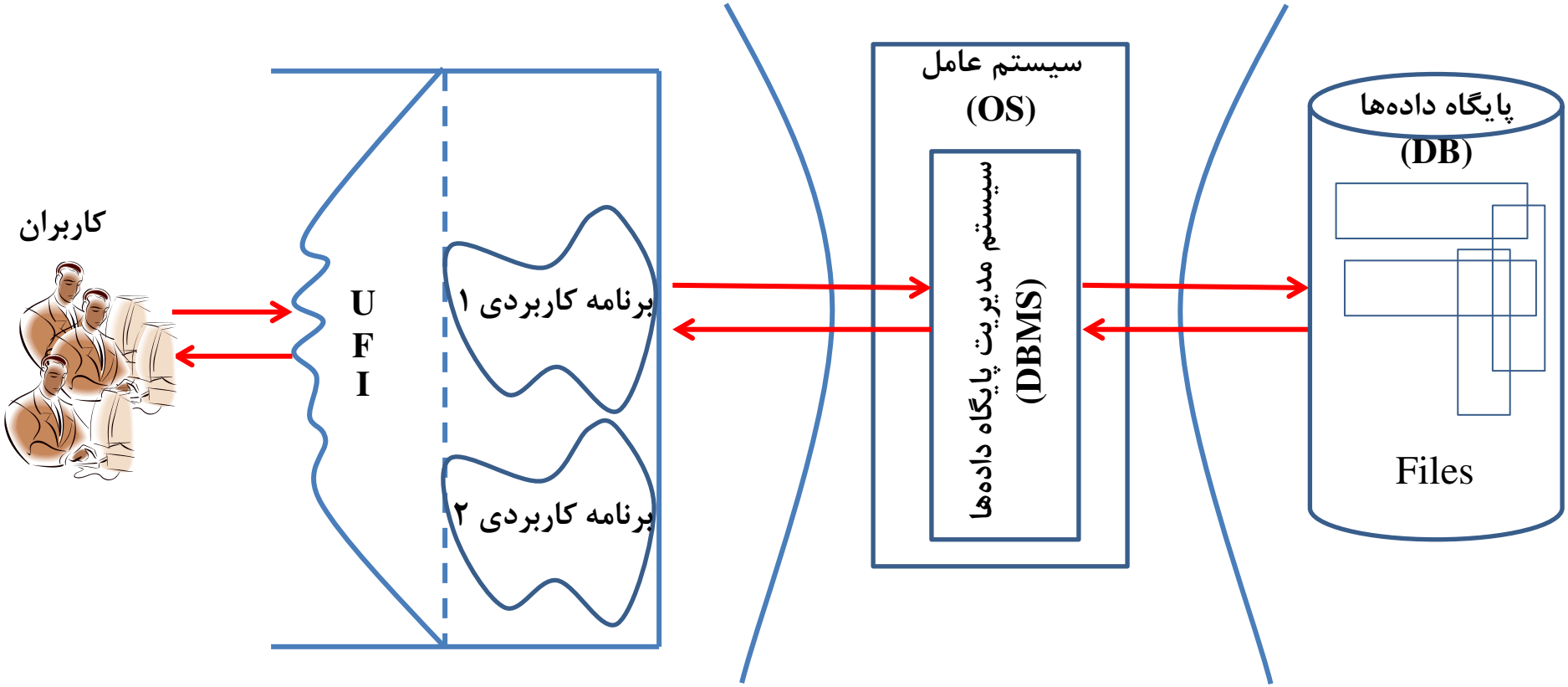
طراحی پایگاه داده‌ها (۴۰۳۸۴)

مرتضی امینی

نیمسال دوم ۹۴-۹۵

- نیازمند توسعه سیستم‌های اطلاعاتی یا برنامه‌های کاربردی برای استفاده از اطلاعات
- وجود حجم زیادی از داده‌ها و اطلاعات ذخیره شده ← پایگاه داده‌ها
- نیازمند سیستم واسطی برای ذخیره، جستجو، بازیابی و به‌روزرسانی اطلاعات ← سیستم مدیریت

پایگاه داده‌ها (سمپاد – DBMS)


کاربران
U
F
I
برنامه کاربردی ۱
برنامه کاربردی ۲
سیستم عامل
(OS)
سیستم مدیریت پایگاه داده‌ها
(DBMS)
پایگاه داده‌ها
(DB)
Files

- **سوال:** برای تولید یک سیستم پایگاهی در یک محیط عملیاتی چه باید کرد؟

امکانات موردنیاز در ایجاد پایگاه داده‌ها

- **سوال:** در مدلسازی، طراحی و پیاده‌سازی پایگاه داده‌ها چه امکاناتی نیاز است؟

یک روش و زبان مدلسازی داده‌ها

روش نمودار روابط موجودیت‌ها (ER)

یک مدل داده‌ای برای طراحی منطقی

مدل رابطه‌ای و جدولی

یک زبان برای انجام عملیات

جبر رابطه‌ای و حساب رابطه‌ای

یک زبان استاندارد برای تعریف، کنترل و انجام عملیات پایگاهی

زبان SQL

یک سیستم مدیریت پایگاه داده‌ها

...

سرفصل‌های درس

۱- کلیات

- تعریف پایگاه داده‌ها، مشی فایلینگ و مشی پایگاهی، عناصر محیط پایگاه داده، انواع معماری سیستم پایگاهی

۲- مدلسازی معنایی داده‌ها با روش ER و EER

- نمودار ER و اجزای آن، انواع دام‌ها، تکنیک‌های تخصیص، تعمیم، تجزیه، ترکیب و تجمیع، ویژگی‌های روش مدلسازی معنایی

۳- آشنایی با ساختار داده‌ای جدولی (رابطه‌ای)

- ساختار جدولی و اجزای آن، پایگاه داده جدولی و طراحی آن، زبان پایگاه داده جدولی (SQL)

۴- معماری سه سطحی پایگاه (پیشنهادی ANSI)

- دید (نمای) ادراکی، دید داخلی، دید خارجی، تبدیلات بین سطوح، عملیات از دید خارجی و مشکلات آن، استقلال داده‌ای فیزیکی و منطقی

۵- سیستم مدیریت پایگاه داده‌ها (DBMS)

- ریزفعالیت‌های ایجاد سیستم پایگاهی، مزایا و معایب تکنولوژی پایگاهی، وظایف، اجزا و رده‌بندی سمپادها، تیم مدیریت پایگاه داده‌ها (DBA)

سرفصل‌های درس (ادامه)

۶- مفاهیم اساسی مدل داده رابطه‌ای

- رابطه و مفاهیم مربوطه، میدان (دامنه)، انواع رابطه، رابطه‌های نرمال و غیرنرمال، انواع کلید در مدل رابطه‌ای

۷- اصول طراحی پایگاه داده‌های رابطه‌ای به روش بالا به پایین

- تکنیک‌های تبدیل مدلسازی معنایی به طراحی منطقی

۸- اصول طراحی پایگاه داده‌های رابطه‌ای به روش سنتز

- روش سنتز (نرمال‌ترسازی رابطه‌ها)، مفاهیمی از تئوری وابستگی، شرح فرم‌های نرمال، تجزیه مطلوب

۹- جامعیت در مدل رابطه‌ای

- قواعد کاربری، مکانیزم‌های اِعمال قواعد جامعیت کاربری، قواعد جامعیت موجودیتی و ارجاعی (C1 و C2)

۱۰- عملیات در پایگاه رابطه‌ای

- جبر رابطه‌ای، حساب رابطه‌ای

نکته: یادگیری زبان SQL به عهده دانشجو است.

(در کلاس به شکل مختصر معرفی می‌شود)

مراجع درس

- **مفاهیم بنیادی پایگاه داده‌ها** نوشته سیدمحمدتقی روحانی رانکوهی، ویراست چهارم، ۱۳۹۰.
- **An Introduction to Database Systems**, By C.J. Date, 8th Edition, 2003.
- **Fundamental of Database Systems**, By R. Elmasri, 7th Edition, 2015.
- **Database Systems**, By T. Connolly and C. Begg, 6th Edition, 2014.
- **Database Management Systems**, By R. Ramakrishnan and J. Gehrke, 3nd Edition, 2002.
- **Database System Concepts**, By A. Silberschartz, H.F. Korth and S. Sudarshan, 6th Edition, 2010.

شیوه ارزیابی

- میان‌ترم
- پایان‌ترم
- تمرین‌های نظری
- پروژه عملی – مستمر در طی ترم
- کوئیزهای موردی و فعالیتهای کلاسی
- نمرات تشویقی – ارائه مطلب، کار اضافه، حل تمرینهای اضافی، ...

نحوه توزیع نمرات در پایان ترم مشخص می‌شود.

انتظار می‌رود دانشجویان در همه موارد حداکثر تلاش خود را داشته باشند.

پرسش و پاسخ . . .

amini@sharif.edu

به نام آنکه جان را فکرت آموخت

بخش اول :
مقدمه

مرتضی امینی

نیمسال دوم ۹۴-۹۵

(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)

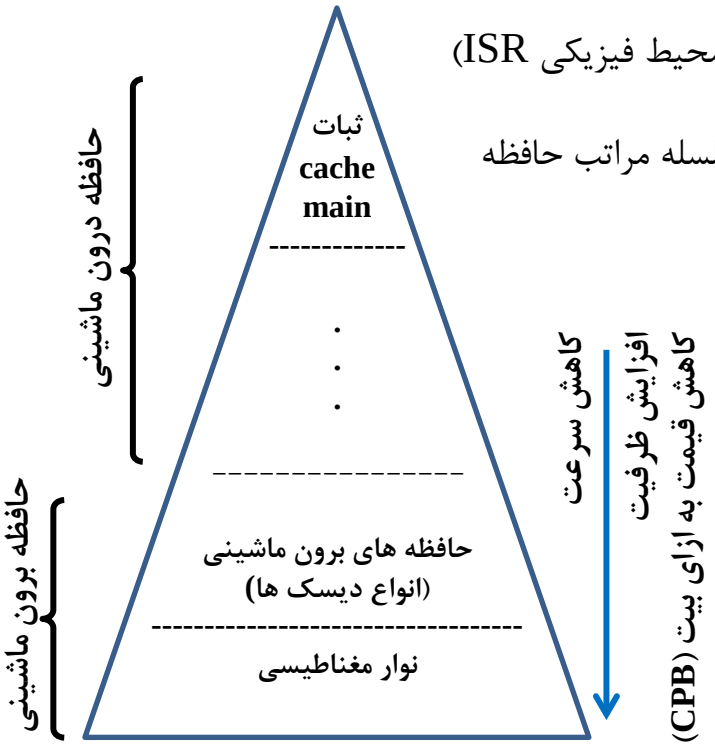
- هر سیستم نرم‌افزاری از مجموعه ای از داده‌های ذخیره شده ممکن است استفاده کند.

در قالب تعدادی فایل (محیط فیزیکی ISR)

در کجا؟ ← در یک سلسله مراتب حافظه

به لحاظ ساختاری

- ساختمند (structured) — well-formatted است، فرمت ثابت و از پیش تعیین شده دارد.
- نیم ساختمند (semi-structured)
- ناساختمند (un-structured)



کنکاوی: آیا نیاز به تحمیل یک ساختار در اینها داریم؟ آیا واقعا داده ناساختمند داریم؟

یادآوری

انواع سیستم نرم‌افزاری:

- بنیادی یا پایه (سیستم‌های عامل)
- نیمه بنیادی (DBMS، DMS، کامپایلرها، اسمبلرها، و ...)
- کاربردی (برنامه‌های کاربردی)
- ابزاری: انواع toolها

Data Management System

Database Management System

محیط فیزیکی «ذ.ب.ا»

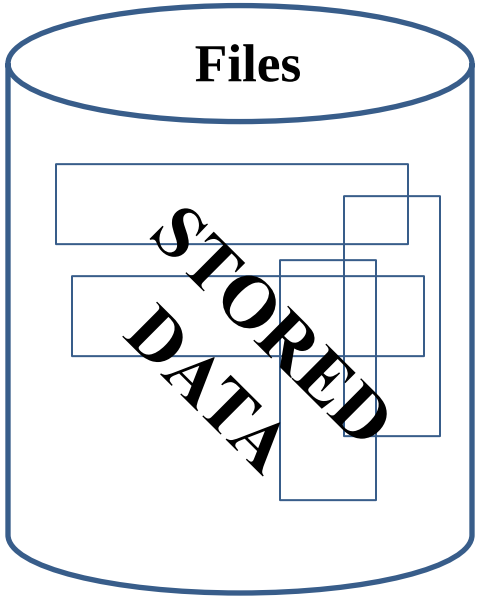
کنجکاوی: دلایل استفاده از این سلسله مراتب حافظه چیست؟

کنجکاوی: چه داده ای، برای چه مدتی، در کدامیک از مراتب سلسله مذکور قرار می‌گیرد؟

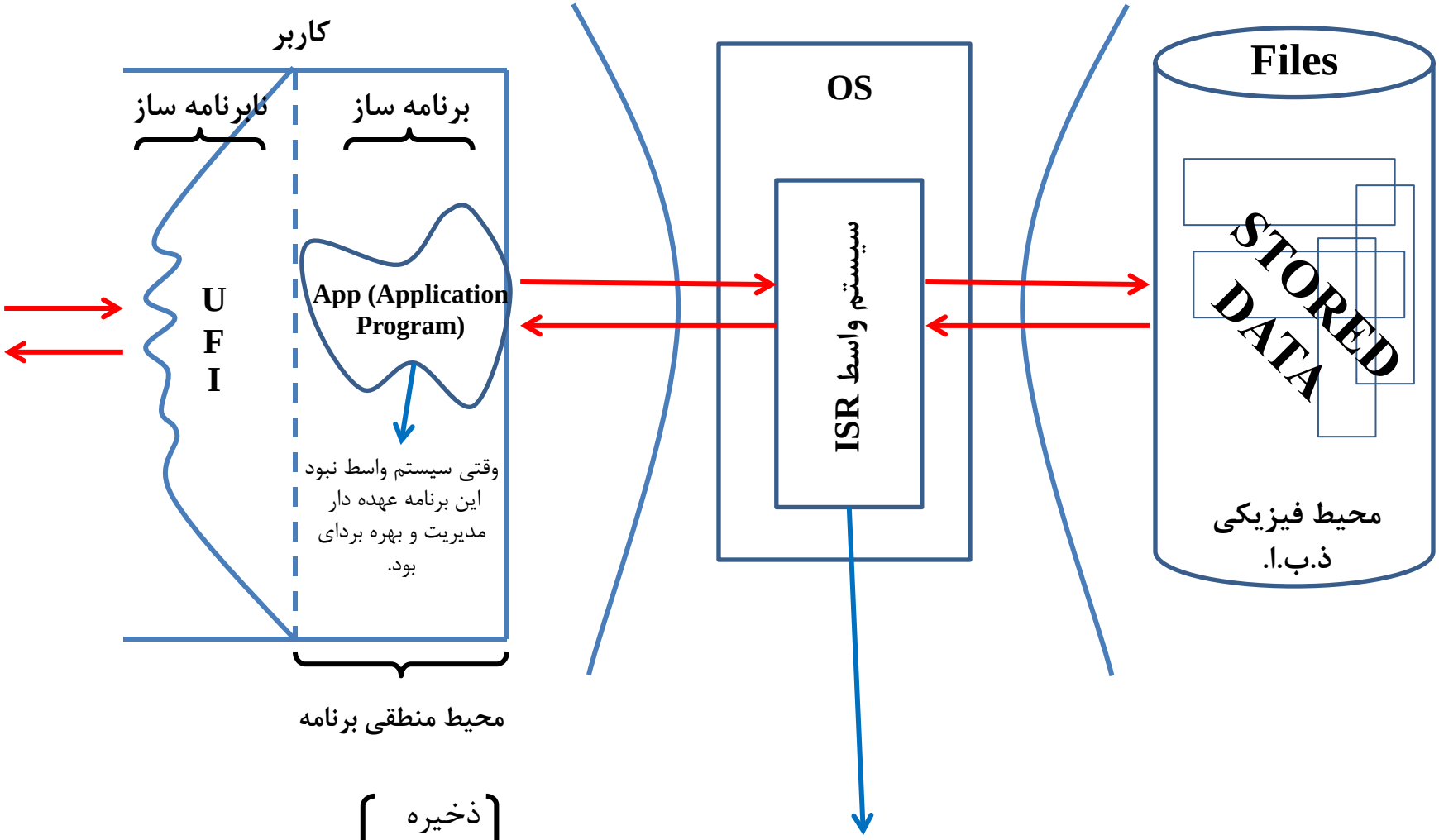
کنجکاوی: خصوصیات عمومی فایل‌ها چیست؟

- **محیط فیزیکی "ذ.ب.ا." (ذخیره و بازیابی اطلاعات) یا ISR (Information Storage and Retrieval)**

ISR: باید { ایجاد، مدیریت، بهره برداری } شود. ← نیاز به یک سیستم واسط ذ.ب.ا داریم.



سیستم واسط ذ.ب.ا.



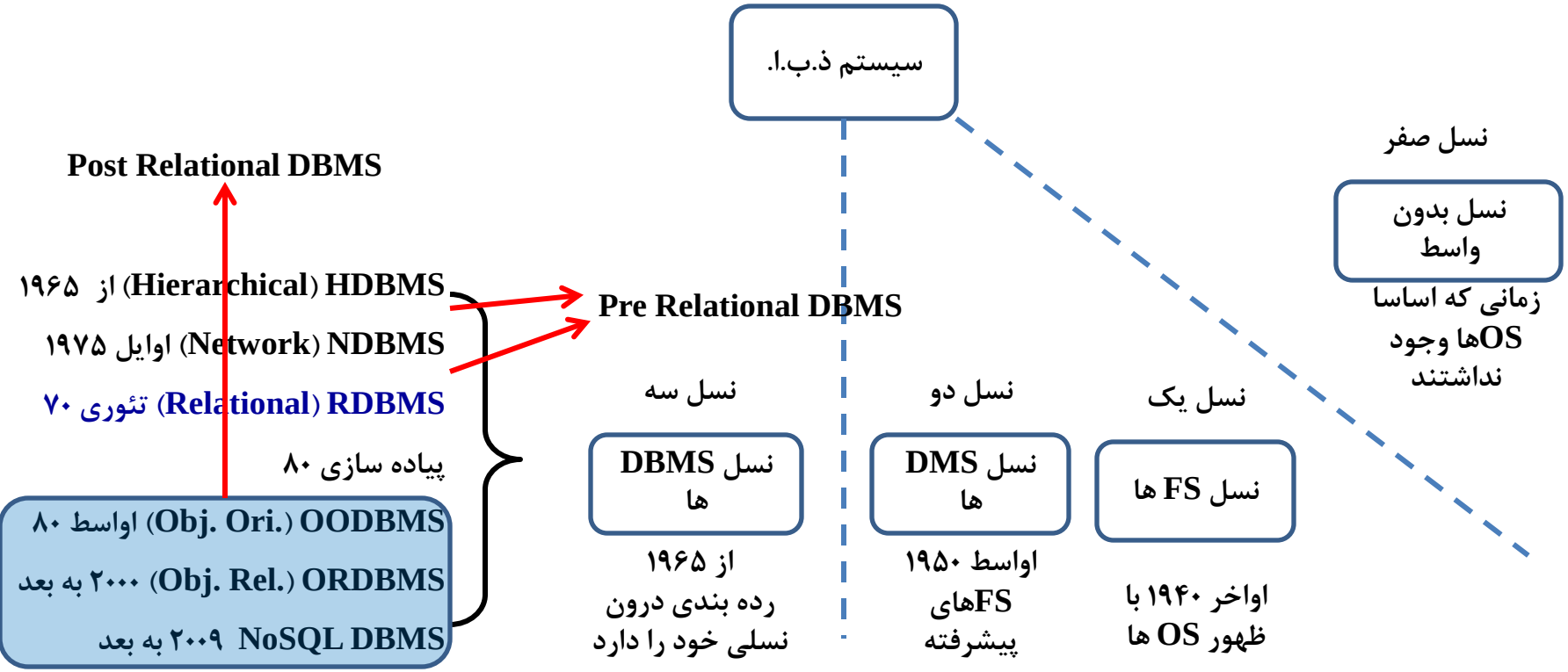
این سیستم این امکان را می‌دهد تا کاربر داده‌های خود را { ذخیره / بازیابی / پردازش } کند.

...

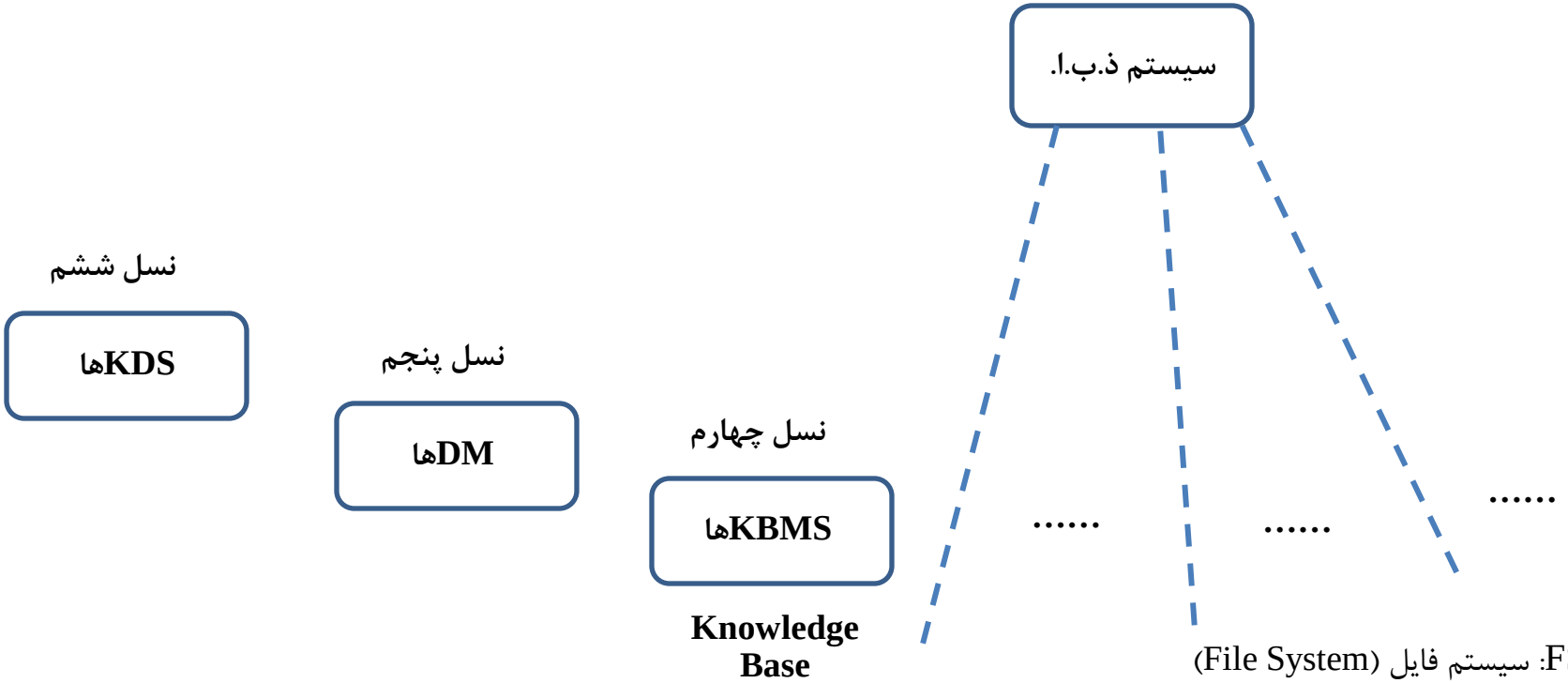
سیر تحول سیستم ذ.ب.ا. (۱)

کنجکاوی: رده بندی از مفهوم کاربر ارایه کنید؟ به بیان دیگر گونه های دیگر کاربر کدامند؟

- سیستم واسط "ISR" سیر تحول خاص خود را دارد :
  - ۶ نسل تکنولوژیک قابل بازیابی است (به طور کلی) [دیدگاه نرم‌افزاری]



سیر تحول سیستم ذ.ب.ا. (۲)



- FS: سیستم فایل (File System)
- DMS: سیستم مدیریت داده‌ها (Data Management System)
- DBMS: سیستم مدیریت پایگاه داده‌ها (DataBase Management System)
- KBMS: سیستم مدیریت پایگاه دانش (Knowledge Base Management System)
- DM: سیستم داده‌کاوی (Data Mining System)
- KDS: سیستم کشف دانش (Knowledge Discovery System)

سیر تحول سیستم ذ.ب.ا. (۳)

- در این نسل‌بندی، نسل بعدی نسل قبلی را منسوخ نمی‌کند. نسل بعدی نسل قبلی را تکمیل می‌کند و از آن استفاده می‌کند.
- انواع نیازهای پردازشی، کنترلی، و عملیاتی سبب ایجاد نسل‌های سیستم «ذ.ب.ا.» شد.

- داده (Data)
  - تعریف اول ANSI: نمایش بوده‌ها، پدیده‌ها، مفاهیم یا شناخته‌ها به طرزی صوری و مناسب برای برقراری ارتباط، تفسیر یا پردازش توسط انسان یا هر امکان خودکار
  - تعریف دوم ANSI: هر نمایشی اعم از کاراکتری (نویسه‌ای) یا کمیت‌های قیاسی که معنایی به آن قابل انتساب باشد (توسط انسان یا یک مکانیسم خودکار)
- اطلاع (Information)
  - تعریف دقیق و جامعی از مفهوم اطلاع وجود ندارد.
  - تعریف اول [LIPS92]: اطلاع، داده پردازش شده است.
  - تعریف دوم [روحا ۷۸-الف]: معنایی که انسان به داده منتسب می‌کند، از طریق قراردادهای شناخته شده‌ای که در نمایش داده به کار می‌روند.
  - برخی داده را همان مقدار واقعا ذخیره شده و اطلاع را معنای آن می‌دانند. بنابراین اطلاع دارای خاصیت اطلاع‌دهندگی و ارتباط‌دهندگی است، در حالیکه داده مجرد این خاصیت را ندارد.

دانش (Knowledge)

- تعریف [FROS87]: دانش عبارت است از نمایش نمادین جنبه‌هایی از بخشی از جهان واقع (جهان موردنظر یا محیط مطرح)
  - مثال: شنبه هوا بارانی است. حسن فرزند علی است.
- تعریف دوم [روحا ۹۱]: دانش منطقی نوعی شناخت است که از یک مجموعه از اطلاعات بر اساس یک مجموعه از قواعد مشخص، معمولا با روش استقراء حاصل می‌شود. حصول این شناخت می‌تواند توسط انسان یا یک سیستم خودکار انجام شود.

تعریف پایگاه داده:

Persistent Integrated Interconnected Redundancy

- مجموعه‌ای است از داده‌های ذخیره شده، پایا ، مجتمع ، به هم مرتبط ، و حتی الامکان فاقد افزونگی، (دارای معماری خاصّ خود، مبتنی بر یک مدل داده ای مشخص) ، تحت کنترل یک سیستم متمرکز، مورد استفاده یک یا چند کاربر در یک سازمان (در یک محیط) به طور اشتراکی و همروند.

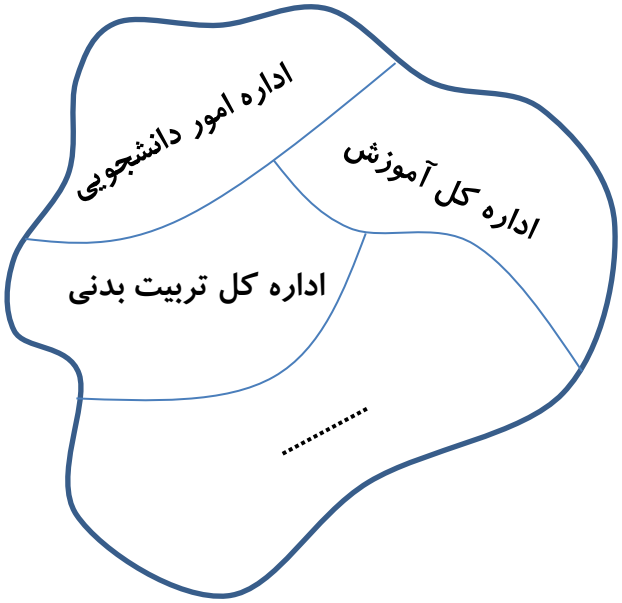
Data Model Shared Concurrent Centralized System Multi User

- مثال کاربردی

- محیط عملیاتی: دانشگاه

بخشی از جهان واقعی که قصد ایجاد سیستم برای آن را داریم.

Micro Real World (خرد جهان واقع)
Mini World
Universe of Discourse (جهان مطرح)



- نکته: هر محیط از تعدادی زیر محیط تشکیل شده است.
- در هر محیط مجموعه‌ای از نوع موجودیت‌ها وجود دارند که نیازهای کاربران ناظر به آنهاست (یعنی داده‌ای پردازشی به داده‌هایی در مورد آنها نیاز دارند).

- **نکته:** زیرمحیط های یک محیط معمولا با هم اشتراک دارند در نوع موجودیت‌ها (Entity Type یا Object Type)
  - مثال: در محیط دانشگاه دانشجو، استاد، درس، کلاس، و ...
  - مثال: نوع موجودیت دانشجو در هر سه زیر محیط مطرح است.

- **مسئله (خواسته): ایجاد سیستم(های) کاربردی برای این زیر محیط‌ها**
  - برای این منظور در اساس دو مَشی-روش (approach) وجود دارد.
    - مشی فایلینگ [سنتی یا کلاسیک] یا ناپایگاهی
    - مشی پایگاهی Database Approach

یعنی ممکن است مشی‌های بینابینی نیز وجود داشته باشد.

یادآوری

کارهای لازم در مشی فایلینگ به طور خلاصه:

- توجه: این کارها معمولا برای هر زیرمحیط به طور جداگانه انجام می شود. ← تعدادی سیستم کاربردی جدا (نامجتمع) و بی ارتباط در یک محیط ...

۱- مطالعه و شناخت محیط

۲- انجام مهندسی نیاز ها Requirement Engineering ←

۱- تشخیص نیاز های داده‌ای

۲- تشخیص نیاز های پردازشی

۳- مستندسازی نیازها

۴- دریافت تایید سازمان

۳- تعیین مشخصات سیستم کاربردی System Specification

۴- [انتخاب پیکربندی سخت افزار و نرم افزار H/S]

۵- [انتخاب یک FS و/یا DMS]سیستم واسط ISR

۶- طراحی تعدادی فایل (طبق مشخصات سیستم)

۶-۱- تعیین فرمت رکورد

۶-۲- تعیین ساختار فایل

ساختار فایل: ساختاری که براساس آن فقره داده ها (رکوردها) در سطح منطقی [و/یا فیزیکی] با یکدیگر مرتبطند.

ساختار فایل یک امکان برای نمایش ارتباط بین فقره داده‌هاست (Data Items) خواه در سطح نمایش منطقی باشد یا فیزیکی.

کنجکاوی: چند نوع ساختار فایل وجود دارد؟

۶-۳- نحوه دسترسی به رکوردها – استراتژی دسترسی

۶-۴- اندازه فایل ها

۶-۵- میزان گسترش چه میزان باشد

۶-۶- ارتباط با فایل های دیگر

۶-۷- عملیات مجاز در فایل‌ها + کاربران

مثال مقدماتی (مشی فایلینگ)

- **کارهای لازم در مشی فایلینگ به طور خلاصه : (ادامه)**

۷- طراحی واسط‌های کاربری (UFI)

۸- طراحی تعدادی برنامه کاربردی (Application Program) [ضمن تعیین تراکنش(ها)]

۹- تولید برنامه‌های { ایجاد / کنترل / پردازش } فایل‌ها

۱۰- ایجاد محیط فیزیکی «ذ.ب.ا.» به طور آزمایشی (برای داده های تست)

۱۱- ایجاد محیط فیزیکی «ذ.ب.ا.» با داده‌های واقعی اما حجم محدود و انجام تست مرحله دوم

[۱۲- ایجاد محیط فیزیکی «ذ.ب.ا.» با داده‌های واقعی و حجم واقعی و انجام تست مرحله سوم]

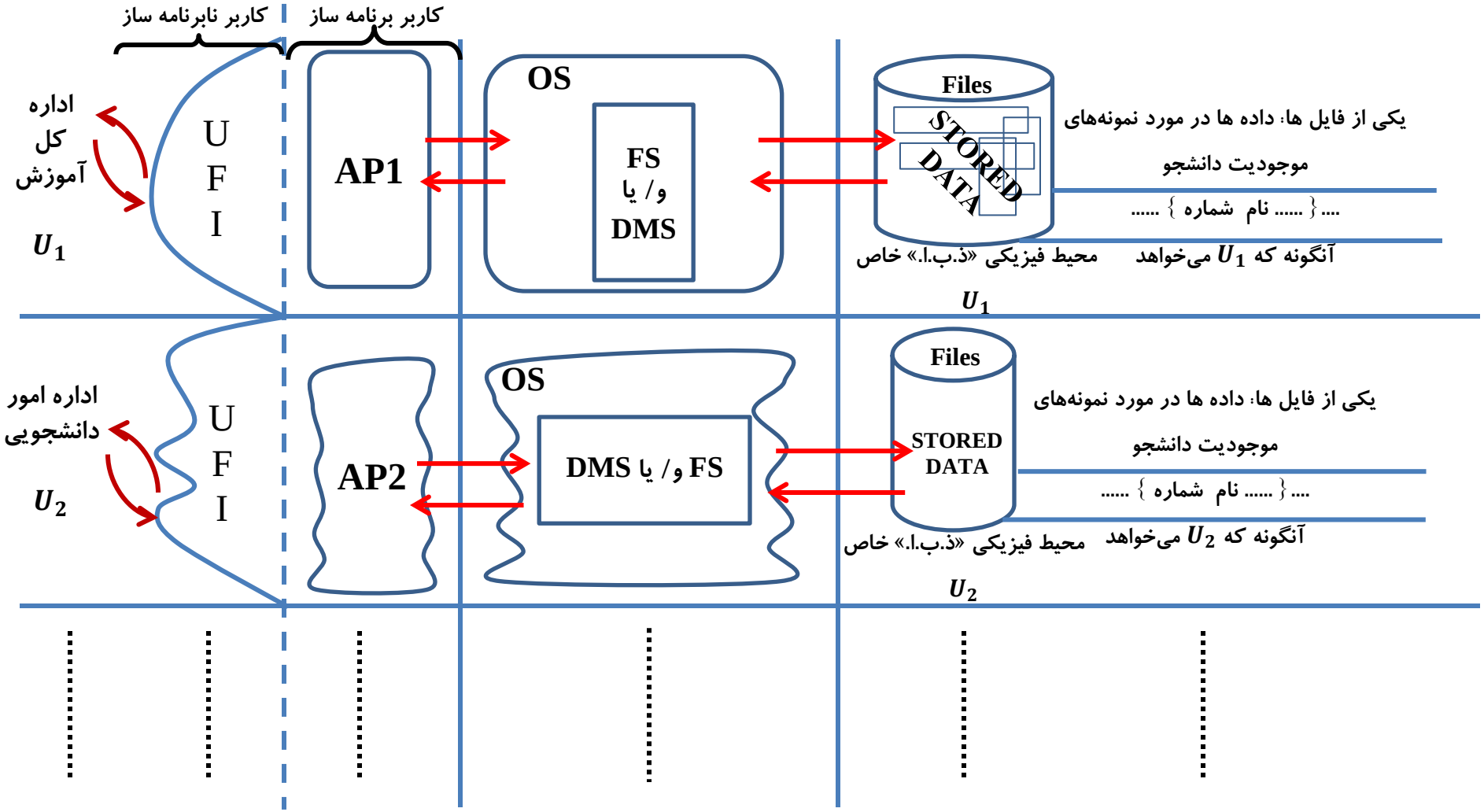
۱۳- رفع اشکال‌ها در هر مرحله

۱۴- ایجاد محیط فیزیکی واقعی با نصب، پیکربندی و ورود داده‌های اولیه (Data Entry)

۱۵- آغاز بهره برداری و نگهداری سیستم

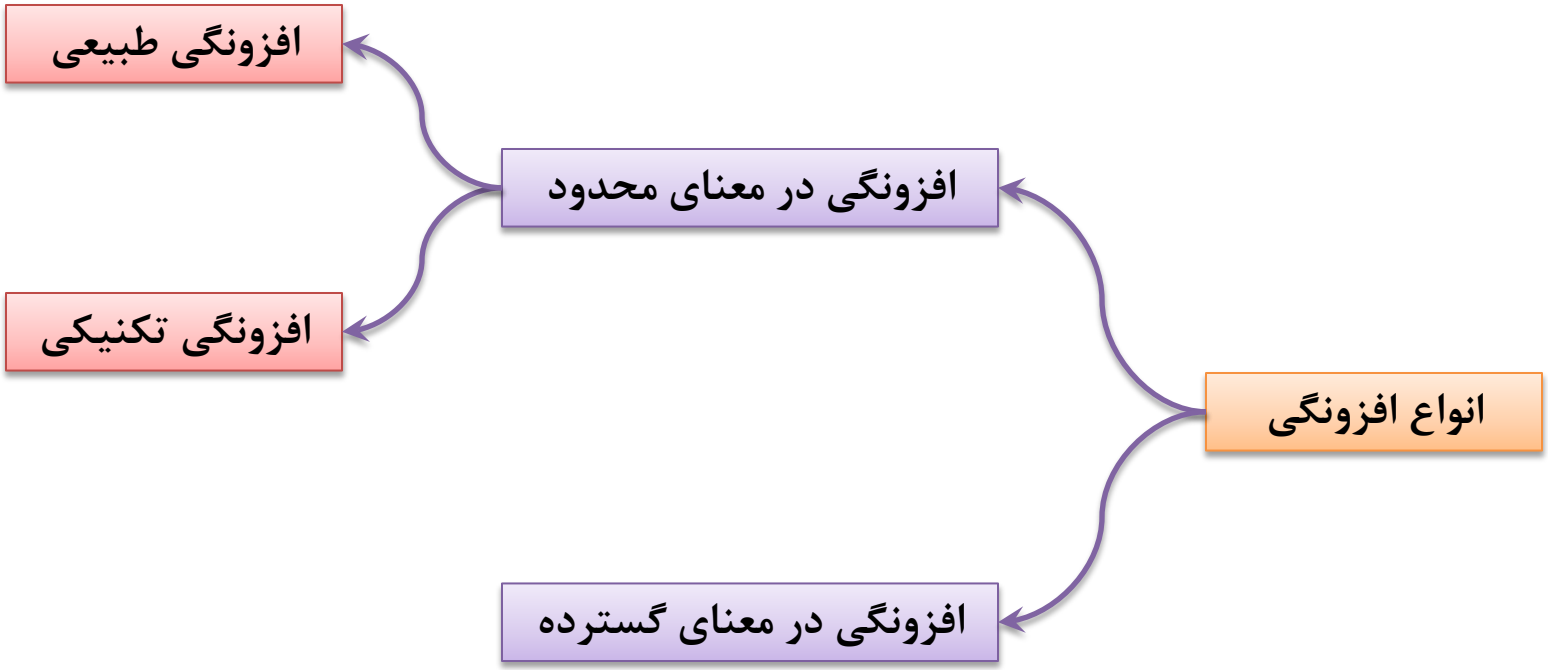
۱۶- رفع معایب و بهینه‌سازی سیستم

نمایش شماتیک مشی فایلینگ



معایب مشی فایلینگ

- **برخی از معایب مشی فایلینگ:**
  - وجود سیستم های نامجتمع در یک سازمان [محیط] و نامرتبط به هم
  - عدم وجود یک سیستم کنترل متمرکز روی کل داده‌های سازمان
  - وجود افزونگی زیاد
  - خطر بروز ناسازگاری داده ها (Data Inconsistency) ← کنجکاوی: جنبه های بروز ناسازگاری کدامند؟
  - عدم امکان اعمال ضوابط حفظ امنیت داده‌ها (Data Security)
  - عدم امکان اشتراکی شدن داده ها (Data Sharing) [یا در حداقل و یا با دشواری]
  - مصرف نابهینه سخت افزار (به ویژه سخت افزار ذخیره‌ساز)
  - وابسته بودن برنامه ها به جنبه های فایلینگ محیط ذخیره‌سازی، به گونه‌ای که اگر قرار باشد در فایلینگ تغییراتی ایجاد شود، برنامه ها هم متناسبا باید تغییر یابد. (به طور مثال فرمت ساختار یا نحوه دسترسی (Access Strategy) را تغییر دهیم)


انواع افزونگی
افزونگی در معنای محدود
افزونگی در معنای گسترده
افزونگی طبیعی
افزونگی تکنیکی

افزونگی (ادامه)

- **افزونگی در معنای محدود** (یعنی درون فایلی -intrafile redundancy – در مباحث فایلینگ)
  - عبارت است از تکرار ذخیره سازی مقادیر (value) یک صفت یا بیش از یک صفت در فایل داده‌ای یا فایل کمکی آن.
- این نوع افزونگی گونه‌هایی دارد:

۱- **طبیعی**: ناشی از ماهیت داده های محیط (مثل صفت رشته دانشجو که برای دانشجویان مختلف می‌تواند یکسان و در نتیجه تکراری باشد)

- کنجکاوی: برای کاهش مصرف حافظه در حالت افزونگی طبیعی چه باید کرد؟

۲- **تکنیکی**: ناشی از استفاده از یک تکنیک معمولا برای افزایش سرعت (مثل نمایه سازی [شاخص بندی Indexing])

افزونگی (ادامه)

- **افزونگی در معنای گسترده** (یعنی برون‌فایلی- در مباحث پایگاه داده)
- عبارت است از تکرار ذخیره‌سازی داده‌ها در مورد نمونه‌های یک یا بیش از یک نوع موجودیت از یک محیط.
  - این نوع افزونگی نه از نوع طبیعی و نه از نوع تکنیکی است بلکه ناشی از رهیافت انتخاب شده برای طراحی و تولید سیستم‌های کاربردی است.
  - به طور مثال تکرار اطلاعات دانشجویان در دو زیرسیستم اداره کل آموزش و زیرسیستم اداره امور دانشجویی.
- **نکته:** افزونگی از نوع طبیعی و تکنیکی در پایگاه داده هم می‌تواند وجود داشته باشد.

کنکاوی: دلایل بروز افزونگی در سیستم های ISR به ویژه سیستم های پایگاهی کدامند؟

یادآوری: نمایه‌سازی (نمونه‌ای از افزونگی تکنیکی)

یادآوری

- تشکیل شده از تعدادی درایه (مدخل-entry)

هر مدخل اشاره دارد به { یک / یا / گروهی (به صورت چند سطحی) } از رکورد ها

یک ← نمایه متراکم dense

گروهی (به صورت چند سطحی) ← نمایه نامتراکم Non-dense

آدرس	مقدار

آدرس ← تکیه گاه (Anchor point)

مقدار ← مقدار یک صفت (معمولا کلید)

یادآوری: نمایه‌سازی (ادامه)

نمایه نامتراکم

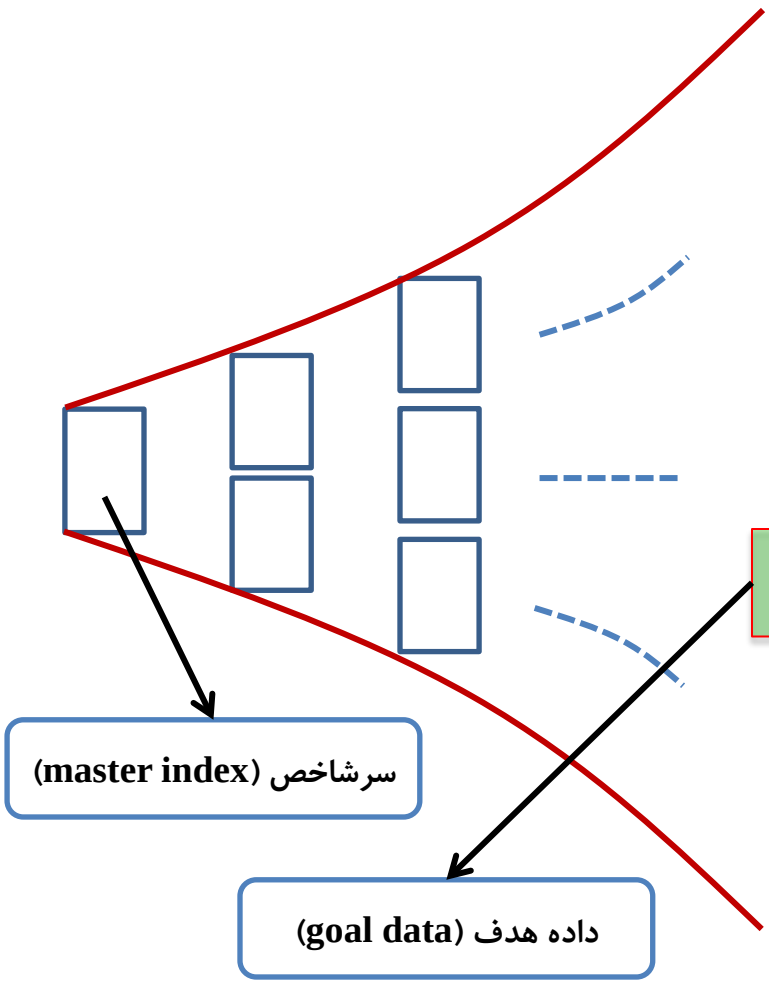
(چندسطحی معمولا با ساختار B-Tree یا B^+-Tree)

نمایه متراکم

(زمان جستجو بالاست)

فایل نمایه سازی شده

۱۰۰	۱	۱۰۰		نرم‌افزار
۱۰۱	۲	۱۰۱		نرم‌افزار
۱۰۲	۳	۱۰۲		سخت‌افزار
۱۰۳	۴	۱۰۳		نرم‌افزار
۱۰۴	۵	۱۰۴		سخت‌افزار
۱۰۵	۶	۱۰۵		نرم‌افزار
⋮		⋮		
k		k		نرم‌افزار
⋮		⋮		
۹۹۷	۸۹۸	۹۹۷		سخت‌افزار
۹۹۸	۸۹۹	۹۹۸		سخت‌افزار
۹۹۹	۹۰۰	۹۹۹		نرم‌افزار



مثال مقدماتی : اما در مشی پایگاهی

- **کارهای لازم در انجام یک «پروژه پایگاهی»:** (فعلا نه در جزئیات)

به منظور ایجاد یک سیستم یکپارچه

۱- مطالعه و شناخت محیط

۲- انجام مهندسی نیاز ها Requirement Engineering

۳- تعیین مشخصات سیستم یکپارچه

۱- تشخیص نیاز های داده ای

۲- تشخیص نیاز های پردازشی

۳- مستندسازی نیاز ها

۴- دریافت تایید سازمان

[۴- انتخاب (حداقل) یک پیکربندی H/S]

[۵- انتخاب (حداقل) یک DBMS (یک تصمیم گیری حیاتی است)]

۶- مدلسازی معنایی داده ها (data semantic modeling) ← در مشی فایلینگ انجام نمی‌شود.

۷- طر احی منطقی پایگاه داده Logical Data Base Design

۸- طراحی فیزیکی پایگاه داده Physical Data Base Design

مجتمع

۹- طراحی UFI ها

۱۰- طراحی AP ها [ضمن تعریف تراکنش ها (Transactions)]

ادامه مثال مقدماتی (مشی پایگاهی)

- ادامه:...

۱۱- تولید برنامه‌های تعریف (ایجاد) و کنترل DB

۱۲- تولید برنامه‌های عملیات در داده‌ها (پردازش داده‌ها)

۱۳- ایجاد محیط فیزیکی «ذ.ب.ا.» با داده های تستی و رفع اشکال ها (تست مرحله اول)

۱۴- ایجاد محیط فیزیکی «ذ.ب.ا.» با داده های واقعی اما حجم محدود و انجام تست مرحله دوم

۱۵- ایجاد محیط فیزیکی «ذ.ب.ا.» با داد های واقعی و حجم واقعی و انجام تست مرحله سوم

۱۶- تنظیم سیستم پایگاهی (Data Base System Tuning)

به طور مثال به منظور افزایش کارایی

۱۷- آغاز بهره برداری و نگهداری از سیستم

۱۸- گسترش سیستم

یکی از ویژگی های DBMS گسترش پذیری سیستم است.

۱۹- رفع معایب و بهینه‌سازی سیستم

گفتگوی

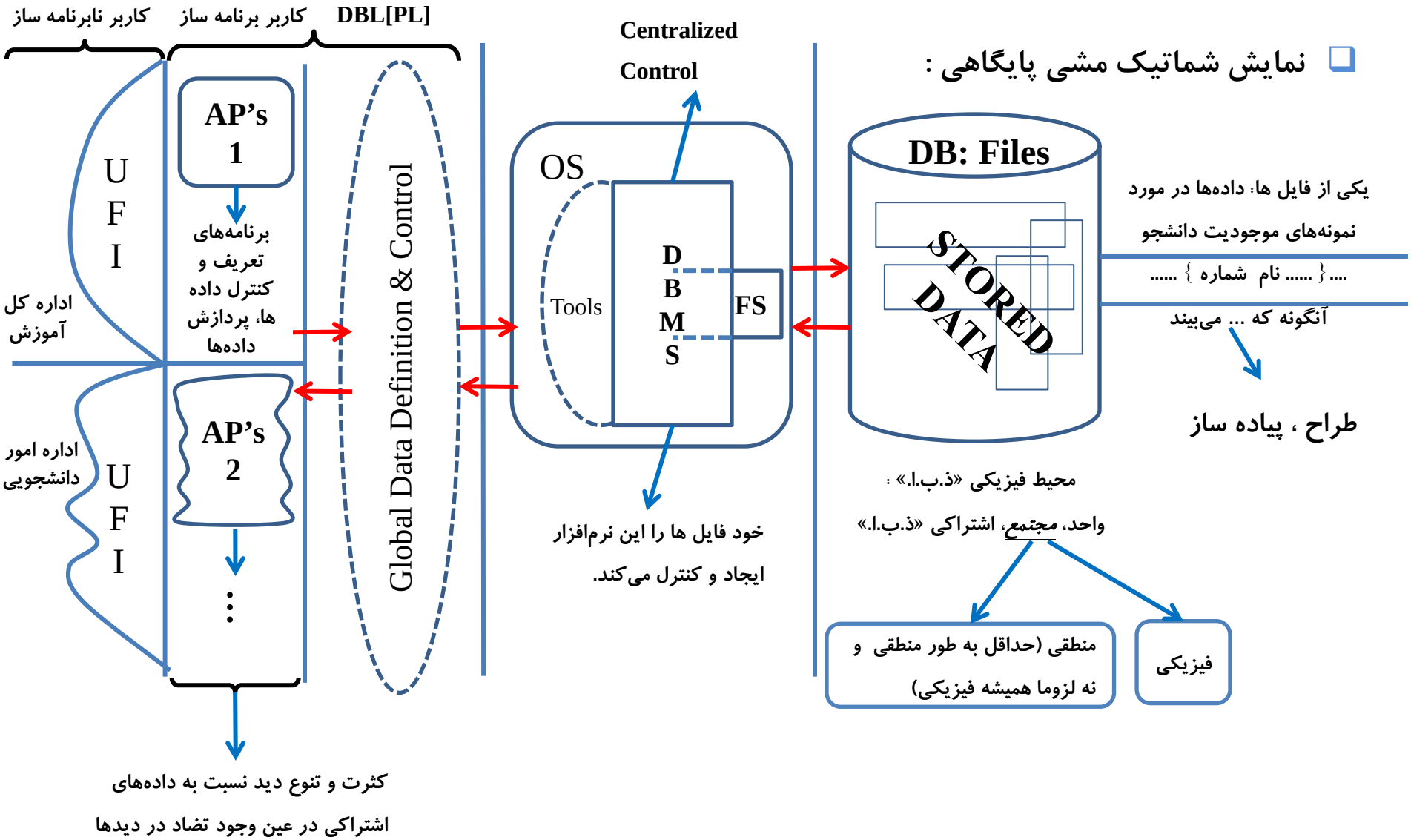
مزایا و معایب جداسازی این دو دسته برنامه

تعریف و کنترل و عملیات در داده‌ها چیست؟

۱- از دیدگاه عملیات در داده‌ها

۲- از دیدگاه زبان‌های برنامه‌سازی

ادامه مثال مقدماتی (مشی پایگاهی)



ادامه مثال مقدماتی (مشی پایگاهی)

کنکاوی؟ چگونه از این کثرت دید می توان به آن «وحدت» رسید؟ ←

۱- خود نرم افزار DBMS

۲- معماری پایگاه داده

- تمرین: مزایای مشی پایگاهی چیست؟ ← (طبق معلومات فعلی: عکس معایب مشی فایلینگ)
- تمرین: چند سطح تعریف داده داریم؟

یادآوری: تراکنش Transaction:

- دنباله ای از عملیات («قطعه برنامه») که معمولاً حد اقل یک عمل تغییردهنده (درج، حذف، به روزرسانی) در محیط ذخیره‌سازی داده‌ها انجام می‌دهد و باید یا به تمامی اجرا شود و یا اجرا نشده تلقی شود.
- دارای خواص ACID (**A**tomicity **C**onsistency **I**solation **D**urability)

A: یا همه یا هیچ

C: سازگاری

I: انفراد و جدایی

D: دوام (پایداری)

مثال: شرط سازگاری پایگاه داده در این مثال : A+B ثابت باشد

```
BEGIN TRANS
        READ (A)
        A = A – 50
        UPDATE (A)
        READ (B)
        B = B + 50
        UPDATE (B)
END TRANS
```

عناصر اصلی محیط پایگاهی:

۱- سخت افزار
- ذخیره سازی
- پردازشگر
- ارتباطی (همرسانی) Data Communication

۲- نرم افزار

۳- کاربر

۴- داده

عناصر محیط پایگاهی – (۱) سخت‌افزار

- **سخت افزار ذخیره سازی:**
  - رسانه اصلی: دیسک ترجیحا با تکنولوژی RAID (Redundant Array of Inexpensive Disk)
  - رسانه فرعی: نوار مغناطیسی [از جمله برای تولید نسخه های پشتیبان]

کنکاوی

: تکنیک های تولید نسخه پشتیبان؟

سطوح مختلف Back up

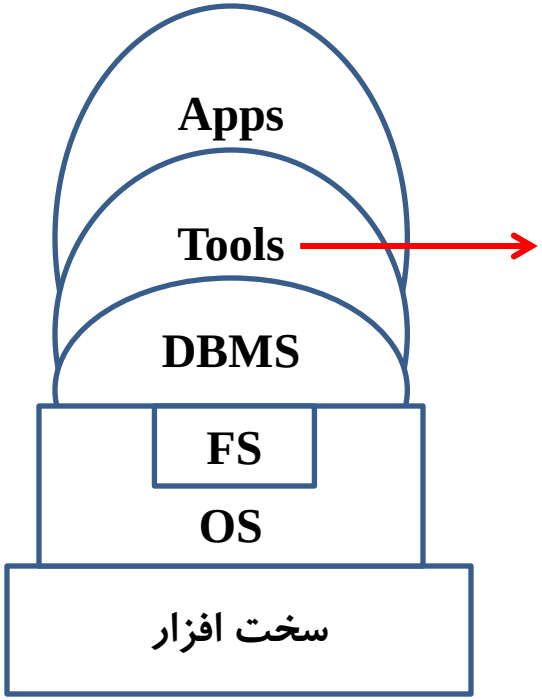
- اغلب DBMSهای امروزی تکنیک های تولید Back up را دارا هستند.

- **سخت افزار پردازشگر:**
  - کامپیوتر های معمولی از هر رده [PC, main,...]
  - اما ماشین‌های خاص DB هم داریم : DB Machines

- **سخت افزار ارتباطی (همرسانی):**
  - امکانات محلی: برای ارتباط دستگاه‌های جانبی با پردازنده
  - امکانات شبکه‌ای: برای ایجاد شبکه در سیستم پایگاهی نامتمرکز

عناصر محیط پایگاهی – (۲) نرم افزار

- انواع نرم افزارهای مطرح در محیط پایگاهی:
  - سیستم عامل و سیستم فایل (OS و FS)
  - سیستم مدیریت پایگاه داده‌ها (DBMS)
  - ابزارها (Tools)
  - برنامه‌های کاربردی (Apps)

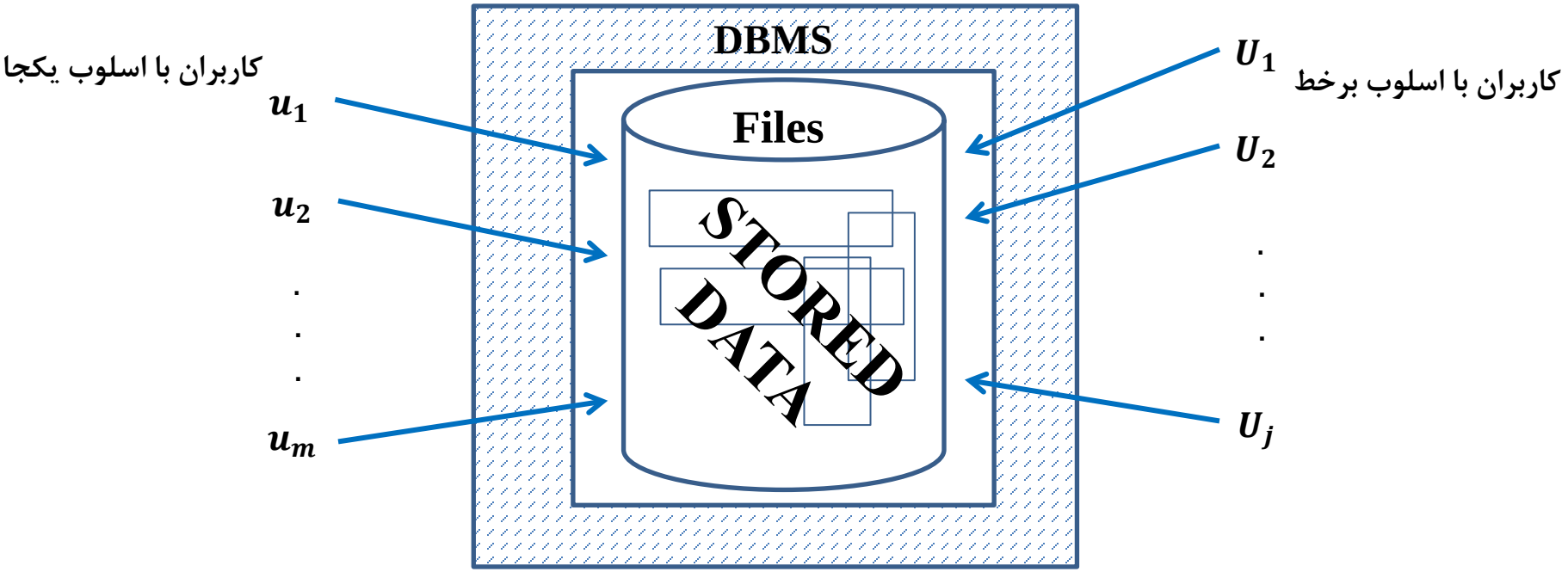


یا با خود DBMS می فروشند،
یا جداگانه خریداری می‌شود و به
امکانات آن اضافه می‌شود.

تسهیلات نرم افزار

عناصر محیط پایگاهی – (۳) کاربر

- در معنای عام هر استفاده‌کننده از سیستم پایگاهی را **کاربر** گوییم که انواع مختلفی دارد.



عناصر محیط پایگاهی – (۳) کاربر (انواع)

- انواع کاربر از نظر اسلوب عملیاتی:
  - **Batch- یکجا** (تعدادی برنامه یا پرس‌وجو جمع آوری می‌شود و به صورت یکجا به سیستم داده می‌شود و جواب آن بر می‌گردد.)
  - **Online – برخط – پیوسته** (یک برنامه یا پرس‌وجو به سیستم داده می‌شود، اجرا می‌شود، و جوابش برمی‌گردد.)
  - **Interactive-تعاملی –** بسته به اینکه چه جوابی داده شود عمل دیگری از کامپیوتر درخواست می‌شود.
    - Online لزوما Interactive نیست اما Interactive لزوما Online است.

- سیستم پایگاهی به صورت پیش فرض چند کاربره (multi-user) است.

عناصر محیط پایگاهی – (۴) داده

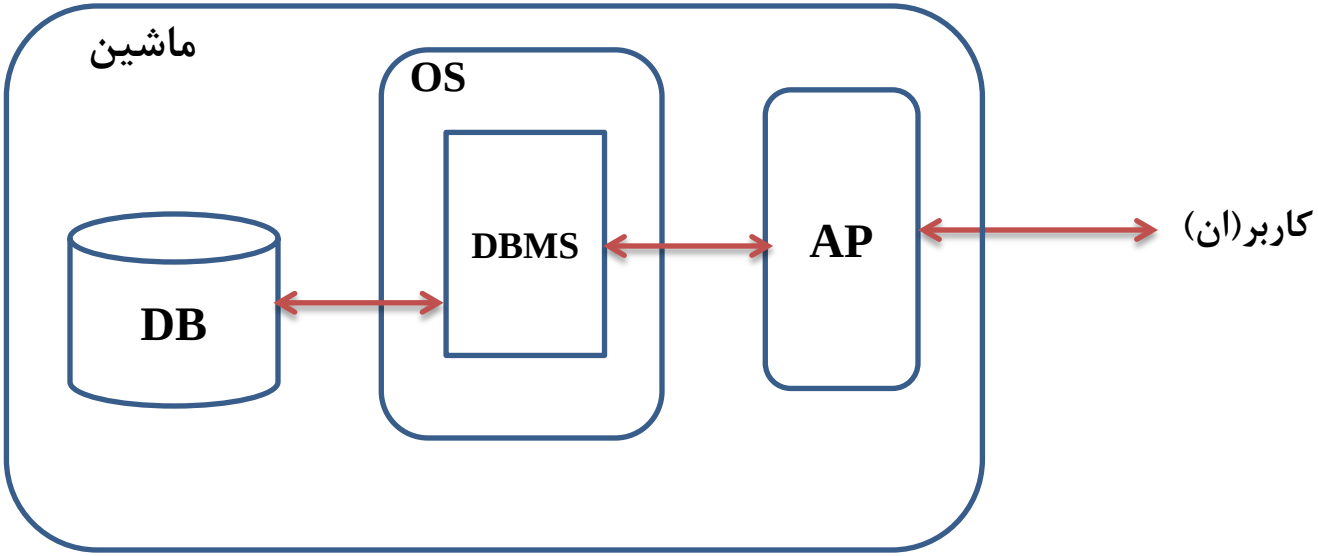
- **داده‌های ذخیره شده** در یک سیستم پایگاهی عبارتند از:
  - داده‌های کاربران
  - داده‌های سیستمی
- مباحث مرتبط با داده در محیط پایگاهی در ادامه درس مطرح می‌گردد.

انواع معماری سیستم پایگاهی

- **سوال:** می‌خواهیم یک سیستم کاربردی پایگاهی ایجاد کنیم.بر اساس کدام معماری ایجادکنیم؟
- **در توصیف معماری یک سیستم باید مشخص کنیم که**
  - از چه مولفه‌هایی، از هر مولفه چند عدد و با چه کیفیتی تشکیل شده است،
  - مولفه‌ها چگونه با هم ترکیب شده‌اند (جنبه ساختاری سیستم)،
  - مولفه‌ها چگونه با یکدیگر در تعامل هستند (جنبه رفتاری سیستم).
- **انواع معماری سیستم پایگاهی:**
  - معماری متمرکز
  - معماری نامتمرکز
    - معماری مشتری-خدمتگزار
    - معماری توزیع‌شده
    - معماری چندپایگاهی
    - معماری با پردازش موازی

معماری متمرکز

- در این معماری یک پایگاه داده (متمرکز و مجتمع) روی یک سیستم کامپیوتری و بدون ارتباط با سیستم کامپیوتری دیگر ایجاد می‌شود.
- معمولا به صورت تک کاربری و برای کاربردهای کوچک و با امکانات محدود از این معماری استفاده می‌شود.



معماری مشتری– خدمتگزار

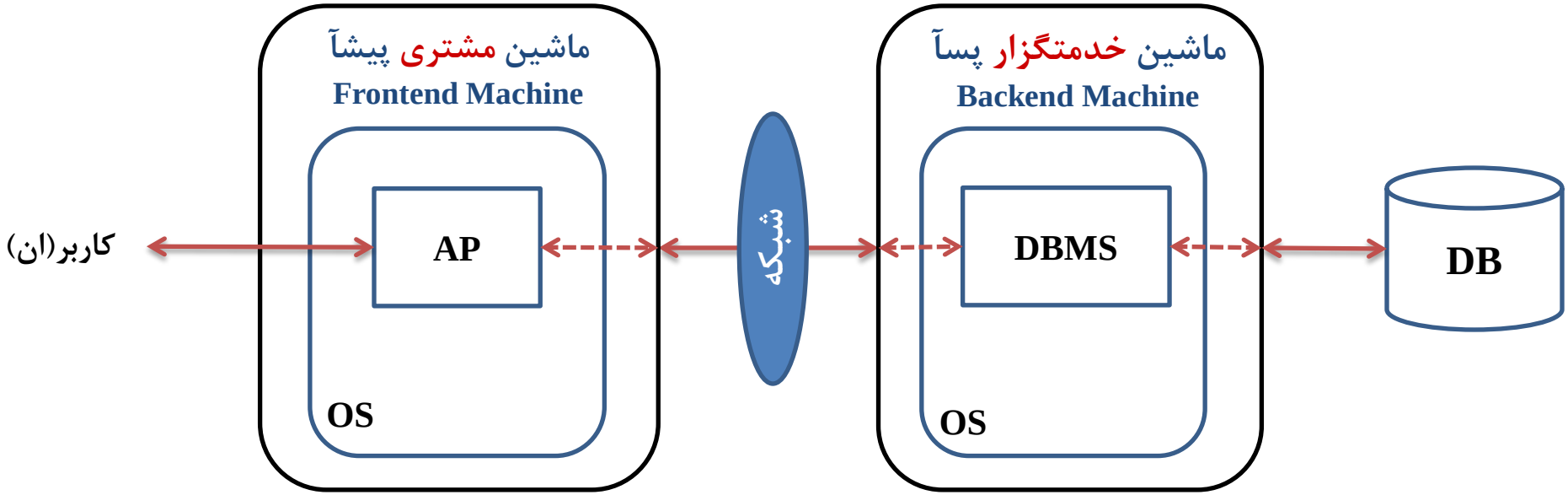
- **دلیل:** دلیل اصلی استفاده از معماری مشتری–خدمتگزار (Client-Server): تقسیم وظایف سیستم
- **تعریف:** هر ماشینی (فیزیکی یا منطقی) که خدمتی را به ماشین دیگر بدهد، **خدمتگزار** نامیده می‌شود.

مثال: نمونه‌هایی از انواع خدمتگزارها: File Server، Print Server، Message Server، DB Server

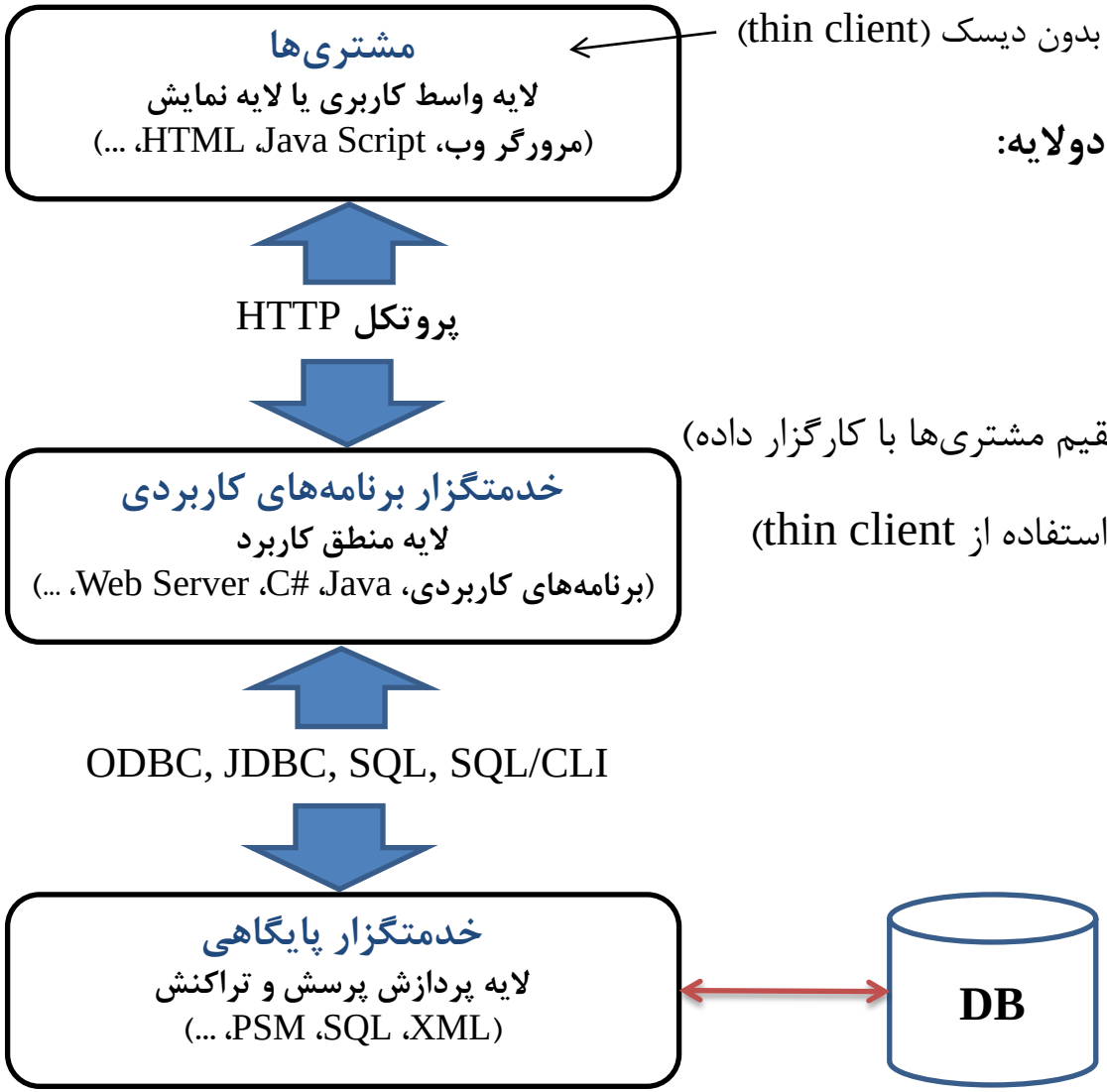
- **انواع معماری مشتری– خدمتگزار**
  - معماری تک مشتری– تک خدمتگزار
  - معماری چند مشتری– تک خدمتگزار
  - معماری تک مشتری– چند خدمتگزار
  - معماری چند مشتری– چند خدمتگزار

معماری مشتری –خدمتگزار دو لایه

- معمولا شامل دو سایت:
  - **سایت مشتری:** تمام برنامه‌های کاربردی در آن اجرا می‌شوند.
  - **سایت خدمتگزار:** تمام داده‌ها در آن ذخیره می‌شوند
- به این معماری، **معماری دولایه** (2-tier) نیز گویند.



معماری مشتری– خدمتگزار سه لایه

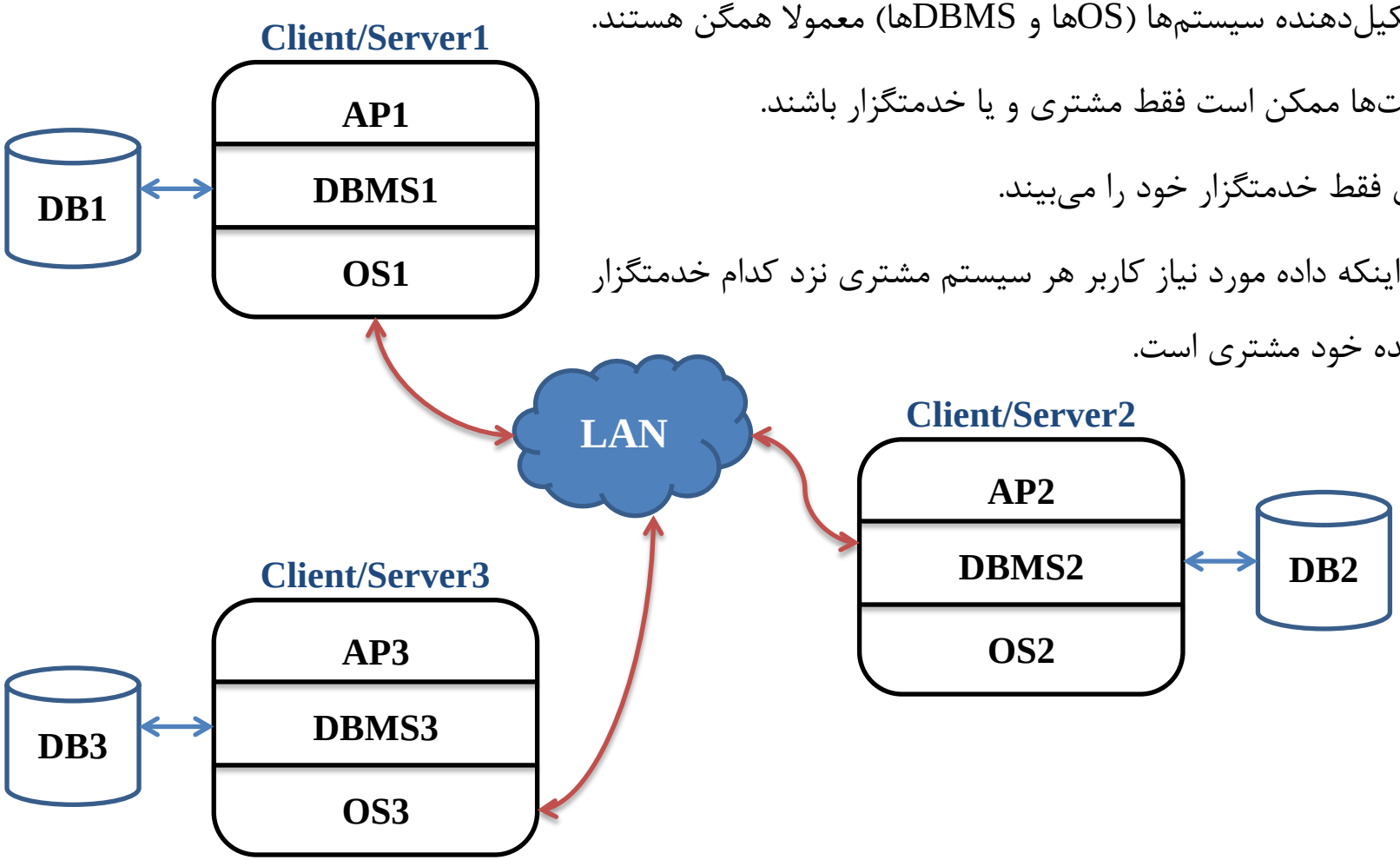


ماشین‌های ساده، ارزان و حتی بدون دیسک (thin client)

- **برخی مزایای معماری سه لایه نسبت به دولایه:**
  - گسترش‌پذیری بهتر
  - کارایی بالاتر
  - امنیت داده‌ای بیشتر (عدم ارتباط مستقیم مشتری‌ها با کارگزار داده)
  - قابلیت کاهش هزینه سخت افزاری (با استفاده از thin client)

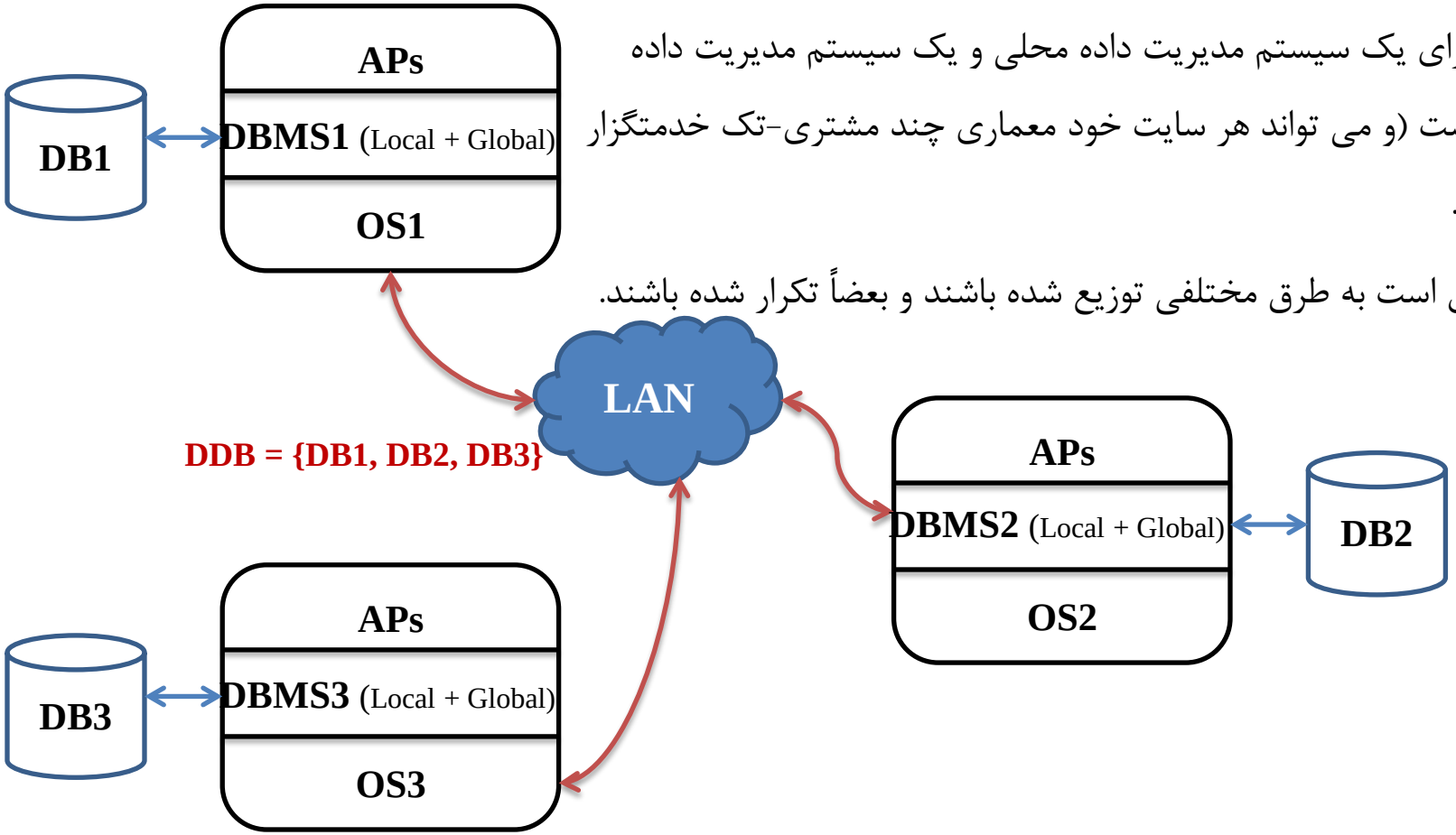
معماری چند مشتری–چند خدمتگزار

- سیستم‌های پایگاهی همزمان یا ناهمزمان ایجاد می‌شوند.
- اجزای تشکیل‌دهنده سیستم‌ها (OSها و DBMSها) معمولا همگن هستند.
- برخی سایت‌ها ممکن است فقط مشتری و یا خدمتگزار باشند.
- هر مشتری فقط خدمتگزار خود را می‌بیند.
- مسئولیت اینکه داده مورد نیاز کاربر هر سیستم مشتری نزد کدام خدمتگزار است برعهده خود مشتری است.



معماری توزیع‌شده

- مجموعه‌ای است از چند پایگاه داده منطقاً یکپارچه (مجتمع)، ولی به طور فیزیکی توزیع شده روی یک شبکه کامپیوتری.
- توزیع شدگی از دید برنامه‌ها و کاربران پایگاه داده پنهان است.
- هر سایت دارای یک سیستم مدیریت داده محلی و یک سیستم مدیریت داده توزیع‌شده است (و می تواند هر سایت خود معماری چند مشتری-تک خدمتگزار داشته باشد).
- داده‌ها ممکن است به طرق مختلفی توزیع شده باشند و بعضاً تکرار شده باشند.



پرسش و پاسخ . . .

amini@sharif.edu

به نام آنکه جان را فکرت آموخت

بخش دوم :
مدلسازی معنایی داده‌ها

مرتضی امینی

نیمسال دوم ۹۴-۹۵

(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)

مدلسازی معنایی داده‌ها (Semantic Data Modeling)

- داده های ذخیره شدنی در DB
  - کاربری
  - سیستمی

ابتدا باید مدل شوند.

داده‌های کاربری

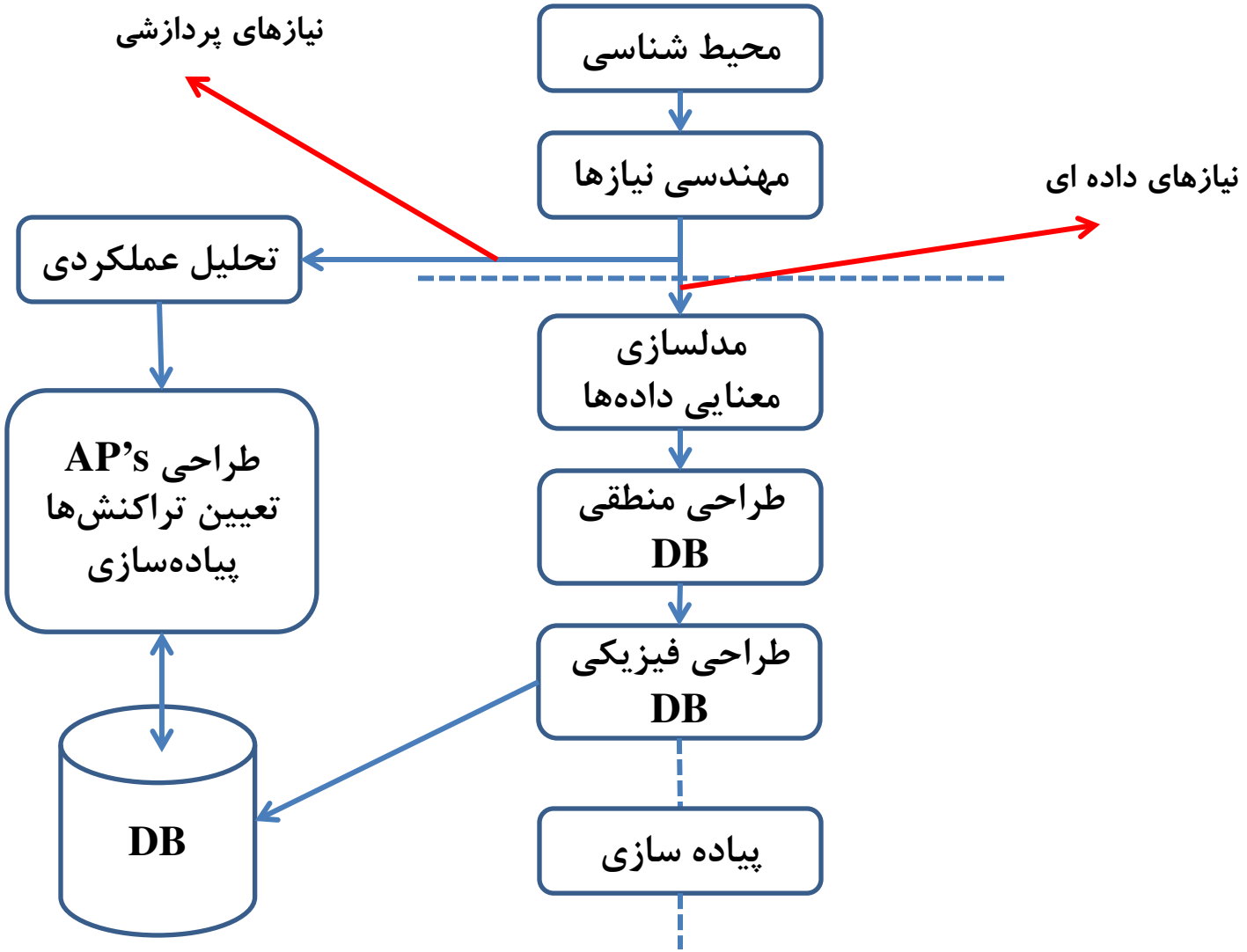
- موسومند به داده‌های عملیاتی (مثلا اطلاعات دانشجویان، درس‌ها و اساتید در محیط عملیاتی دانشگاه)
- پایا هستند: بعد از اجرای برنامه کاربر کماکان در سیستم ماندگارند[حسب تعریف]
- لزوماً همان داده های ورودی/خروجی (I/O) نیستند. هر داده موجود در پایگاه داده لزوما داده ورودی نیست و هر داده خروجی از پایگاه لزوما در پایگاه ذخیره شده نیست (مانند داده‌های محاسبه شده از داده‌های موجود- مثلا میانگین نمرات)

داده‌های سیستمی

- سیستم تولید می‌کند برای انجام وظایفش (مثلا اطلاعات مربوط به جداول پایگاه داده و یا اطلاعات مربوط به ستونهای موجود در جداول)

مراحل تولید سیستم اطلاعاتی

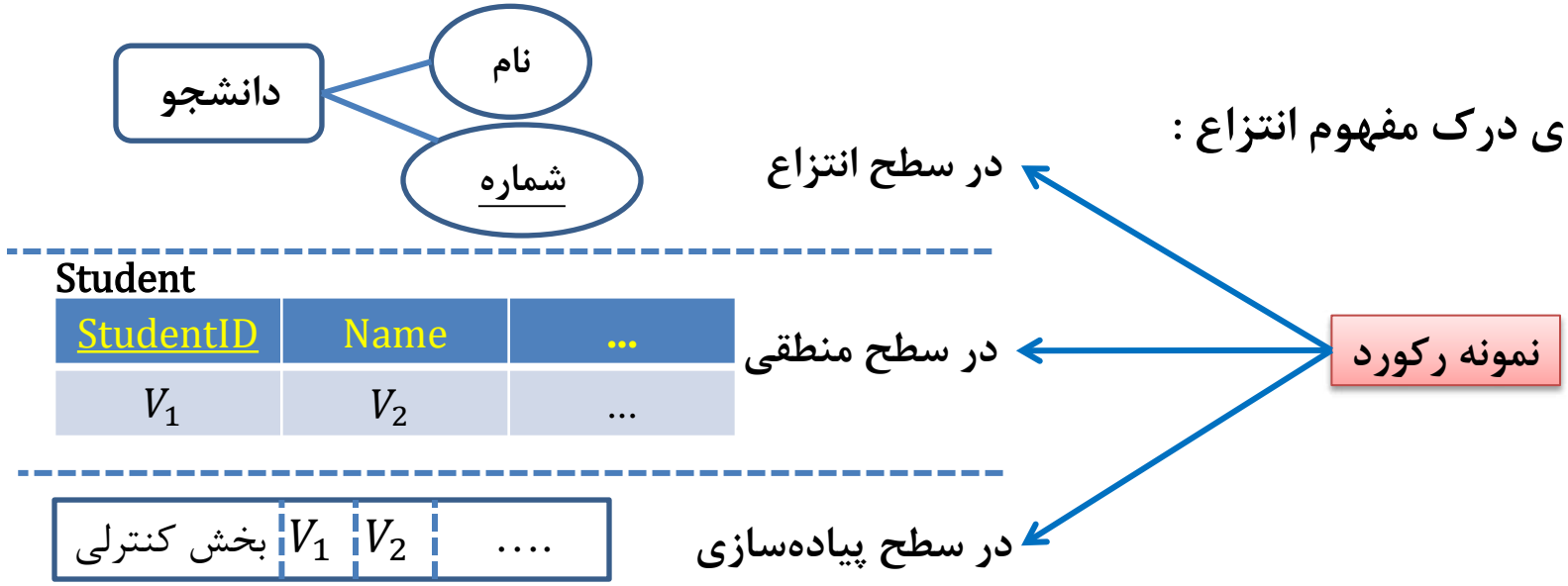
بخش دوم: مدلسازی معنایی داده ها



مدلسازی معنایی داده‌ها:

- ارائه یک مدل کلی (در بالاترین سطح انتزاع) از داده‌های محیط با استفاده از مفاهیم انتزاعی و براساس *معنایی* که کاربر برای داده‌ها قائل است.
- **مفهوم انتزاعی:** مفهومی است فراتر از سطح منطقی و طبعاً فراتر از سطح پیاده‌سازی

برای درک مفهوم انتزاع :



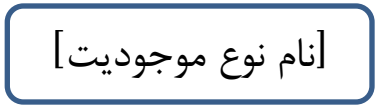
- **برای مدلسازی نیاز به روش داریم:**
  - روش رایج تر در دانش و تکنولوژی پایگاه داده
    - **روش ER (Entity Relationship):** ER مبنایی / ER گسترش یافته (Extended or Enhanced ER)
    - **روش UML (Unified Modeling Language):** خاصّ مدلسازی معنایی داده‌ها نیست بلکه برای مدلسازی و طراحی سیستم های نرم‌افزاری است. لذا با آن می‌توان پایگاه داده را مدل کرد.

- **سه مفهوم اساسی داریم:**
  - نوع موجودیت Entity Type
  - صفت (خصیصه – ویژگی) Attribute
  - نوع ارتباط Relationship Type

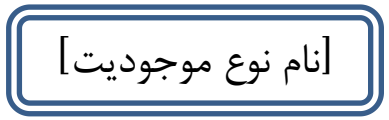
- **نمودار ER:**
  - نموداری است که سه مفهوم اساسی نوع موجودیت، صفت و نوع ارتباط در آن نمایش داده می‌شوند. در واقع این نمودار امکانی است برای نمایش مدلسازی و اولین طرح پایگاه داده‌ها در بالاترین سطح انتزاع.
  - برای رسم این نمودار به نمادهایی نیاز داریم. در این درس از نمادهای چِن استفاده می‌شود.

نمادهای نمودار ER مبنایی

- نوع موجودیت



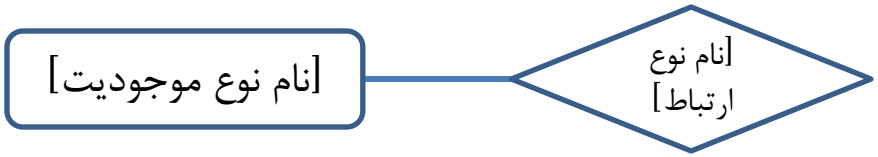
- نوع موجودیت ضعیف



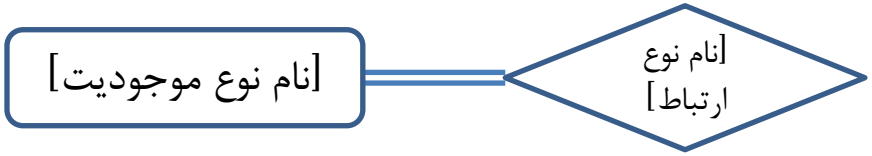
- نوع ارتباط

- نوع ارتباط موجودیت ضعیف با قوی

- مشارکت نوع موجودیت در نوع ارتباط



- مشارکت الزامی



نمادهای نمودار ER مبنایی (ادامه)

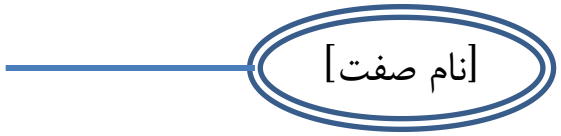
- صفت

- صفت شناسه اول

- صفت شناسه دوم (در صورت وجود)

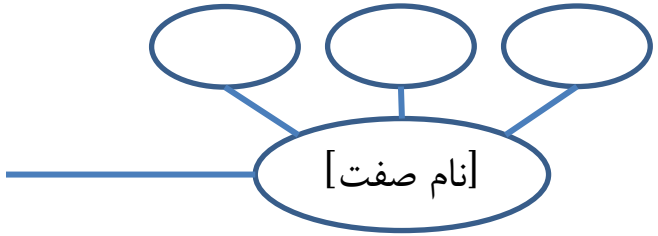
- صفت شناسه مرکب (مثلا دو صفتی)

- صفت چندمقداری

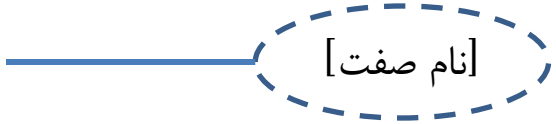


نمادهای نمودار ER مبنایی (ادامه)

- صفت مرکب



- صفت مشتق (مجازی یا محاسبه‌شدنی)



- چندی ارتباط

نوع موجودیت:

- مفهوم کلی شیئ، چیز، پدیده و به طور کلی آنچه از یک محیط که می‌خواهیم در موردش اطلاع داشته باشیم.

- خرد جهان واقع Micro Real World
- Mini World
- جهان مطرح Universe of Discourse(UOD)

مثال

محیط عملیاتی : دانشگاه

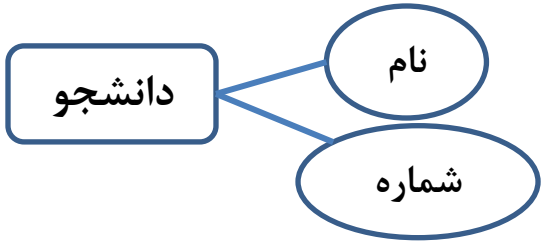
- نوع موجودیت‌ها
  - ۱- دانشجو
  - ۲- درس
  - ۳- استاد
  - ۴- کارمند
  - ...

- تذکر: اولین قدم در مدلسازی معنایی تشخیص درست نوع موجودیت‌ها است.

کنکاوی

در مثال فوق آیا دانشگاه یک نوع موجودیت در نظر گرفته می‌شود یا خیر؟

ER مبنایی – نوع موجودیت (ادامه)



- هر نوع موجودیت:
  - یک نام دارد.
  - یک معنا دارد.
  - مجموعه‌ای از صفات دارد (حداقل یکی).
  - نمونه‌هایی دارد (حداقل یک نمونه).
  - ارتباط(هایی) با نوع موجودیت(های) دیگر دارد.
  - نوع موجودیت دو گونه است.
    - قوی (مستقل) Strong
    - ضعیف (وابسته) Weak

کنجکاوی: در چه حالتی بهتر است نوع موجودیت تک صفتی را نوع موجودیت بگیریم؟ در چه حالتی نگیریم؟

کنجکاوی: در چه حالتی نوع موجودیت تک نمونه‌ای را موجودیت در نظر می‌گیریم؟

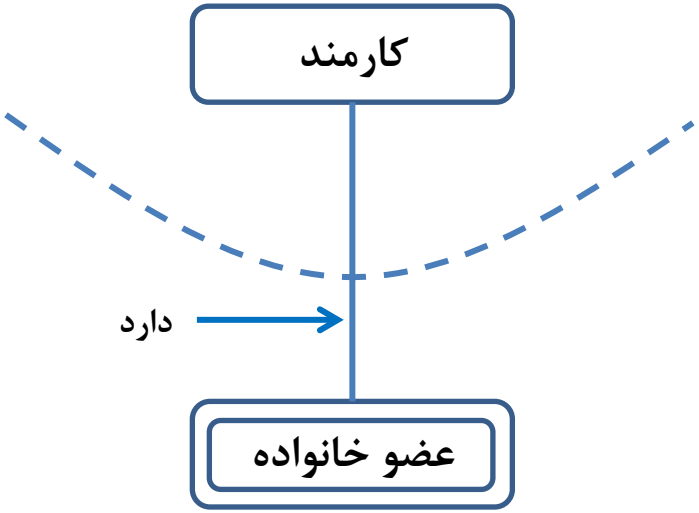
کنجکاوی: آیا نوع موجودیت ایزوله داریم؟

تعریف موجودیت قوی:

- نوع موجودیت E را **قوی** گوییم هرگاه خود مستقلاً در محیط مطرح باشد.

تعریف موجودیت ضعیف:

- نوع موجودیت F را **ضعیفِ** نوع موجودیت E گوییم هرگاه به آن «وابستگی وجودی» داشته باشد. (اگر E مطرح نباشد F هم مطرح نیست) به عبارتی F در مدلسازی دیده می‌شود به اعتبار E.
- تذکر: قوی و ضعیف بودن نسبی است.



مثال: عضو *خانواده* وابسته به نوع موجودیت *کارمند* است.

- صفت:
  - خصیصه یا ویژگی نوع موجودیت و هر نوع موجودیت مجموعه‌ای از صفات دارد که حالت یا وضع آن را توصیف می‌کند.

مثال

- محیط عملیاتی: دانشگاه
- نوع موجودیت: درس
- صفات: شماره، نام، تعداد واحد، زمان برگزاری، تاریخ امتحان، نوع درس (پایه، تخصصی، اختیاری،...)، سطح درس (کارشناسی، کارشناسی ارشد، دکترا)، ماهیت درس (نظری، عملی، ترکیبی)

- **هر صفت:**
  - یک نام دارد.
  - یک معنا دارد (معنای مشخص در حیطه معنایی مشخص).
  - یک دامنه یا میدان (Domain) دارد.

محدودیت میدانی یا دامنه‌ای

- **محدودیتهای صفت:**

۱– محدودیت میدانی { معنای، نوع، طیف مقادیر } صفت را مشخص می‌کند. و نه لزوماً نام صفت را.

۲– محدودیت نمایشی. **مثال:** قالب تاریخ yyyy/mm/dd

۳– محدودیت پردازشی ناشی از نوع صفت یا ناشی از قواعد محیط [غیر از آنچه ناشی از میدان است]

مثال: سن کاهش نمی‌یابد.

مثال: عدم جمع دو آدرس: محدودیت ناشی از میدان است.

۴– محدودیت وابستگی به یک صفت دیگر. **مثال:** وابستگی شمول به صفت دیگر $B\{values\} \subseteq A\{values\}$

۵– محدودیت یکتایی مقدار. **مثال:** شماره دانشجویی

کنکاوی؟ آیا صفت محدودیت های دیگری هم دارد؟

مثال

نمره دانشجو GR ← از دامنه Grade

- نوع : Real
- طیف مقادیر : [0,..,20]

رده بندی صفت:

- صفت
  - ۱- **شناسه** Entity Identifier (EID)
  - ۲- **ناشناسه**

Entity Identifier (EID) ←
- ۱- **یکتایی** مقدار Uniqueness
- ۲- مقادیرش همیشه **معلوم** باشد (هیچمقدارناپذیر) Not Null

ویژگی ذاتی ←

- ۳- طول کد نمایش حتی‌الامکان کوتاه
- ۴- مقادیرش حتی‌الامکان تغییر ناپذیر

بهتر است که داشته باشد. ←

- صفت
  - ۱- **ساده – تجزیه ناپذیر**: از نظر معنایی در یک محیط مشخص - اگر صفت را تجزیه کنیم، خودِ تکه ها مقداری از صفت در آن محیط نشود. مثال: عنوان درس
  - ۲- **مرکّب**: از چند صفت ساده (و می‌تواند ساختار سلسله مراتبی هم داشته باشد) مثال: آدرس (ترکیبی از استان، شهر، خیابان، ...)

- **توجه:** ساده یا مرکب بودن نسبی است و نه مطلق. بستگی به حیطه معنایی و کاربرد دارد. (مثال: آدرس از دید نشریه (ساده) یا از دید شهرداری (مرکب).

گفتگوی: اینکه صفت مرکّب را در یک فیلد ذخیره کنیم یا اجزا را در فیلد های مجزا به چه عواملی بستگی دارد؟

- صفت
  - ۱- **تک مقداری:** به ازای یک نمونه از نوع موجودیت E، حداکثر یک مقدار می‌گیرد. مثال: نام درس
  - ۲- **چند مقداری:** حدّاقل برای یک نمونه از نوع موجودیت E، بیش از یک مقدار . مثال: شماره تلفن استاد
- توجه
  - ساده – تک مقداری
  - مرکب - تک مقداری
  - ساده - چند مقداری
  - مرکب - چند مقداری
- صفت
  - ۱- **هیچمقدار پذیر** (Nullable یا Nullvalue): مقدار صفت می تواند ناشناخته، ناموجود، تعریف نشده یا غیر قابل اعمال باشد. مثال: شماره تلفن دانشجو
  - ۲- **هیچمقدارناپذیر** (Not nullabe): حتما مقدار صفت برای هر نمونه موجودیت باید معلوم باشد. مثال: شماره درس

گفتگوی: مشکلات هیچمقدار؟ package ها با آن چه برخوردی دارند؟

- صفت
  - ۱- **واقعی (Real)**: مقدار ذخیره شده در DB دارد. مثال: نمره درس
  - ۲- **مجازی – مشتق (Virtual)**: مقدار ذخیره شده در DB ندارد، سیستم با پردازشی معمولاً **محاسبه** و مقدارش را در اختیار کاربر قرار می دهد. مثال: میانگین نمرات درس

- **تذکر:** اگر صفتی ماهیت **محاسبه شوندگی** داشته باشد لزوماً مجازی نیست و ممکن است برای افزایش سرعت و در صورتی که بسامد (فرکانس) ارجاع زیاد باشد مقدار ذخیره شده داشته باشد.

نوع ارتباط Relationship Type:

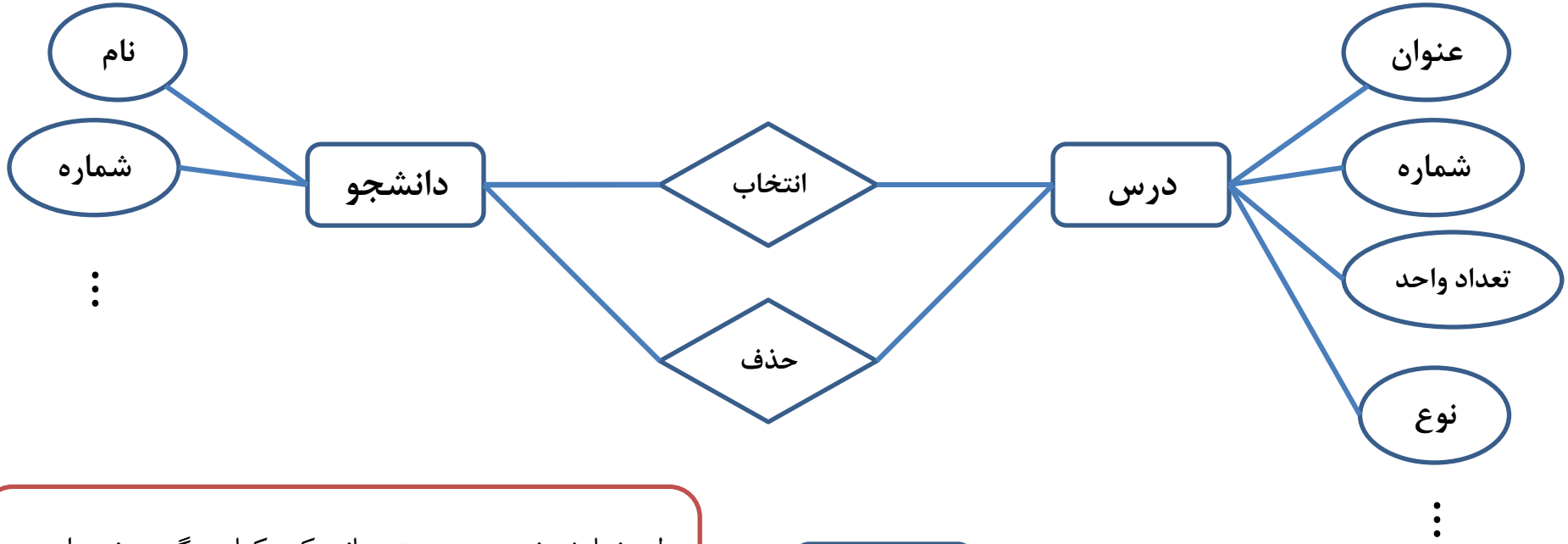
- رابطه، اندرکنش و یا تعامل بین $N \geq 1$ نوع موجودیت ← $N = 1$ ارتباط با خود - بازگشتی (self-relationship)

مثال

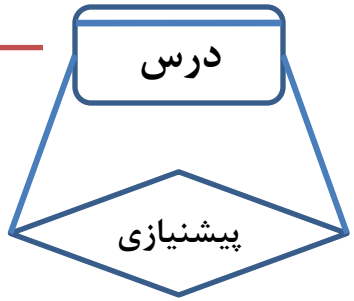
ارتباط نوع موجودیت‌های دانشجو و درس

- دانشجو درس را انتخاب می‌کند.
- دانشجو درس را حذف می‌کند.

ER مبنایی – نوع ارتباط (ادامه)



مثال ارتباط موجودیت با خود :



طرز نمایش نوع موجودیت زمانی که یکبار دیگر در نمودار ER آمده باشد. (به خاطر اجتناب از شلوغ شدن نمودار)

گفتگوی مفهوم پیشنیازی درس را به چند روش دیگر می‌توان مدل کرد؟

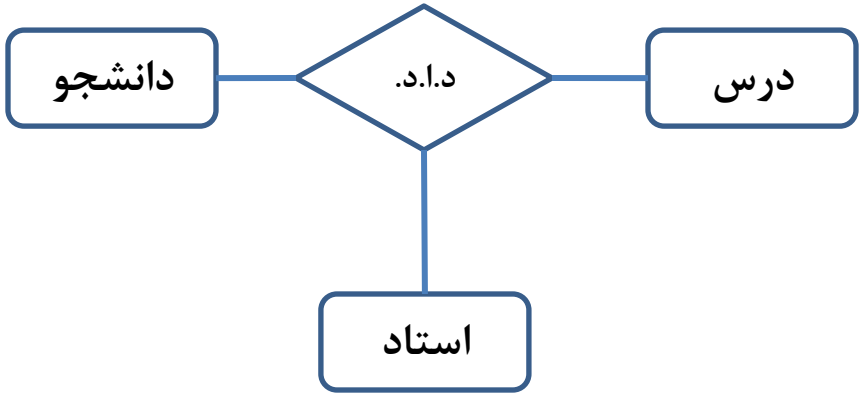
ER مبنایی – نوع ارتباط (ادامه)

- نوع ارتباط:
  - یک نام دارد.
  - یک معنا دارد.
  - شرکت کنندگانی (participants) دارد ($N \geq 1$).
  - به تعداد شرکت کنندگان **درجه** (arity or degree) ارتباط گویند.

N	اصطلاح
۱	ارتباط یگانی
۲	ارتباط دوگانی
۳	ارتباط سه‌گانی
n	ارتباط n-گانی (n-ary)

مثال: درجه یک و دو: مثال‌های پیش دیده

مثال: درجه سه: ارتباط درس، استاد، دانشجو



- تذکر: در عمل به ندرت $N \geq 4$ پیش می‌آید.

ER مبنایی – نوع ارتباط (ادامه)

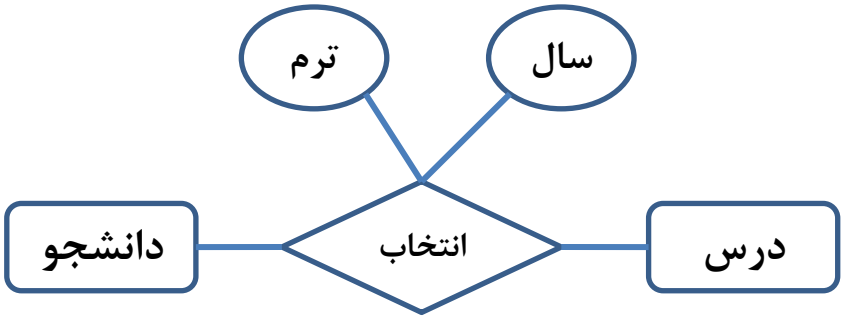
- مشارکت نوع موجودیت E در نوع ارتباط R
  - **الزامی** (کامل): هر نمونه از موجودیت E لزوماً در یک نمونه ارتباط R مشارکت دارد.
  - **غیر الزامی** (ناقص): حداقل یک نمونه موجودیت E وجود دارد که در هیچ نمونه ارتباط R مشارکت ندارد.
- الزامی بودن مشارکت از محدودیت‌های معنایی محیط، ناظر به نوع ارتباط است.

مثال: هر دانشجو لزوماً درسی را انتخاب می‌کند ولی همه دروس لزوماً توسط دانشجویان انتخاب نمی‌شوند.

ER مبنایی – نوع ارتباط (ادامه)

- **هر نوع ارتباط:**
  - می‌تواند صفت(هایی)، موسوم به صفت(های) توصیفی داشته باشد.

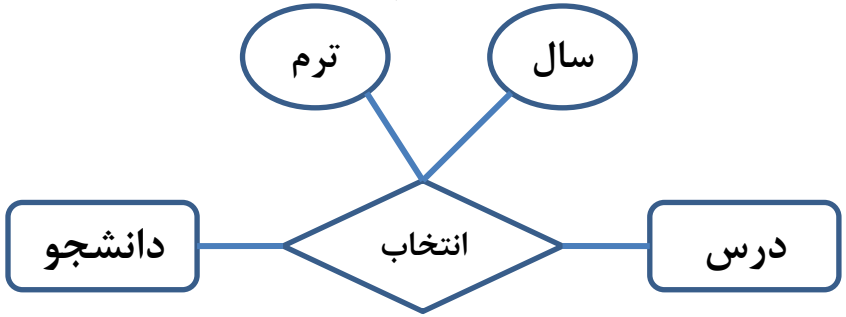
مثال: دانشجوی x درس y را در چه ترم و سالی انتخاب می‌کند؟



- **نکته مهم:** هر نمونه ارتباط باید توسط نمونه موجودیت‌های شرکت کننده در آن ارتباط به طور یکتا قابل شناسایی باشد [Silb2010].

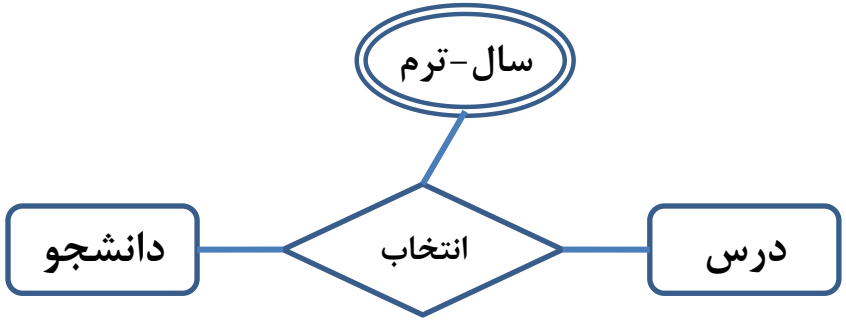
ER مبنایی – نوع ارتباط (ادامه)

- در مواردی که به ظاهر نتوانیم با نمونه موجودیت‌های شرکت کننده، یکتایی نمونه‌های یک ارتباط را تامین نماییم، می‌توانیم از صفت چندمقداری (برای رعایت نکته بیان شده) استفاده کنیم.



دانشجو	درس	سال	ترم
۹۲۱۰۱۲۳۵	۴۰۳۸۴	۹۴-۹۳	۲
۹۲۱۰۱۲۳۵	۴۰۱۳۲	۹۵-۹۴	۱

قابل درج نیست. چون ترکیب دانشجو و درس تکرار می‌شود و دیگر شناسه رابطه محسوب نمی‌شود.



دانشجو	درس	سال	ترم
۹۲۱۰۱۲۳۵	۴۰۱۳۲	۹۵-۹۴	۱
۹۲۱۰۱۲۳۵	۴۰۳۸۴	۹۴-۹۳	۲
		۹۵-۹۴	۱

قابل درج است؛ به عنوان مقادیر دیگر یک صفت مرکب چند مقداری.

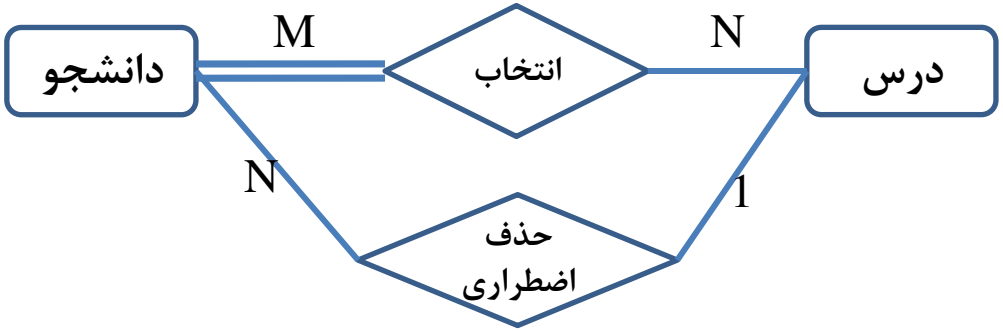
- **چندی ارتباط Multiplicity یا Cardinality Ratio:**
  - چندی ارتباط بین دو نوع موجودیت E و F عبارت است از چگونگی تناظر بین عناصر مجموعه نمونه‌های موجودیت E و عناصر مجموعه نمونه‌های موجودیت F.

تناظر
1:1
1:N
M:N

- اگر دو نوع موجودیت E و F را در نظر بگیریم:
  - در ارتباط یک به یک، یک نمونه از E حداکثر با یک نمونه از F ارتباط دارد و برعکس.
  - در ارتباط یک به چند (از E به F)، یک نمونه از E با n نمونه از F ($n \geq 1$) و در صورت مشارکت غیرالزامی، ($n \geq 0$) ارتباط دارد، ولی یک نمونه از F حداکثر با یک نمونه از E ارتباط دارد.
  - در ارتباط چند به چند، یک نمونه از E با n نمونه از F ($n \geq 1$) ارتباط دارد و برعکس.
- **نکته:** چندی نوع ارتباط چندگانی ($m>2$) عبارت است از تعداد نمونه‌های یک نوع موجودیت شرکت کننده در آن نوع ارتباط، وقتی که تعداد نمونه‌های m-1 نوع موجودیت دیگر شرکت کننده در نوع ارتباط را ثابت فرض کنیم.

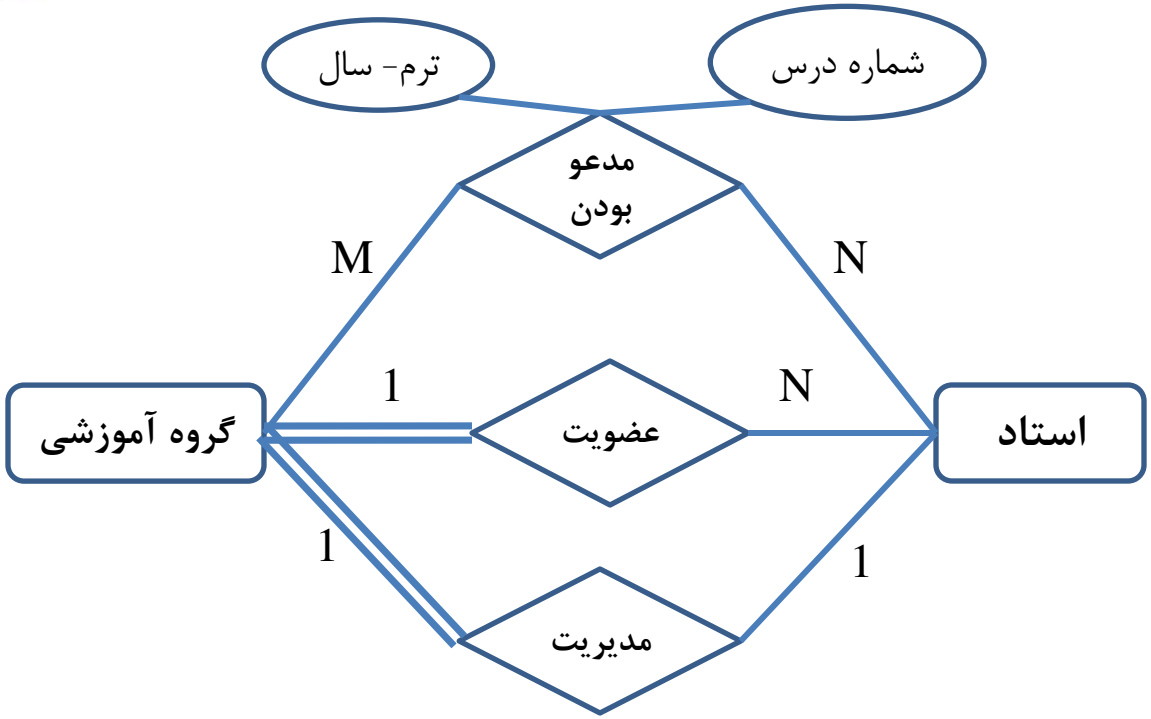
مثال

با فرض اینکه هر دانشجو چند درس می‌تواند انتخاب کند ولی فقط یک درس را می‌تواند حذف اضطراری کند، چندی ارتباطات به صورت زیر خواهد بود.

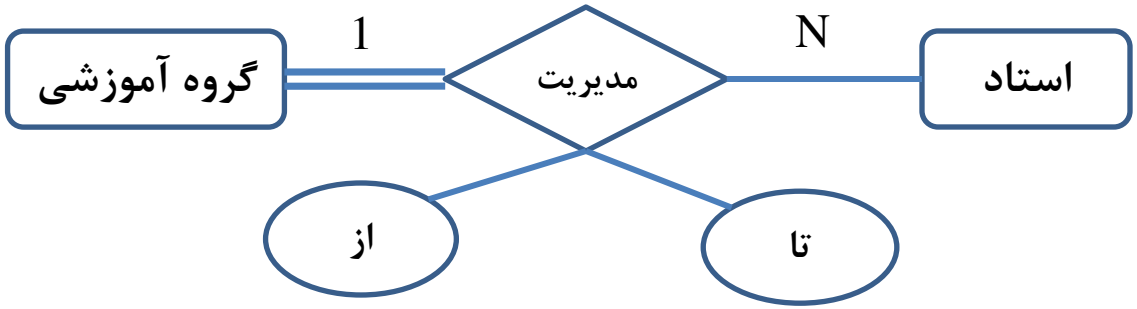


مثال

مثالی دیگر از چندی ارتباط

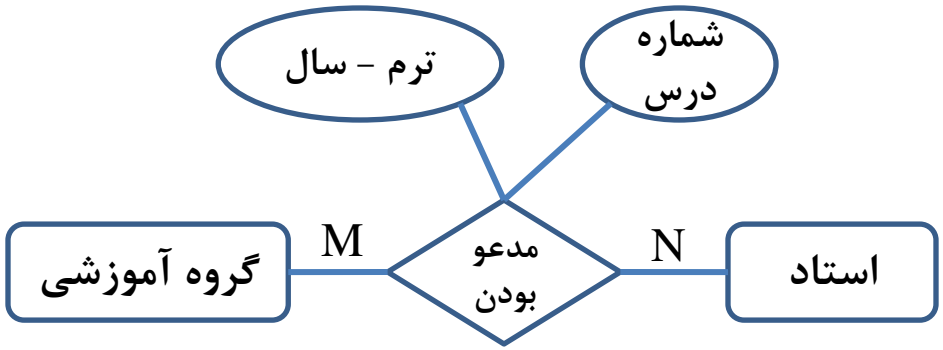


- **تذکر:** اگر به ارتباط صفت هایی از جنس زمان بدهیم، چندی ارتباط می‌تواند بسته به قواعد معنایی محیط تغییر کند.

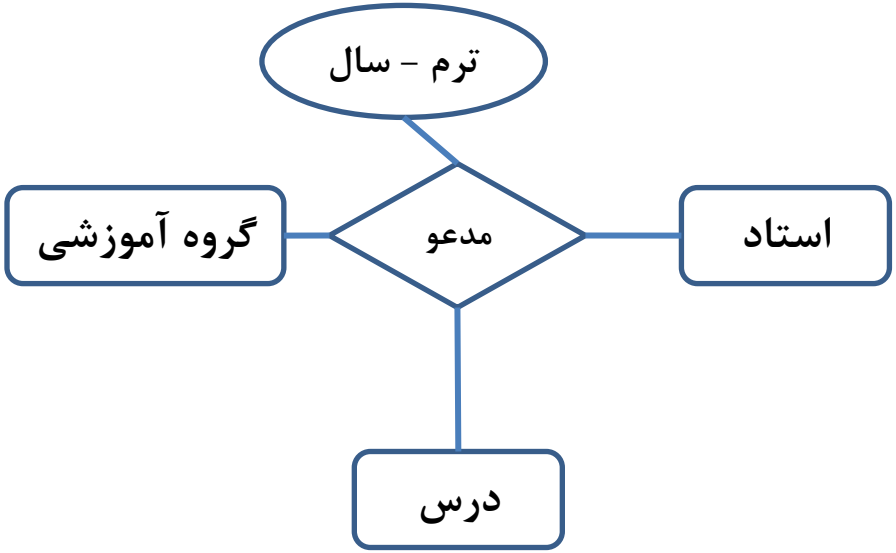


ER مبنایی – نوع ارتباط (ادامه)

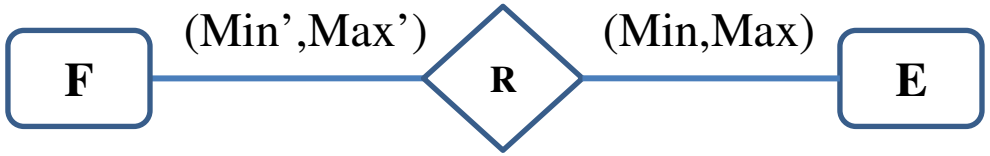
گونه‌های دیگر مدل کردن نوع ارتباط مدعو بودن چیست؟



- با استفاده از نوع ارتباط سه گانی:

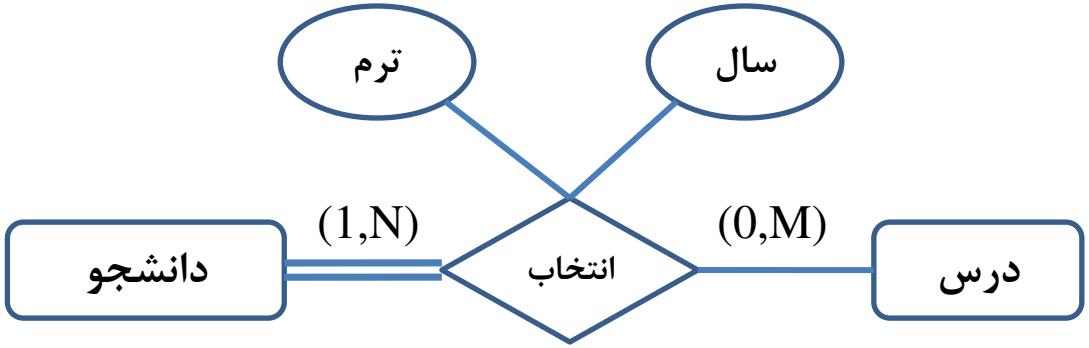


- **تذکر: طرز دیگر نمایش چندی ارتباط**



- هر نمونه e از نوع موجودیت E باید حداقل در Min و حداکشر در Max نمونه از ارتباط R شرکت داشته باشد.

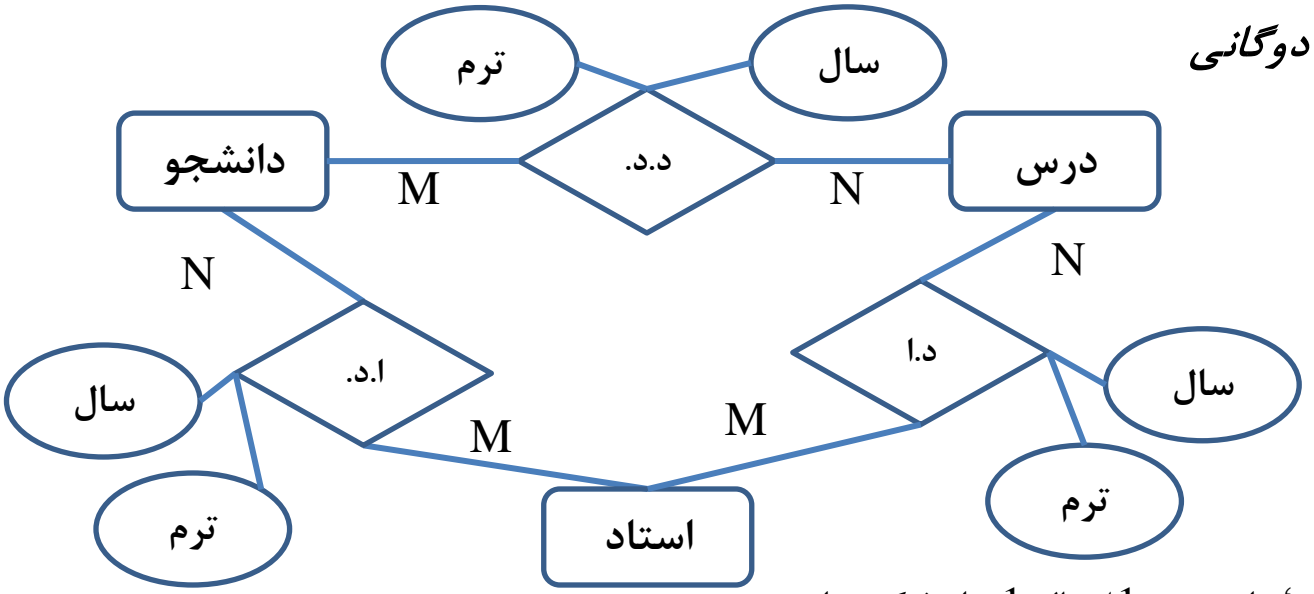
مثال: رابطه انتخاب درس توسط دانشجو



کنجکاوی: مزایای این روش نمایش چندی؟

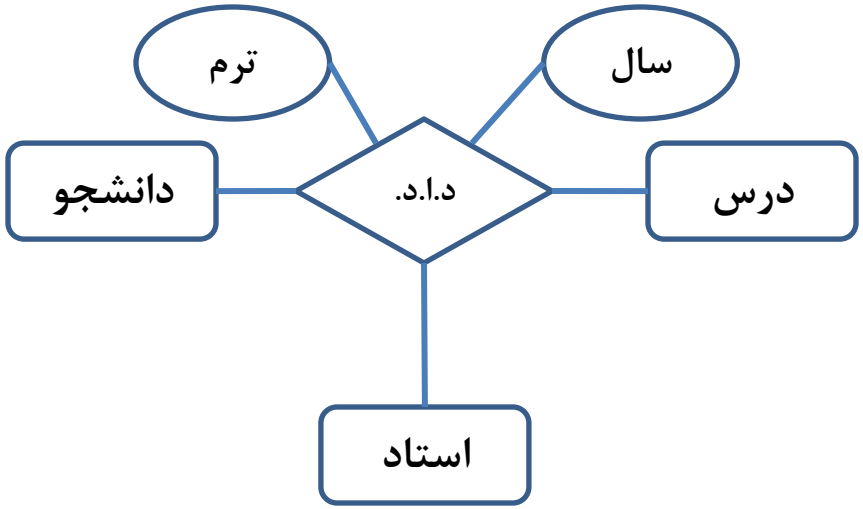
ER مبنایی – نوع ارتباط (ادامه)

- نکته مهم در مورد ارتباط بین سه نوع موجودیت:
  - *مدل یک: سه ارتباط دوگانی*



- سه فقره اطلاع:
  - دانشجو ‘s’ درس ‘c’ را در ترم t1 سال y1 اخذ کرده است.
  - استاد ‘p’ درس ‘c’ را در ترم t1 سال y1 ارایه کرده است.
  - دانشجو ‘s’ دانشجوی استاد ‘p’ است.
- از این سه فقره اطلاع لزوماً همیشه **نمی‌توان** نتیجه گرفت که دانشجو ‘s’ درس ‘c’ را با استاد ‘p’ گذرانده است.

- *مدل دوم: ارتباط سه گانی*

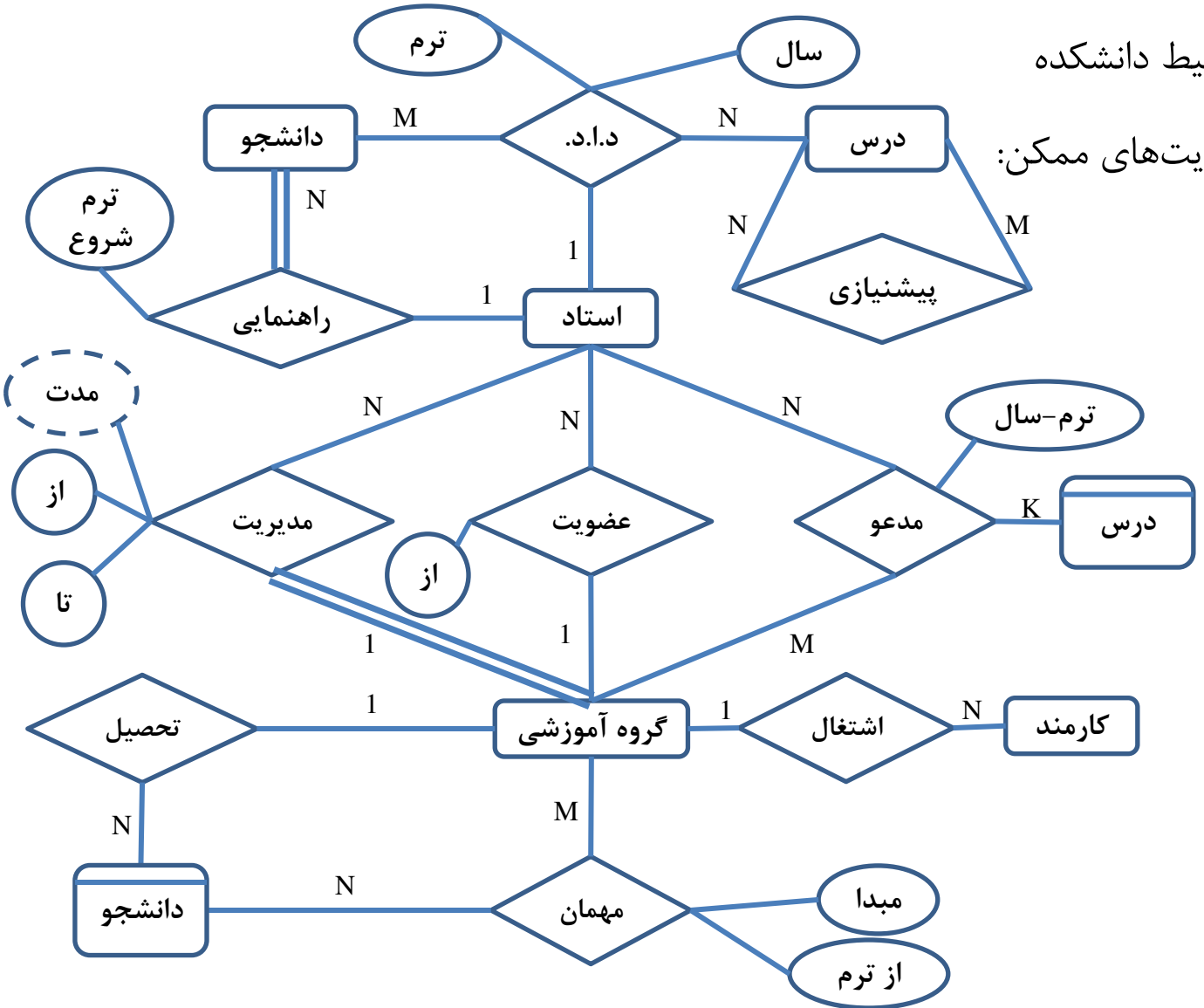


- در حالت سه ارتباط دوگانی اگر از فقره اطلاع‌های دوگانی، فقره اطلاع سه گانی را استنتاج کنیم در شرایطی که از لحاظ معنایی این استنتاج درست نباشد می گوییم دچار **دام پیوندی حلقه‌ای** شده‌ایم.

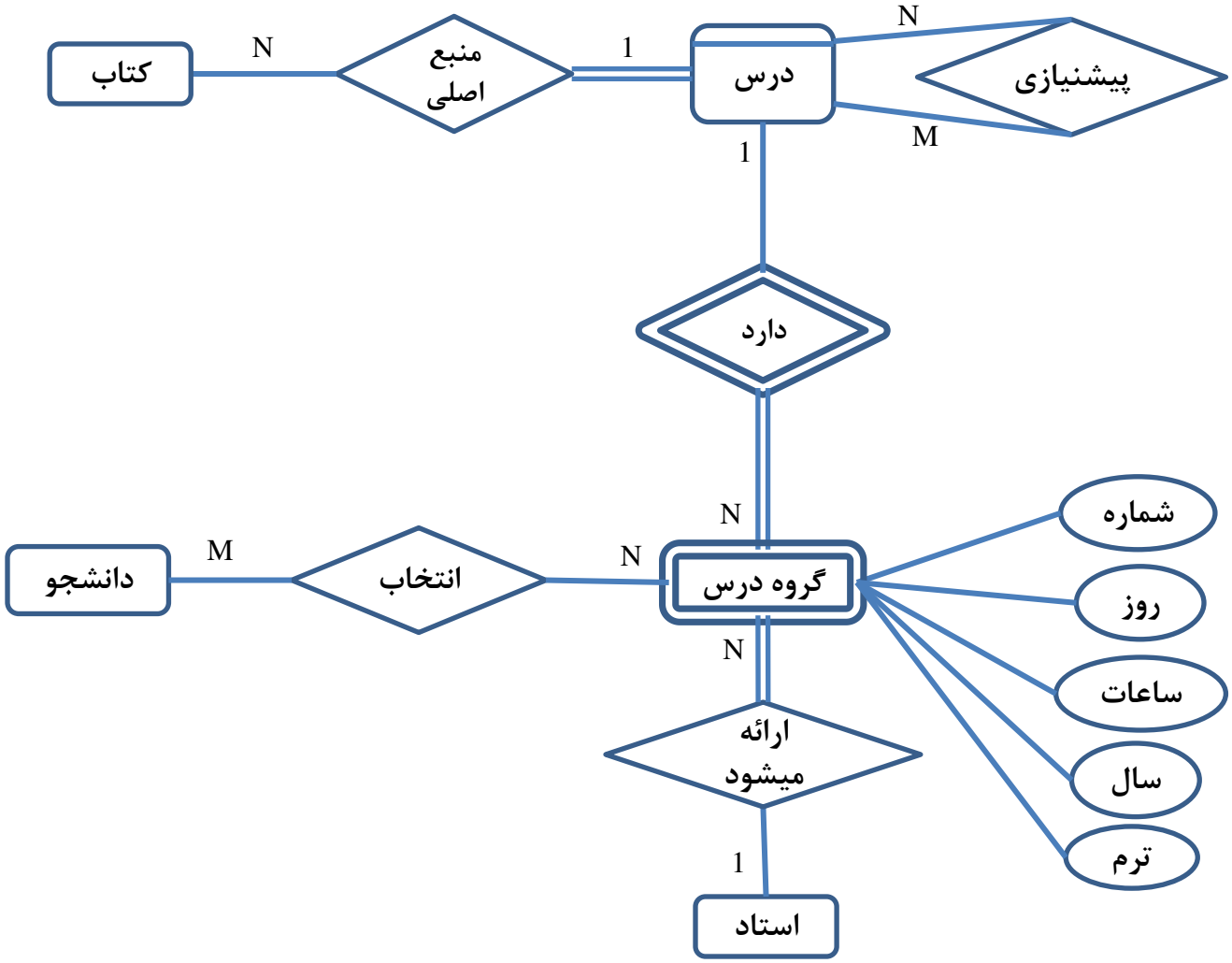
کنکاوی؟ انواع دیگر دام چیست؟ (دام چندشاخه (چتری)، دام گسل (شکافت)، ...)

مثال: محیط دانشکده

- مثال: فعالیت هایی از محیط دانشکده
- بعضی از نوع موجودیت‌های ممکن:
  - دانشجو
  - استاد
  - درس
  - کارمند
  - گروه آموزشی
  - کتاب
  - ...

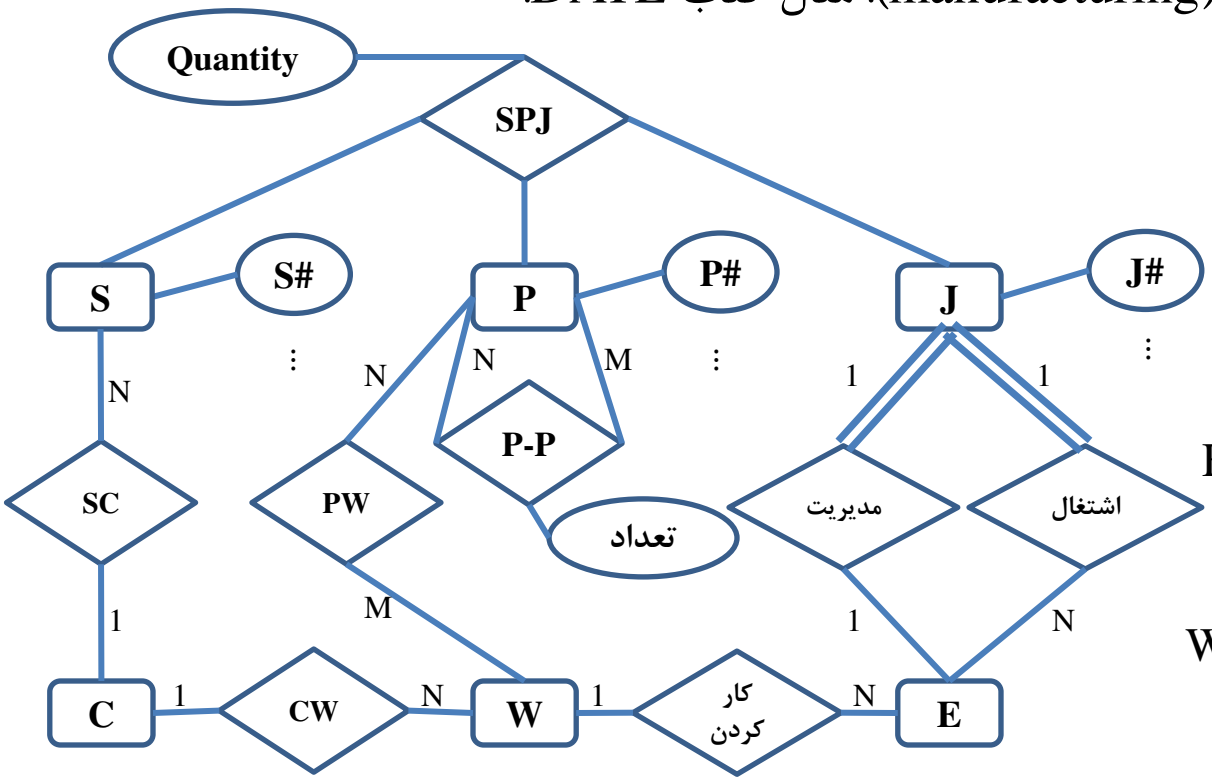


مثال: محیط دانشکده (ادامه)



مثال: محیط تولید

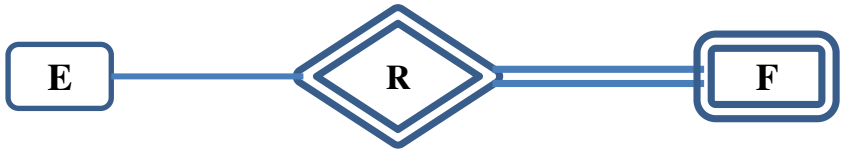
- مثال: محیط تولیدی-کارگاهی (manufacturing). مثال کتاب DATE.
- نوع موجودیت ها:
  - تولید کننده :S Supplier
  - نوع قطعه :P Part
  - پروژه :J Project
  - :E Employee
  - :C City
  - :W Warehouse



گسترش داده شود.

بحث تکمیلی : نوع موجودیت ضعیف

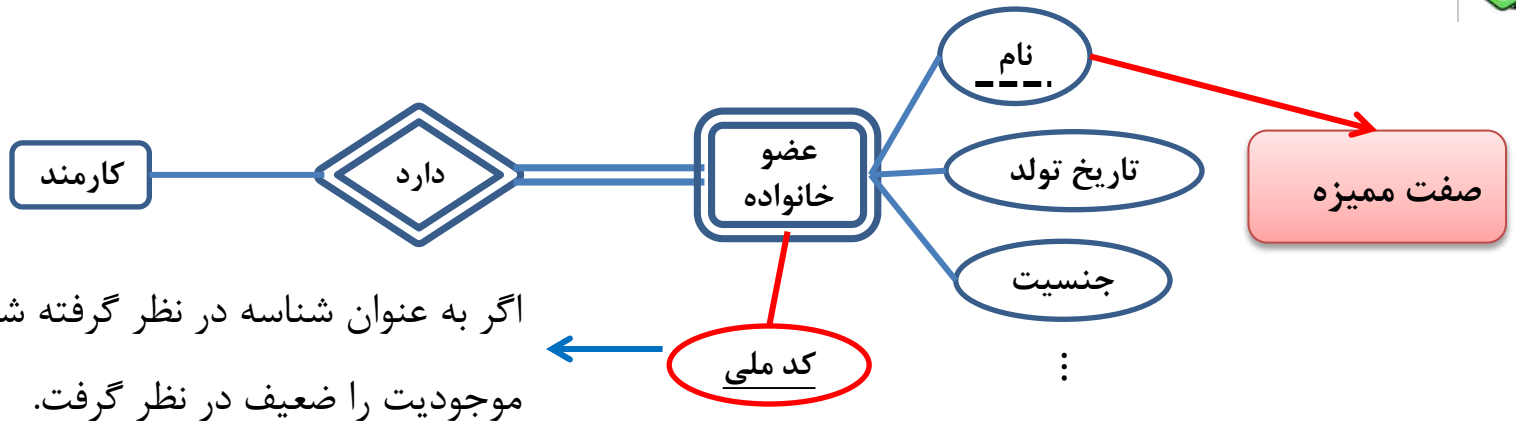
- **نوع موجودیت ضعیف:**
- نوع موجودیت F را ضعیف نوع موجودیت E گوییم هرگاه F با E «وابستگی وجودی» داشته باشد. (یعنی اگر E در مدلسازی مطرح نشود، F هم مطرح نباشد). علاوه‌براین نوع موجودیت ضعیف از خود شناسه ندارد.
- **طرز نمایش:**



- **تاکید:** قوی و ضعیف بودن نسبی است.
- نوع ضعیف از خود شناسه ندارد. بلکه از خود می‌تواند یک **صفت ممیزه-جداساز** (Discriminator) یا به عبارت دیگر یک **کلید جزئی** (Partial Key) دارد.
- **صفت ممیزه (کلید جزئی):**
  - صفتی که یکتایی مقدار دارد اما نه در تمام نمونه‌های نوع ضعیف بلکه در بین مجموعه تمام نوع ضعیف‌های وابسته به یک نمونه از نوع موجودیت قوی (به صورت نسبی یکتاست).
  - در عمل اگر یک نوع موجودیت وابستگی وجودی به نوع موجودیت دیگر داشته باشد و از خود شناسه داشته باشد دیگر ضعیف دیده نمی‌شود.

بحث تکمیلی : نوع موجودیت ضعیف (ادامه)

مثال: عضو خانواده به عنوان یک موجودیت ضعیف



اگر به عنوان شناسه در نظر گرفته شود دیگر نباید موجودیت را ضعیف در نظر گرفت.

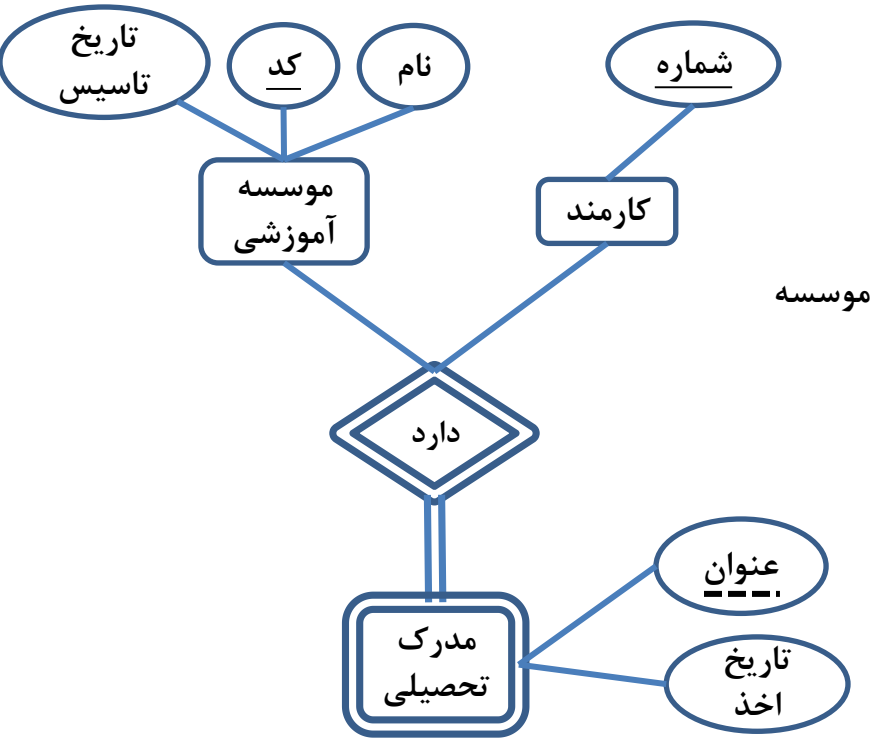
شماره کارمند	نام
۱۰۰	گلی سلی قلی
۲۰۰	ناجی تاجی سلی

بحث تکمیلی : نوع موجودیت ضعیف (ادامه)

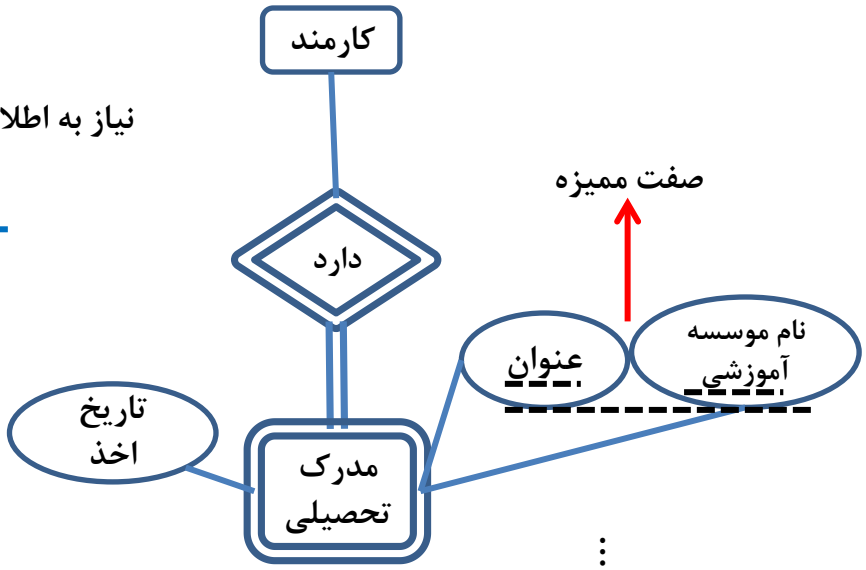
- به ارتباط قوی-ضعیف، **ارتباط شناسا** (Identifying Relation) گویند.
- مشارکت نوع ضعیف در ارتباط شناسا الزامی است.
- چندی ارتباط معمولا 1:N (در حالت خاص 1:1 تمرین: مثال قید شود).

بحث تکمیلی : نوع موجودیت ضعیف (ادامه)

- درجه ارتباط شناسا معمولا ۲ و گاه بیشتر است.



نیاز به اطلاعات بیشتر از موسسه آموزشی

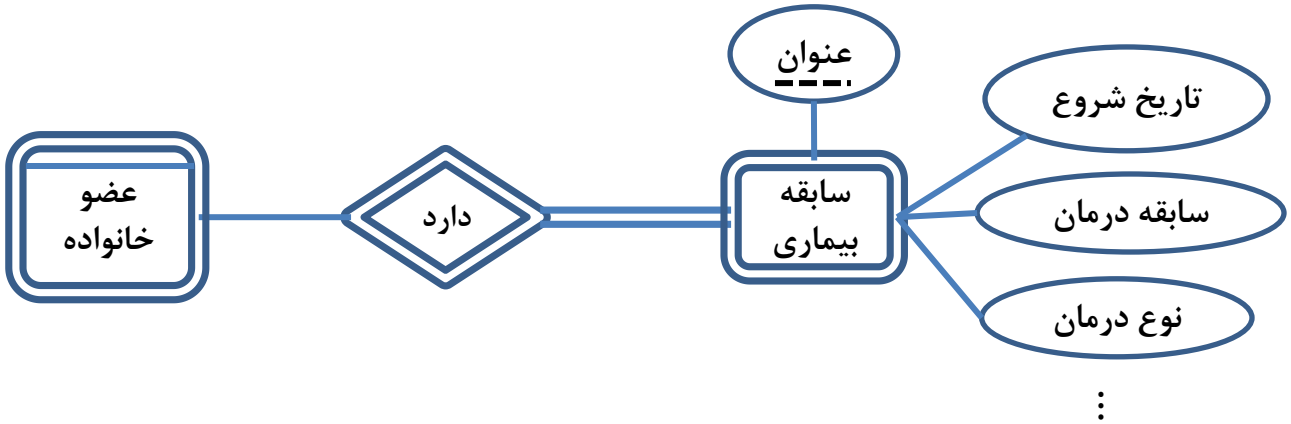


در اینجا مدرک تحصیلی وابستگی وجودی به بیش از یک موجودیت دارد.

آیا این محیط را می‌توان به گونه‌ای دیگر مدل کرد؟

بحث تکمیلی : نوع موجودیت ضعیف (ادامه)

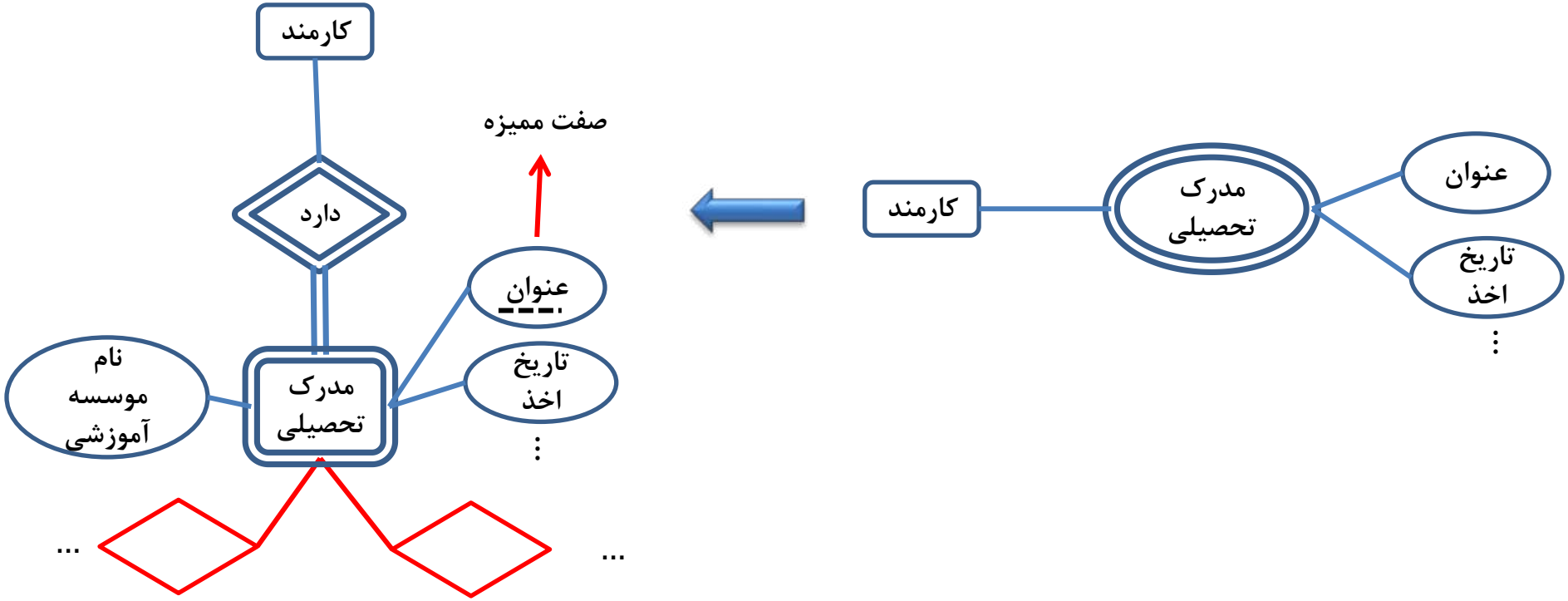
- نوع موجودیت ضعیف می‌تواند خود قوی برای نوع موجودیت ضعیف دیگر باشد.



اگر بخواهیم برای کارمند سابقه بیماری‌اش را نگه داریم چه کارکنیم؟

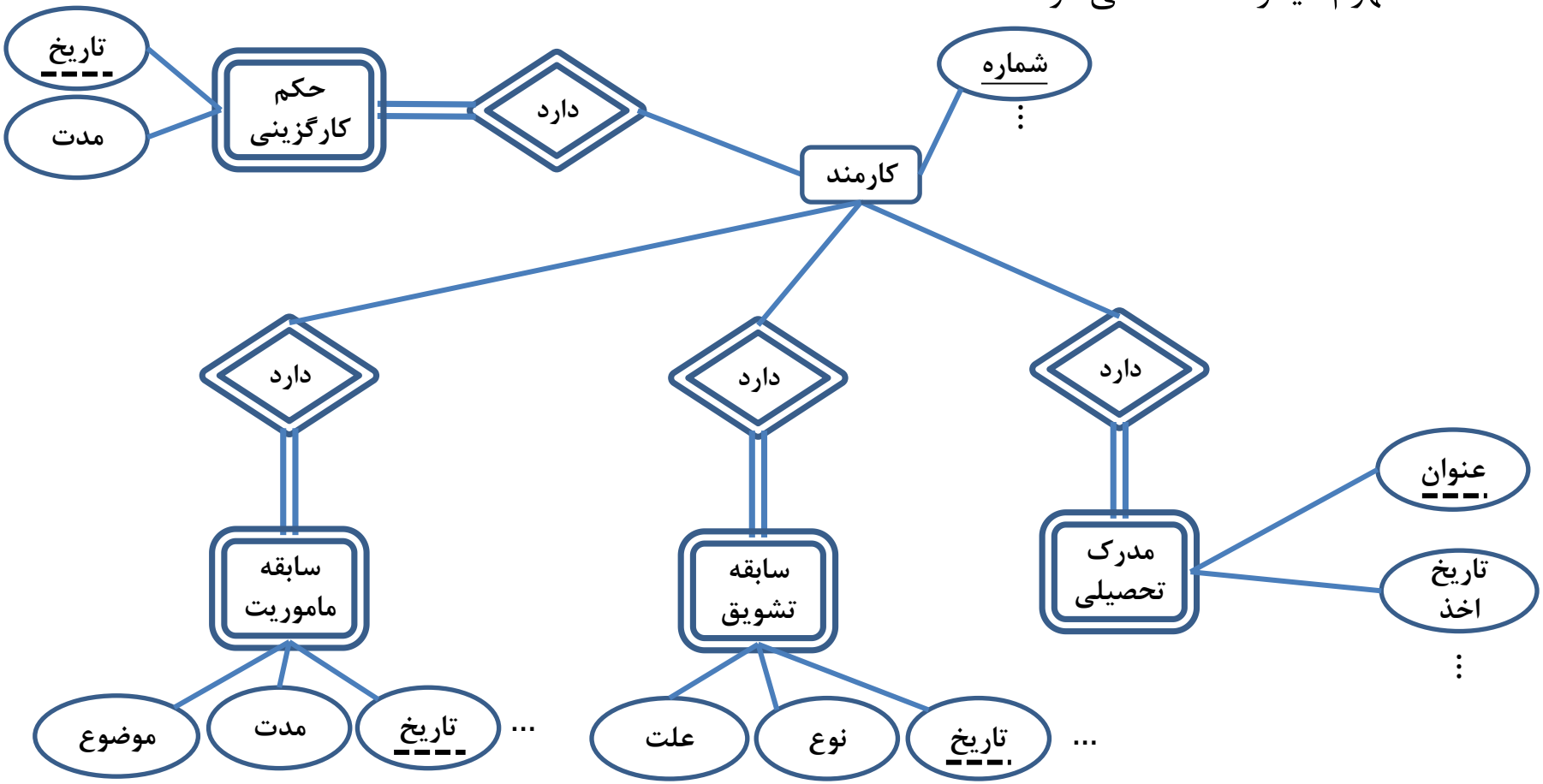
بحث تکمیلی : نوع موجودیت ضعیف (ادامه)

- صفت چند مقداری (به خصوص مرکب) را همیشه می‌توان با مفهوم نوع موجودیت ضعیف مدل کرد (نمایش داد) اما عکس این تکنیک توصیه نمی‌شود.
- **دلیل:** انعطاف پذیری مدل را از نظر گسترش پذیری کاهش می‌دهد، زیرا نوع ضعیف می‌تواند خود نوع ارتباط‌هایی داشته باشد با دیگر نوع موجودیت‌ها، اما وجود ارتباط با صفت معنا ندارد.



بحث تکمیلی : نوع موجودیت ضعیف (ادامه)

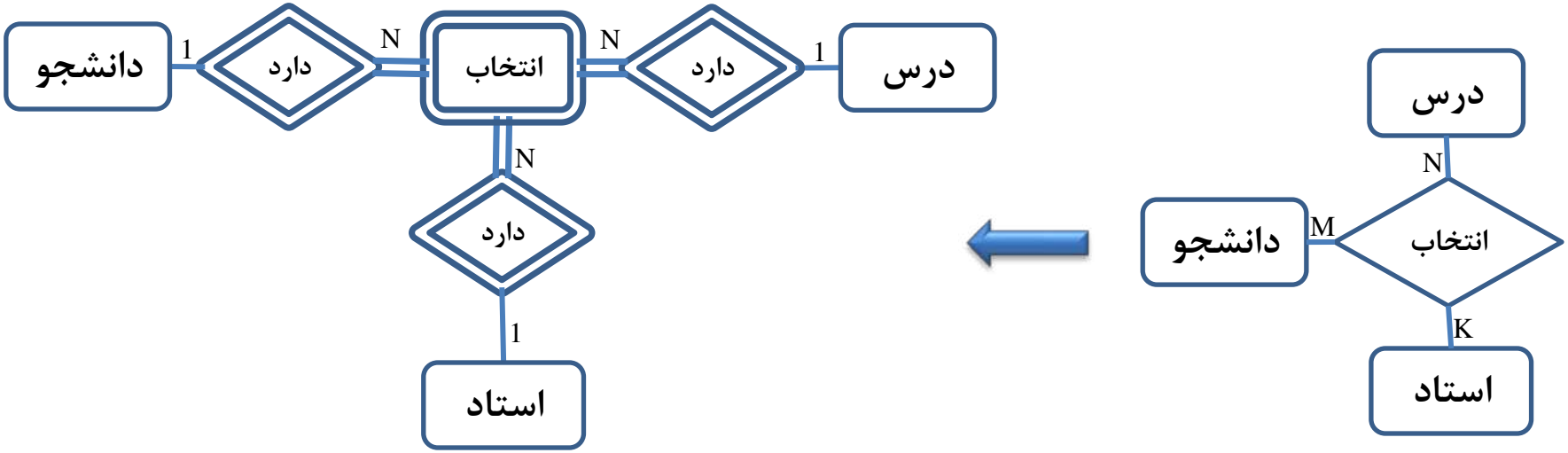
- مفهوم نوع موجودیت ضعیف به ویژه برای مدل کردن پدیده‌های تکرار شونده (در زمان) و وابسته به مفهوم دیگر استفاده می‌شود.



تبدیل ارتباط سه‌گانی به ارتباطات دوگانی

- از مفهوم نوع موجودیت ضعیف می‌توان برای تبدیل یک ارتباط سه‌گانی (یا n-گانی) به ارتباطات دوگانی استفاده کرد.
- اغلب ابزارهای طراحی مبتنی بر روش ER فقط ارتباطات دوگانی را پشتیبانی می‌کنند.

مثال: تبدیل رابطه سه‌گانه انتخاب به سه رابطه دوگانی.

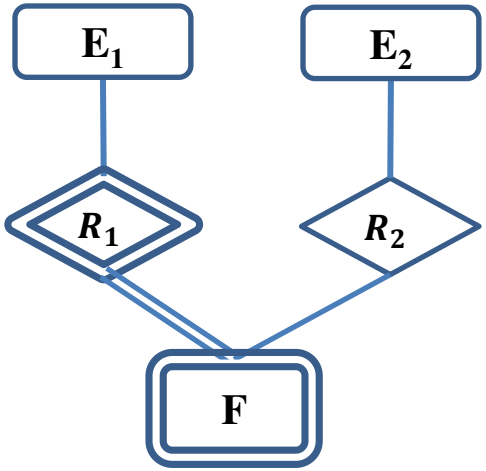


بحث تکمیلی : نوع موجودیت ضعیف (ادامه)

- یک نوع موجودیت ضعیف می‌تواند در یک نوع ارتباط دیگر با نوع موجودیت قوی دیگر شرکت داشته باشد.

مثال: رابطه بین گروه درسی و استاد در مثالهای پیشتر بیان شده.

کنکاوی: مثالی دیگر از مطلب فوق بیاورید.



نکات راهنمای تدوین نمودار ER

- مشکل تصمیم‌گیری در مورد اینکه یک مفهوم، نوع موجودیت در نظر گرفته شود یا صفت یا نوع ارتباط باید در یک فرآیند تدریجی در مدلسازی معنایی داده‌ها اصلاح شود.
  - اگر یک مفهوم، صفت به نظر آید، آنرا **صفت** می‌گیریم، اما اگر به نوع موجودیت دیگری ارجاع داشته باشد، آن را یک **نوع ارتباط** در نظر می‌گیریم.
  - اگر یک (چند) صفت به هم مرتبط (از لحاظ معنایی) در چند نوع موجودیت، مشترک باشند، آنها را به عنوان **صفات یک نوع موجودیت مستقل** منظور می‌کنیم.
  - اگر یک **نوع موجودیت**، تنها یک صفت داشته باشد و تنها با یک نوع موجودیت دیگر مرتبط باشد، آن را **صفت** در نظر می‌گیریم.
  - اگر مجموعه‌ای از صفات مستقلا قابل شناسایی نباشند، آن را به صورت **نوع موجودیت ضعیف** در نظر می‌گیریم.

- **EER**: Extended ER یا Enhanced ER
- ER مبنایی کمداشت‌هایی دارد در نمایش بعضی نوع ارتباطها (که بعدا در حیطه شیئ‌گرایی مطرح شد)

ER

+

بعض نوع ارتباطهای دیگر

۱- ارتباط "IS A"
«گونه ایست» / «هست یک»
← تکنیک تخصیص- ویژه نمایی Specialization
تکنیک تعمیم Generalization

۲- ارتباط "IS A PART OF"
← تکنیک تجزیه
تکنیک ترکیب

۳- ارتباط با ارتباط ← تکنیک تجمیع Aggregation

ارتباط "IS A"

- **ارتباط IS A:** ارتباط بین یک نوع موجودیت عام است با نوع موجودیت (های) خاص آن که بر

زیرنوع (Supertype) — زیرنوع (SubType)

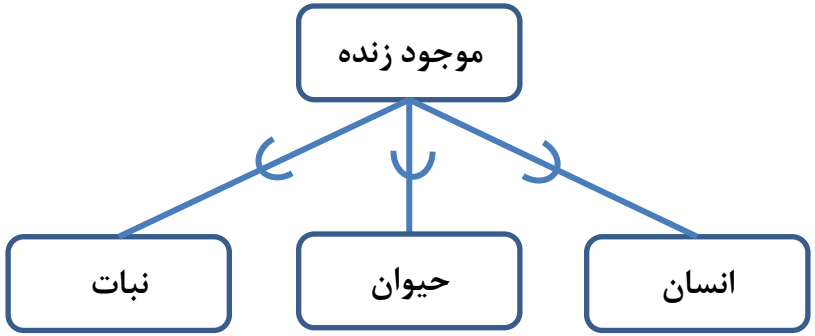
اساس یک ضابطه مشخص بازشناسی می‌شود.

صفت معرف — Defining Attribute

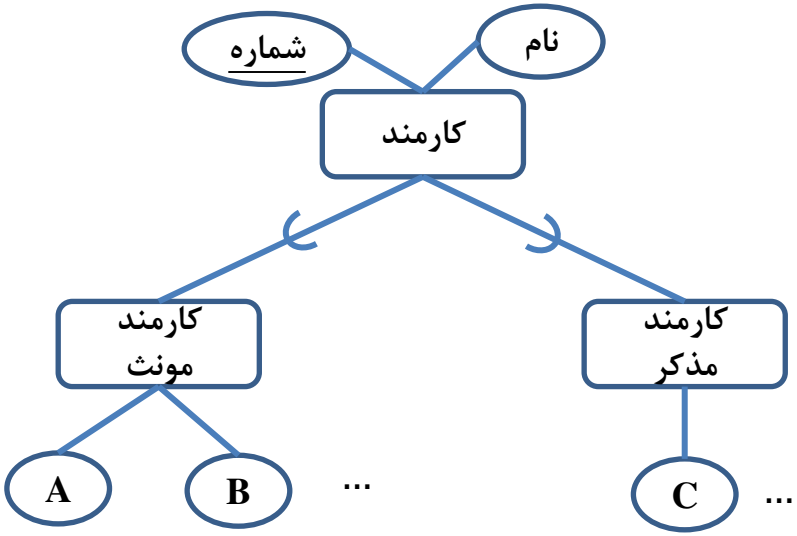
- طرز نوشتن: "F IS-A E"
- وقتی نوع های خاصِ یک نوع عام را بازشناسی می‌کنیم به آن تکنیک ویژه نمایی-تخصیص یا Specialization گوییم.
- عکس این تکنیک را تعمیم یا Generalization گوییم.

ارتباط "IS A" (ادامه)

مثال انواع موجودات زنده



مثال انواع کارمندان



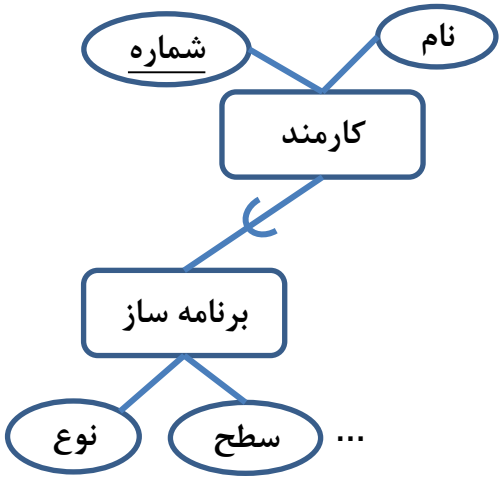
ارتباط "IS A" (ادامه)

- **نکات:**
  - **زیرنوع** مجموعه صفاتی دارد مشترک در تمام **زیرنوع‌ها**
    - درنتیجه **زیرنوع** تمام صفات **زیرنوع** را به ارث می‌برد (وراثت صفات از نوع ساختاری).
    - مفهوم ارث‌بری با تکنیک ارتباط IS-A مدلسازی می‌شود.
    - یادآوری! وراثت ممکن است ساختاری باشد یا رفتاری. در اینجا وراثت صفات، وراثتی ساختاری است.
  - **زیرنوع** مجموعه صفات خاص خود را هم دارد [حداقل یک صفت]
  - اگر m تعداد شاخه های تخصیص منشعب از یک **زیرنوع** باشد داریم: $\boldsymbol{m \geq 1}$

- **تخصیص**
  - ۱- **کامل:** تمام زیرنوع‌های (ممکن) زبرنوع در مدلسازی در نظر گرفته می‌شوند. بدین ترتیب هر نمونه از زبرنوع، جزء نمونه‌های حداقل یکی از زیرنوع‌ها است.
  - ۲- **ناقص:** تمام زیرنوع‌های (ممکن) زبرنوع در مدلسازی در نظر گرفته نمی‌شوند. هر نمونه از زبرنوع لزوما جزء نمونه‌های یکی از زیرنوع‌ها نیست.

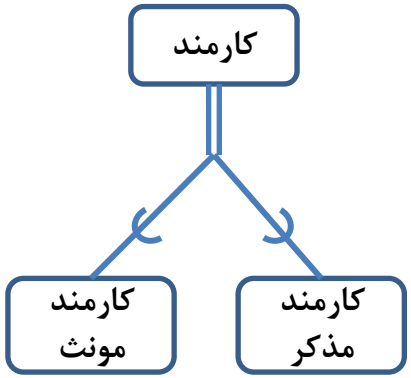
مثال

تخصیص ناقص: براساس مهارت کارمند فقط برنامه‌سازان را جدا کرده‌ایم. ممکن است کارمندی باشد که برنامه‌ساز نباشد.



مثال

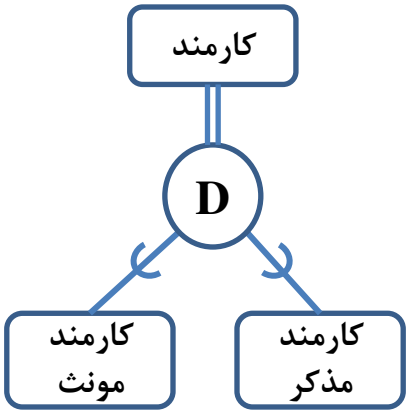
تخصیص کامل: هر نمونه کارمند یا مونث است یا مذکر.



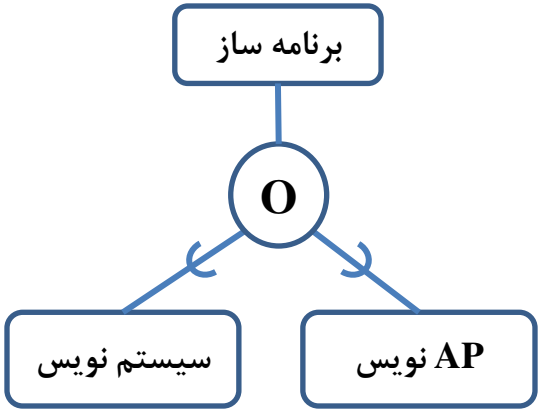
ارتباط "IS A" – تخصیص (ادامه)

- تخصیص
  - ۱- **مجزا:** یک نمونه از **زیرنوع** جزء مجموعه نمونه‌های حداکثر یک **زیرنوع** است.
  - ۲- **همپوشا:** یک نمونه از **زیرنوع** جزء مجموعه نمونه‌های حداقل دو **زیرنوع** است.

مثال تخصیص مجزا

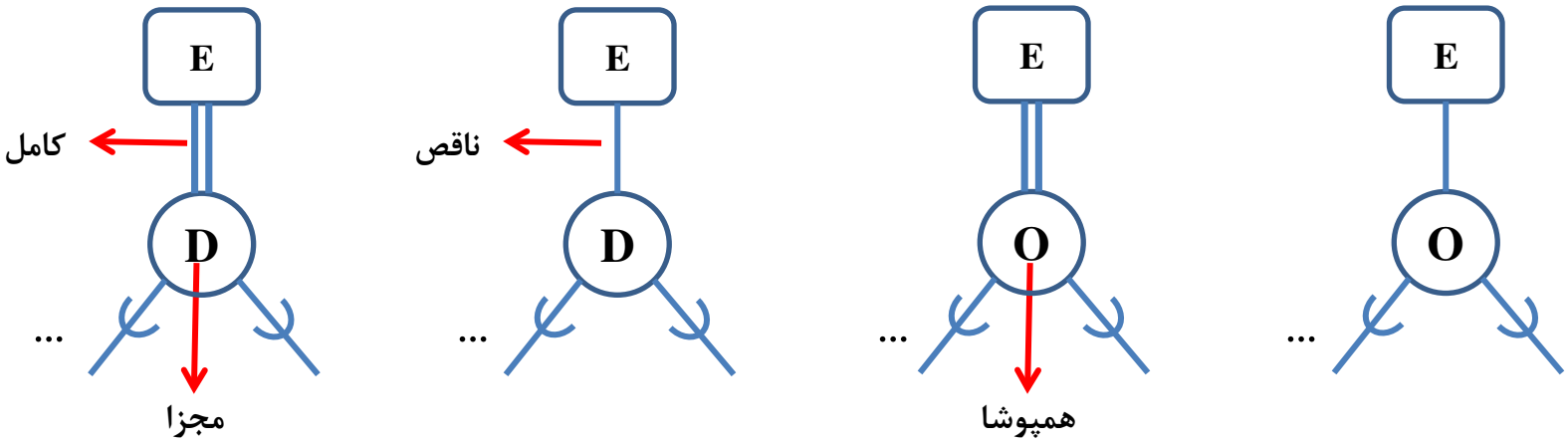


مثال تخصیص همپوشا



ارتباط "IS A" – تخصیص (ادامه)

- براساس این دو ویژگی چهارگونه تخصیص داریم:



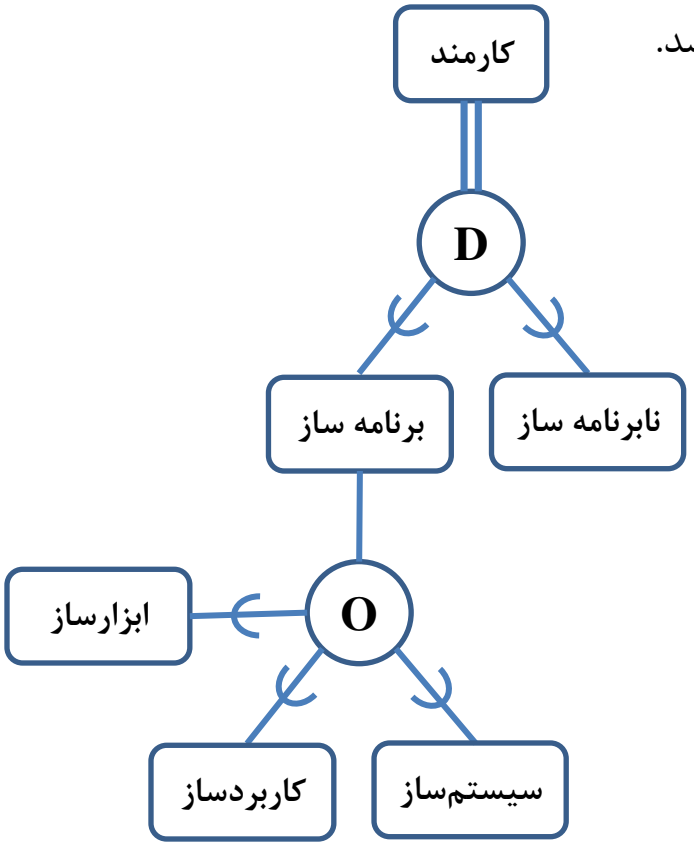
Disjoint

overlapping

ارتباط "IS A" (ادامه)

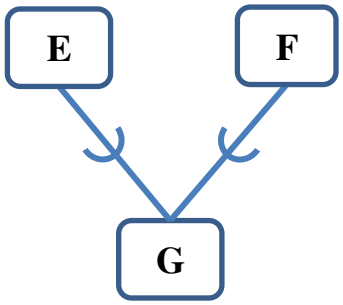
ادامه نکات:

- **زیرنوع** می‌تواند خود **زیرنوع‌هایی** داشته باشد.
  - یعنی ژرفای (عمق) درخت تخصیص می‌تواند بیش از یک باشد.



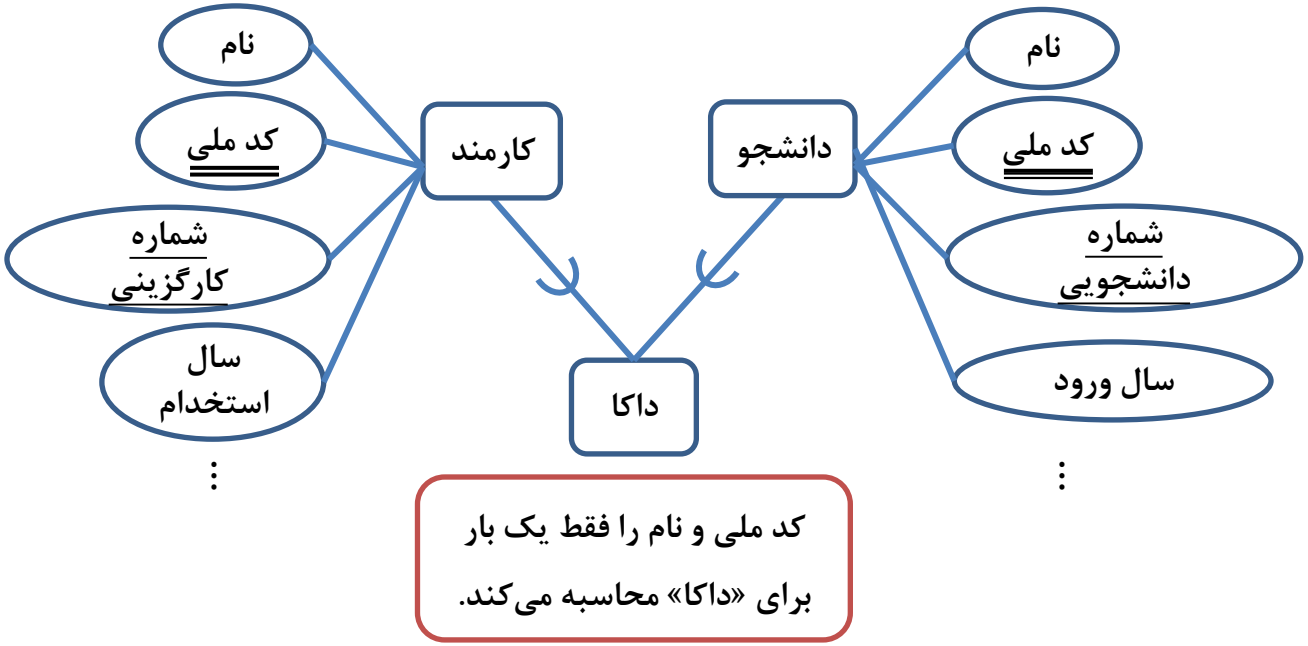
ارتباط "IS A" (ادامه)

- **زیرنوع** می‌تواند بیش از یک **زبرنوع** داشته باشد.
- G صفات را هم از E و هم صفات F را به ارث می‌برد
- **وراثت چندگانه (Multiple Inheritance)** را می‌توان اینگونه مدل کرد.



آیا G می‌تواند از خود نیز صفاتی داشته باشد؟

مثال: ارث‌بری چندگانه



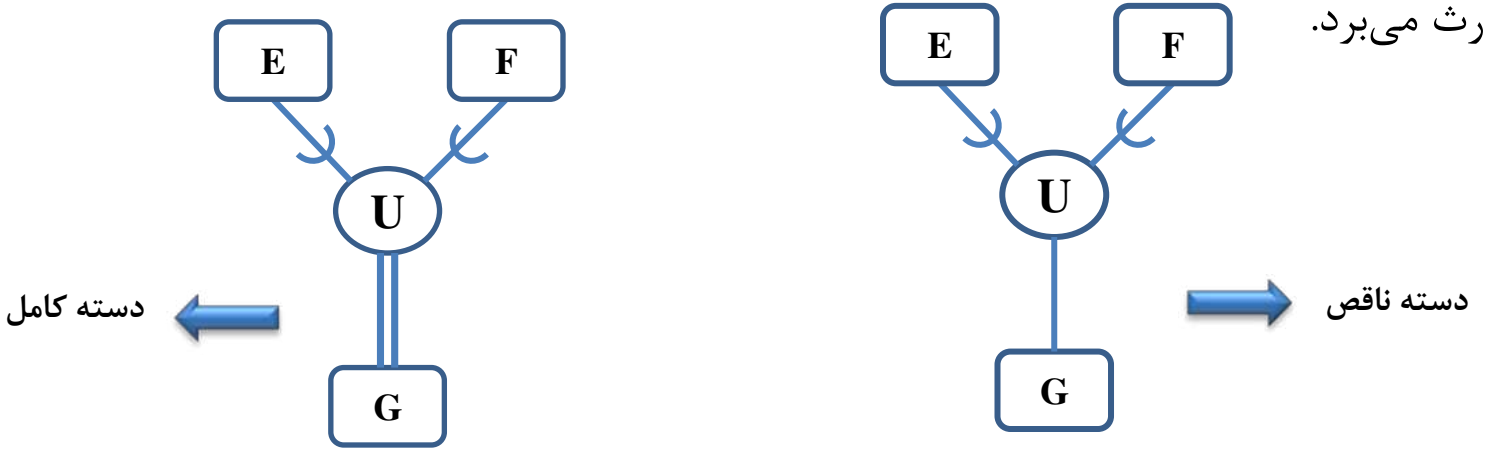
زیرنوع اجتماع (U-Type)

- **زیرنوع اجتماع (U-Type)** یا Category «دسته»
- **زیرنوع** موجودیت G را زیرنوع U-Type **زیرنوع‌های** E,F,... گوییم هرگاه در مجموعه نمونه‌های G نمونه‌هایی از E,F,... وجود داشته باشد. در واقع نمایانگر اجتماعی از نمونه‌ها از انواع مختلف است.

اگر همه نمونه‌ها ← **دسته کامل**

اگر بعض نمونه‌ها ← **دسته ناقص**

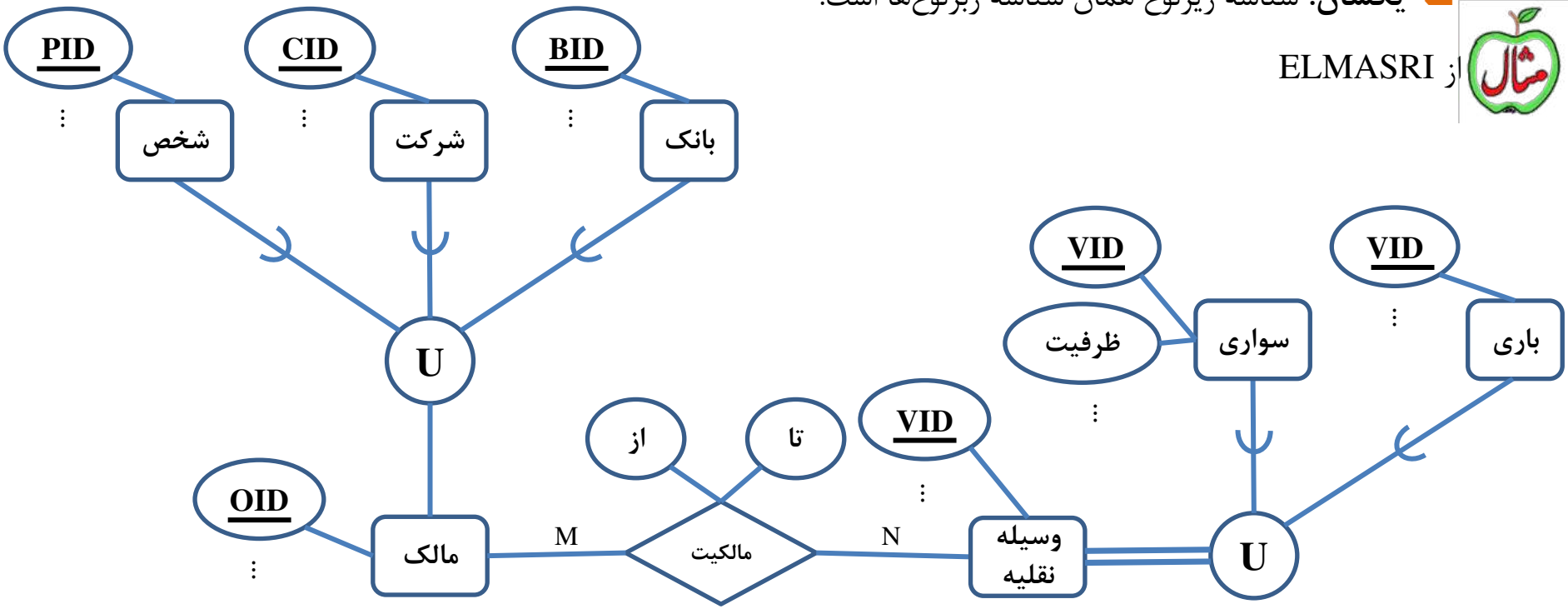
- یک نمونه از زیرنوع اجتماع (دسته)، بسته به اینکه از نوع کدام زبرنوع باشد، صفات همان زبرنوع را به ارث می‌برد.



زیرنوع اجتماع (ادامه)

- شناسه های زیرنوع ها می‌تواند از دامنه های متفاوت باشد.
- **متفاوت:** شناسه زیرنوع شناسه ای است که خود باید در نظر بگیریم.
- **یکسان:** شناسه زیرنوع همان شناسه زیرنوع‌ها است.

مثال از ELMASRI



کنکاوی در چه صورت مدلسازی با U-Type را می‌توان با تکنیک تخصیص (ویژه‌نمایی) معمولی مدل کرد؟ در چه شرایطی کدام یک بهتر است؟

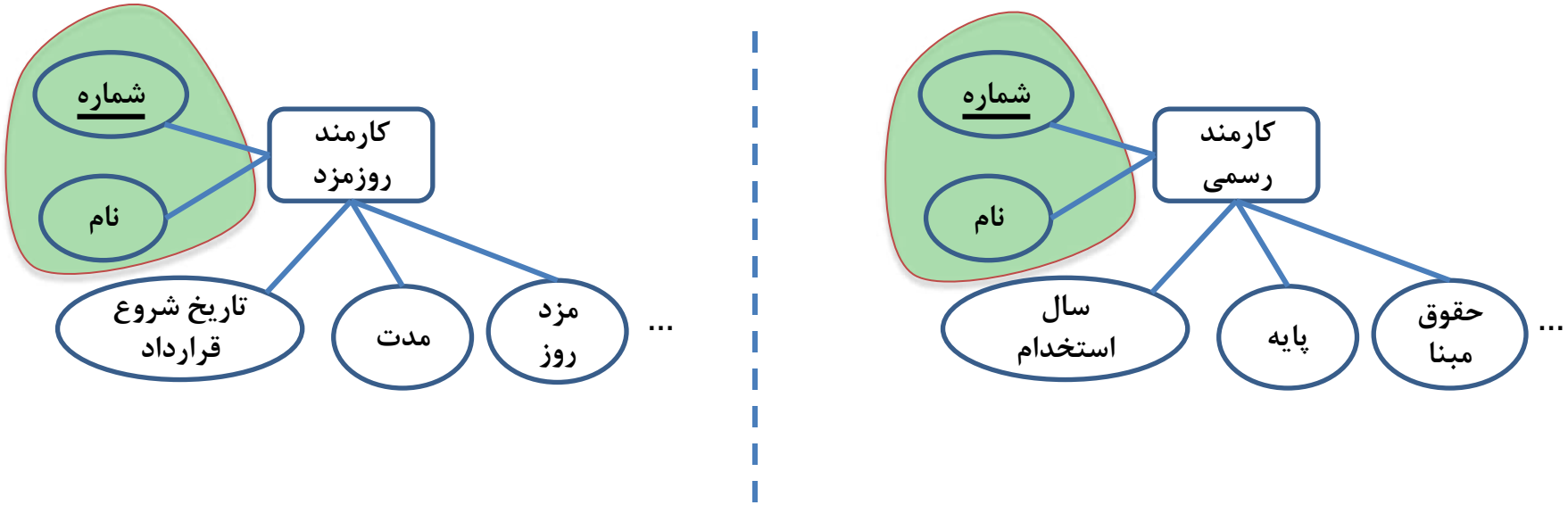
زیرنوع اجتماع (ادامه)

- تمرین : برای محیط با مفاهیم زیر، هم با U-Type و هم بدون U-Type یک مدلسازی ارایه دهید:
  - بانک - دانشگاه
  - شخص (دانشجو – استاد – کارمند و متفرقه)
  - حساب بانکی (کوتاه مدت – بلند مدت – قرض الحسنه و...)
  - عملیات واریز – برداشت - انتقال وجه

تعمیم (Generalization)

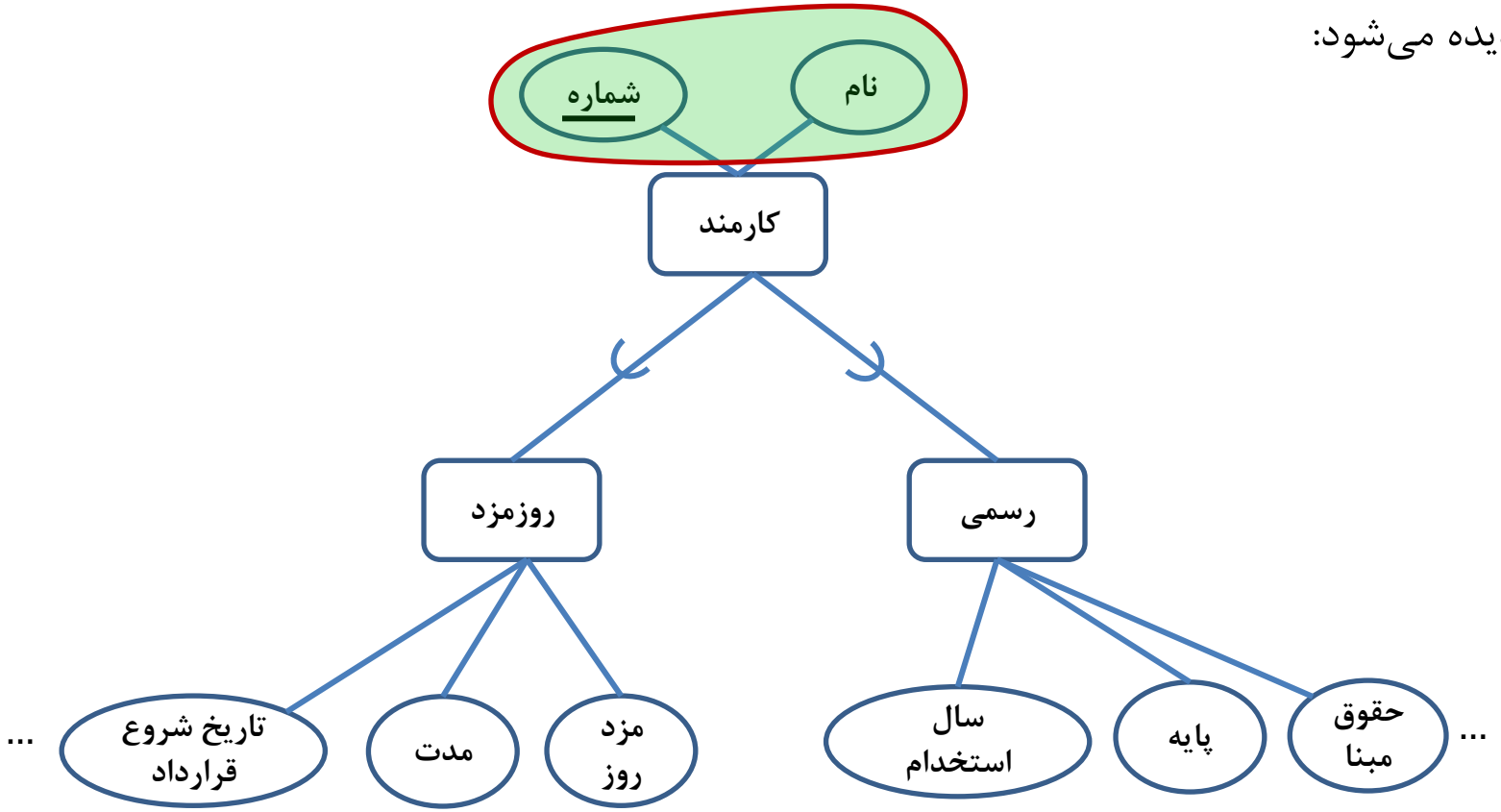
- **تعمیم** عبارت است از تشخیص یک نوع موجودیت جدید (در سطح انتزاع بالاتر) از روی [با داشتن] $n \geq 2$ نوع موجودیت از پیش دیده که ماهیتا از یک نوع باشند. (احیانا به منظور ادغام ERDهای جدا)

مثال: فرض: در یک مدلسازی یا در دو مدلسازی جدا برای دو زیر محیط:



مثال ادامه:

- یک نوع موجودیت (کارمند) در سطح انتزاعی بالاتر دیده می‌شود:



تعمیم (ادامه)

- **شرایط تعمیم:**
  - داشتن شناسه مشترک [یعنی از یک دامنه]
  - حداقل وجود دو نوع زیرنوع
  - هرچه صفات مشترک بیشتر، تعمیم توجیه پذیرتر است [شرطِ لازم نیست ولی شرطِ ارجحیت است].

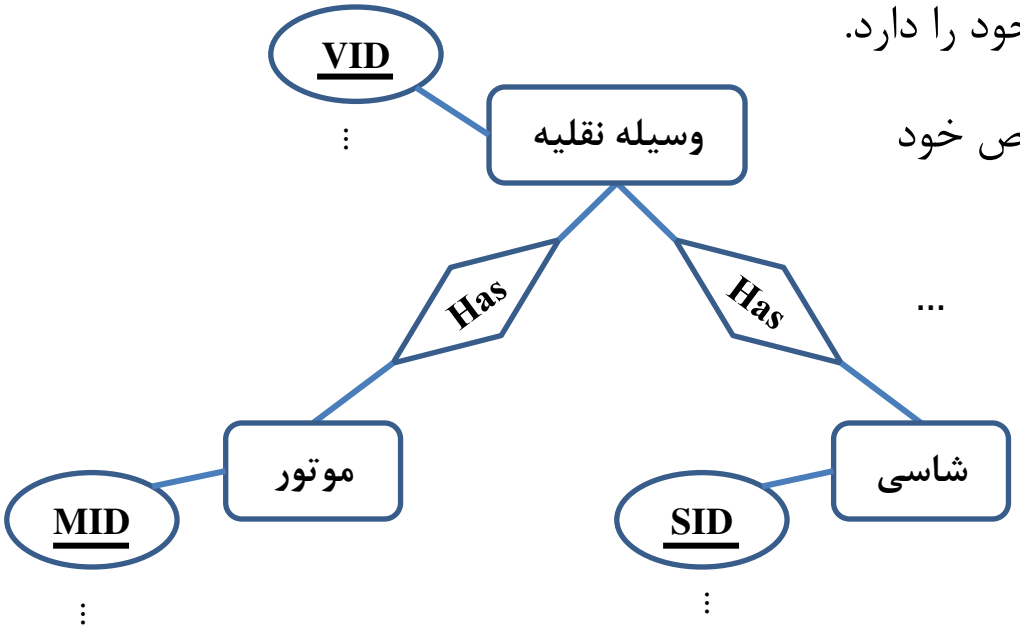
کنجاوی؟ ارتباط‌ها؟

ارتباط "IS-A-PART Of" یا "Has-A" یا "Contains"

- **تعریف:** ارتباط بین نوع موجودیت کل است با نوع موجودیت‌های جزء آن (تشکیل دهنده آن)
- **F** is a part of **E**
- E شامل F است.
- E دارد F.
- نکته: نوع کل مجموعه صفات خاص خود را دارد.
- نکته: نوع جزء هم مجموعه صفات خاص خود را دارد [از جمله شناسه].

مثال

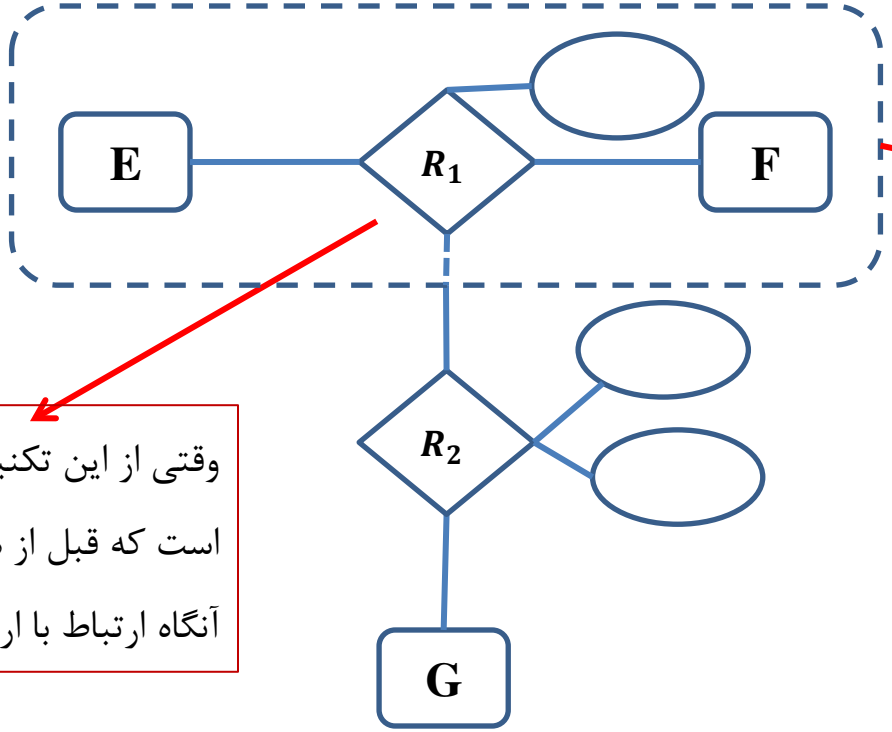
ارتباط شاسی و موتور با وسیله نقلیه



ارتباط "IS-A-PART Of" (ادامه)

- **تفاوت های نوع ضعیف با نوع جزء:**
  - نوع جزء از خود شناسه دارد ولی نوع ضعیف نه.
  - با حذف نوع کل لزوما نوع جزء حذف نمی‌شود (به عبارتی وابستگی وجودی لزوما نداریم.)
  - ...؟
- در ارتباط "IS-A-PART Of" ←
  - **تکنیک تجزیه:** دیدن نوع موجودیت‌های جزء از روی نوع موجودیت کل
  - **تکنیک ترکیب:** دیدن نوع موجودیت کل از روی اجزاء

- **تکنیک تجمیع (Aggregation):** دیدن $N \geq 1$ نوع موجودیت شرکت کننده در ارتباط R، به صورت یک نوع موجودیت **انتزاعی**: به منظور مدلسازی ارتباط با ارتباط (به ویژه زمانی که نوع ارتباط R صفاتی هم داشته باشد).
- ارتباط با ارتباط حیطه معنایی خاص خود را دارد.



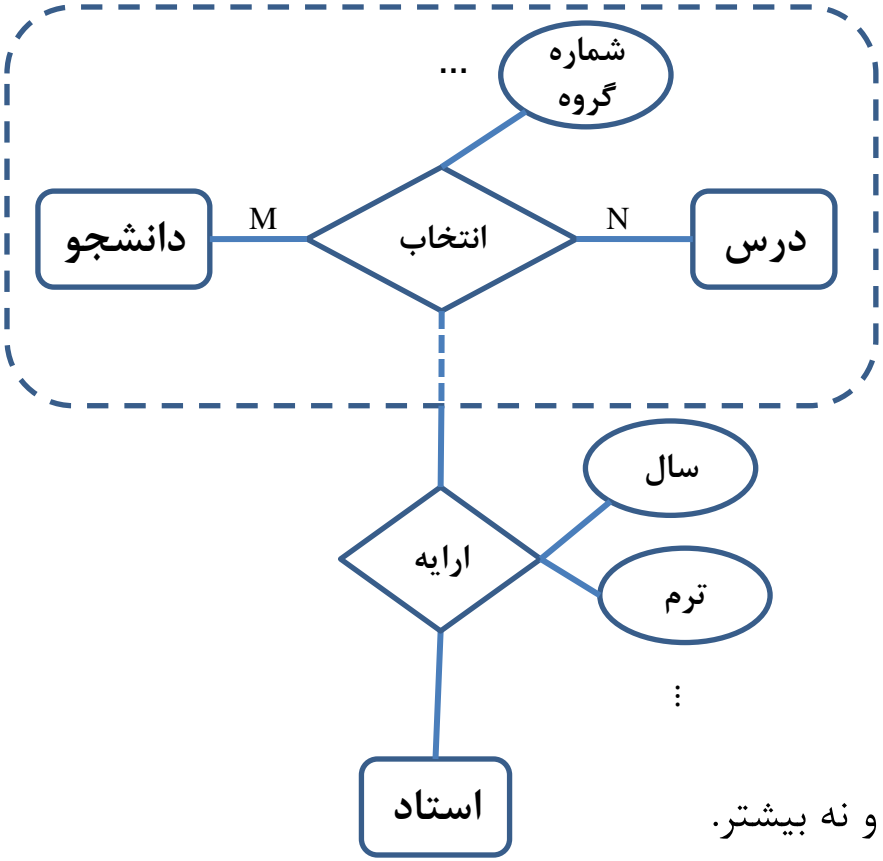
دیدن موجودیتهای دخیل در ارتباط R_1 به صورت یک موجودیت انتزاعی

وقتی از این تکنیک استفاده می‌شود، معنایش این است که قبل از هر چیز به ارتباط R_1 نیاز است. آنگاه ارتباط با ارتباط مطرح شده است.

ارتباط با ارتباط (ادامه)

- معمولا از این تکنیک زمانی استفاده می‌شود که چندی ارتباط M:N باشد. چرا ؟

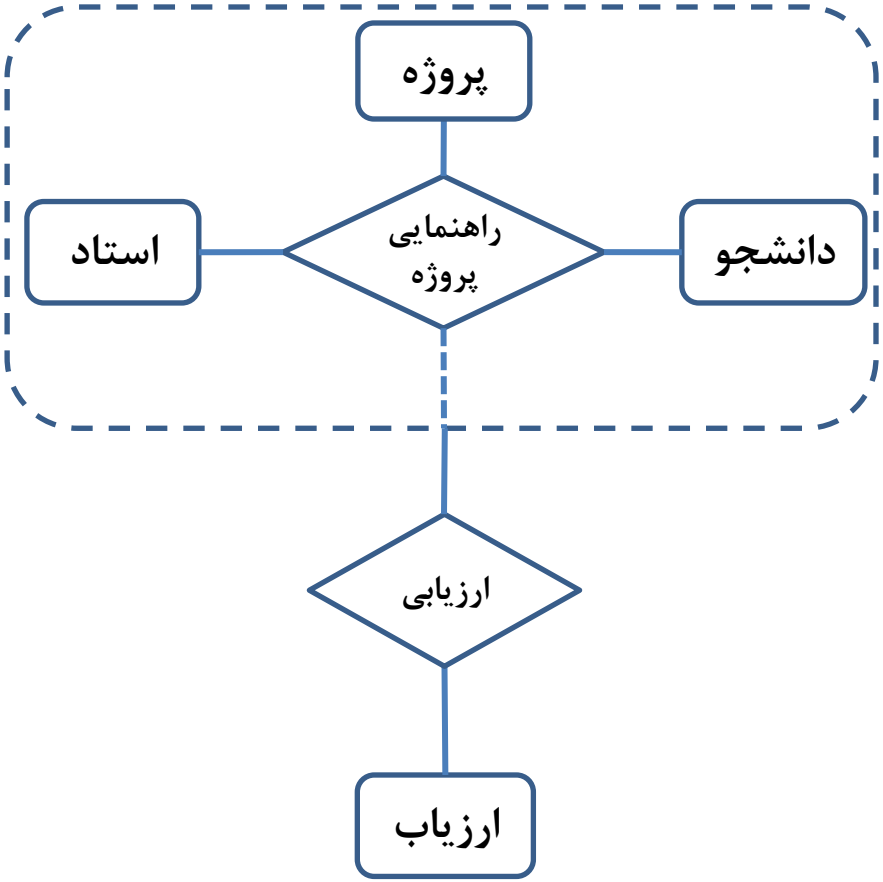
مثال: طرز دیگر مدلسازی برای برای محیط دانشجو – درس – استاد:



- **نکته:** هر Aggregation برای یک ارتباط است و نه بیشتر.

ارتباط با ارتباط (ادامه)

مثال: ارزیابی راهنمایی پروژه پژوهشی دانشجو توسط استاد



- **نکات زیر بررسی شود:**
  - ویژگی های عمومی روش مدلسازی
  - کمداشت های روش [E]ER
  - تناظر بین مفاهیم روش [E]ERو روش UML [در نمودار رده Class diagram]

مراحل مدلسازی معنایی داده‌ها

۱- مطالعه، تحلیل و شناخت محیط

۲- برآورد خواسته‌ها و نیازهای اطلاعاتی و پردازشی همه کاربران ذیربط محیط (مهندسی نیازها) و تشخیص محدودیت‌های معنایی و قواعد فعالیت‌های محیط

۳- بازشناسی نوع‌موجودیت‌های مطرح و تعیین وضع هر نوع‌موجودیت (قوی یا ضعیف بودن آن)

۴- تعیین مجموعه صفات هر نوع‌موجودیت، میدان و جنبه‌های هر صفت

۵- بازشناسی نوع‌ارتباط‌های بین نوع‌موجودیت‌ها، تشخیص الزامی بودن یا نبودن مشارکت در آنها و تشخیص چندی هر ارتباط

۶- رسم نمودار ER (یا EER) به صورت واضح، خوانا و حتی‌الامکان با کمترین افزونگی

۷- فهرست کردن پرسش‌هایی که پاسخ آنها از نمودار به دست می‌آید (بر حسب گزارش‌های مورد نیاز و کلا نیازهای داده‌ای کاربران)

۸- وارسی مدلسازی انجام شده، برای اطمینان از پاسخگو بودن به نیازهای کاربران.

یکپارچه‌سازی نمودارهای جزیی

- گاه به علت وسعت محیط عملیاتی و تعدد کاربران آن لازم است مدلساز به ازای هر زیرمحیط و یا حتی یک کاربر نمودار ER رسم کند.
- در این صورت نیازمند **ادغام و یکپارچه‌سازی نمودارهای ER** هستیم.
- در ادغام چند نمودار ER باید به تعارض‌های (ماهیتا معنایی) بین نمودارها توجه کرد. از جمله موارد زیر:
  - مدلهای نایکسان برای ایده واحد
  - تعارض در نامگذاری یک مفهوم (از لحاظ معنایی) واحد (دو موجودیت Car و Automobile برای اتومبیل)
  - تعارض معنایی دو مفهوم ظاهرا یکسان (دو موجودیت با عنوان Student؛ یکی به معنای دانشجو و دیگری به معنای دانش‌آموز)
  - تعارض در میدان صفت‌ها
  - تعارض در محدودیت‌ها
- تحلیل این تعارض‌ها قبل از تصمیم‌گیری درباره ادغام ERها باید انجام شود.

پرسش و پاسخ . . .

amini@sharif.edu

به نام آنکه جان را فکرت آموخت

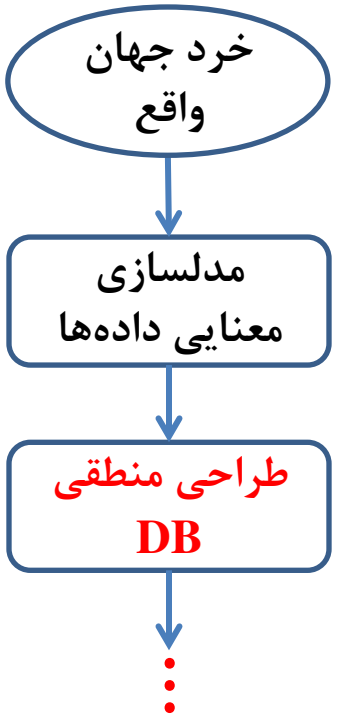
بخش سوم :
طراحی منطقی

مرتضی امینی

نیمسال دوم ۹۴-۹۵

(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)

- مدلسازی داده‌ها می‌تواند در سطوح انتزاعی مختلفی صورت پذیرد.
- سطح پایین‌تر از سطح مدلسازی معنایی داده‌ها، سطح طراحی منطقی است.



- **سطح طراحی منطقی:** برای نمایش پایگاه داده‌ها در این سطح از مفاهیمی استفاده می‌شود که مستقل از مفاهیم محیط فایلینگ پایگاه داده‌ها است.

طراحی منطقی DB (ادامه)

- بحث مقدماتی: دیدگاه کاربردی [و نه تئوریک]
- برای طراحی منطقی پایگاه داده‌ها (و همچنین عملیات در DB و کنترل DB) هم امکان خاصی لازم است: یک **مدل داده (DM)**، که شامل یک **ساختار داده (DS)** است.
- مفاهیم مطرح در طراحی منطقی پایگاه داده‌ها
  - ساختار داده جدولی : TDS
  - پایگاه داده جدولی : TDB
  - زبان پایگاهی جدولی : TDBL

- **چرا ساختار داده (در معنای عام)؟**
  - برای نمایش نوع موجودیت‌ها و ارتباط بین آنها در سطح { منطقی / فیزیکی }

- **دلایل لزوم ساختار داده (DS) در حیطه پایگاهی:**

۱- تامین کننده محیط فرافایلی (محیط انتزاعی)

۲- مبنا و چارچوب طراحی منطقی DB

۳- مبنا و چارچوب طراحی زبان پایگاه داده‌ها DBL

۴- مبنا و چارچوب طراحی خود DBMS

۵- ضابطه‌ای است برای مقایسه سیستم‌ها و ارزیابی آنها

۶- مبنایی است برای ایجاد و گسترش تکنیک‌های طراحی DB

۷- . . .

- DSها [در حیطه دانش و تکنولوژی DB]:
  - ۱- HDS [از Hierarchical DM] — HDBMS
  - ۲- NDS [از Network DM] — NDBMS
  - ← پیش رابطه‌ای (پکیج IMS و IDMS)
  - ۳- RDS [از Relational DM] — RDBMS پکیج‌های جدولی
  - ۴- ODS [از Object DM] — ODBMS
  - ۵- ORDS [از Object Relational DS] — ORDBMS

- **TDS- ساختار داده جدولی:**
  - عنصر ساختاری اساسی در Relational Model (RM): مفهوم **رابطه**
  - رابطه [Relation]: یک مفهوم ریاضی است ...
  - اما از دید کاربر [در عمل]: نمایش جدولی دارد.
    - فعلا به جای RDS می‌گوییم TDS.

(البته رابطه و جدول تفاوتهای جدی با هم دارند که در مباحث بعدی درس بدان پرداخته می‌شود.)

ساختار داده جدولی (TDS)

- اصطلاحات TDS:
  - ***نوع جدول*** ← { نام جدول / نام و نوع ستون ها } برای نمایش **نوع** { موجودیت و/یا ارتباط }
  - ***سطر*** ← برای نمایش **نمونه** { موجودیت / ارتباط }
  - ***ستون*** ← برای نمایش **صفت**
- **عنصر ساختاری اساسی:**
  - هر DS حداقل یک **عنصر ساختاری اساسی** دارد.

تعریف: عنصری است که به کمک آن نوع موجودیت، نوع ارتباط، و یا هردو آنها را نمایش می‌دهیم.

  - TDS فقط یک عنصر ساختاری اساسی دارد : همان **نوع جدول**
  - **نکته:** صفت **شناسه** در نوع موجودیت‌ها، حکم **کلید** را در جدول دارد.

پایگاه داده جدولی (TDB)

TDB چیست؟

- از دید کاربر
  - طراح
  - پیاده ساز DB
  - برنامه ساز AP

از لحاظ نوع: مجموعه‌ای است از تعدادی **نوع جدول** (که آنها را طراحی می‌کنیم)

از لحاظ محتوای داده‌ای [در سطح نمونه]: مجموعه‌ای است از نمونه‌های متمایز یک [چند] **نوع سطر**

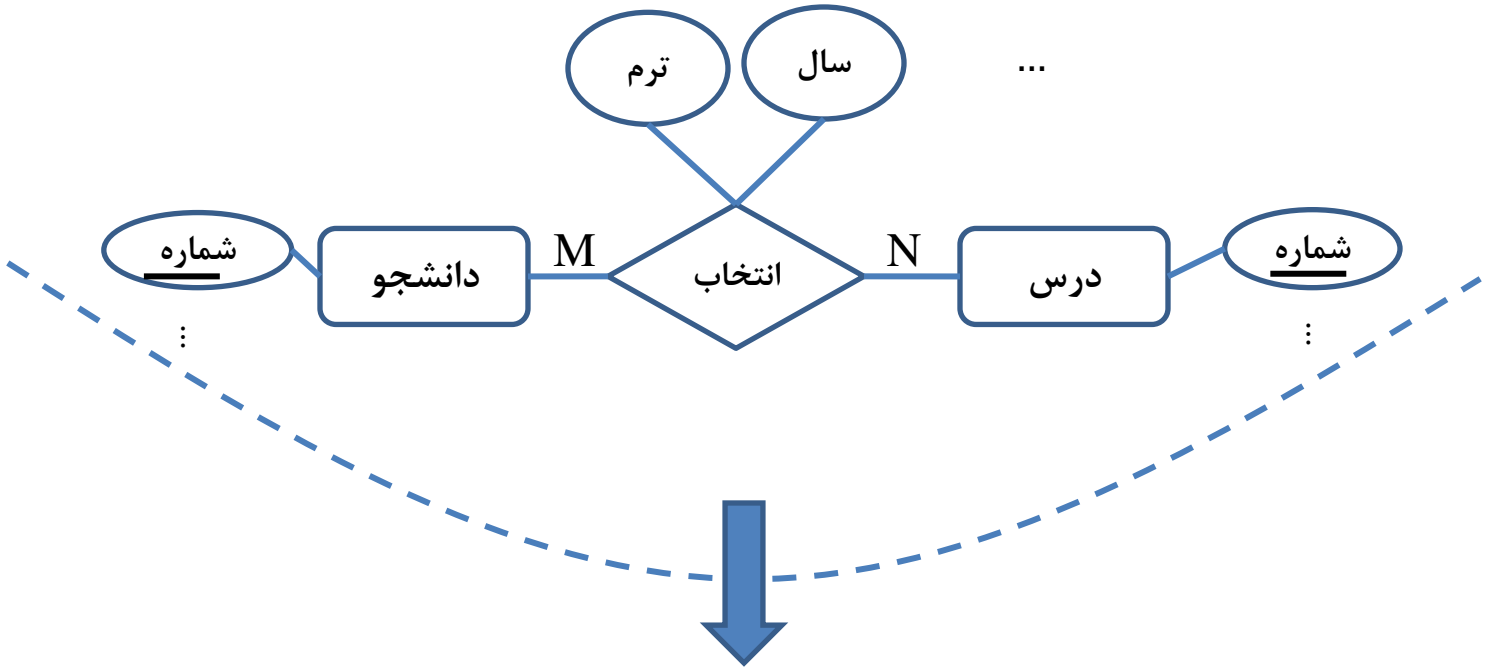
- نوع سطر را همان نوع جدول مشخص می‌کند.

مفهوم کلید

- مفهوم کلید در مدل داده جدولی تعریف نشده است و برگرفته از مفاهیم تعریف شده در مدل داده‌ای رابطه است.
- [*از دیدگاه کاربردی*] **کلید** امکان دسترسی به تک نمونه (از یک نوع موجودیت یا نوع ارتباط) را فراهم می‌نماید. لذا مقدار آن در سطرهای جدولِ مبین موجودیت یا ارتباط، **یکتا** است.
- [*از دیدگاه کاربردی*] یک یا چند صفت (ستون) تشکیل **کلید اصلی** را در یک جدول می‌دهند اگر مقادیر آن(ها) در سطرهای جدول یکتا و معلوم باشد.
- در مواقعی ممکن است بیش از یک کلید داشته باشیم. یکی از کلیدها که مقادیرش در همه سطرها معلوم است را کلید اصلی می‌گیریم (بقیه را با یکتا بودن مقادیر – با استفاده از UNIQUE در SQL- مشخص می‌نماییم).
- در معرفی مدل داده رابطه‌ای، با انواع کلید و تعاریف آنها آشنا می‌شوید.

طراحی منطقی با TDS – رابطه چند به چند

مثال چندی M:N



مساله: تبدیل به TDB [با TDS]

- سه نوع جدول لازم داریم:
  - برای هر نوع موجودیت یک نوع جدول
  - برای نوع ارتباط M:N یک نوع جدول

طراحی منطقی با TDS – رابطه چند به چند (ادامه)

خط زیرین
نمایانگر کلید

STT

STID	STNAME	STLEV	STMJR	STDEID
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	ms	phys	d11
⋮	⋮	⋮	⋮	⋮

COT

COID	COTITLE	CREDIT	COTYPE	CEDEID
⋮	⋮	⋮	⋮	⋮
co3	programming	4	t (تئوری)	d13
⋮	⋮	⋮	⋮	⋮

طراحی منطقی با TDS – رابطه چند به چند (ادامه)

طبق قواعد معنایی محیط ممکن است سال و ترم هم جزو کلید باشند.

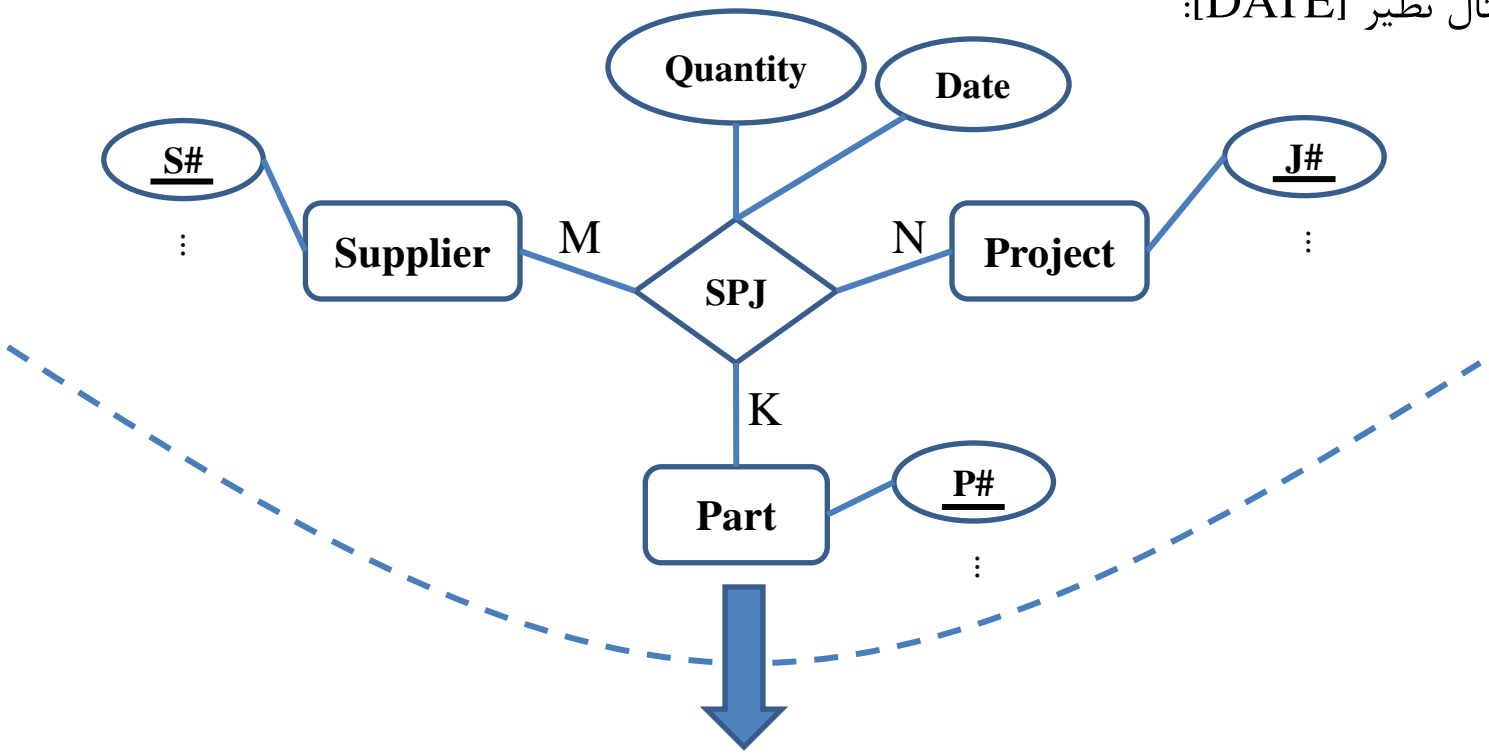
(در واقع اگر صفت چند مقداری مرکب برای رابطه باشند، جزو کلید محسوب می‌شوند.)

STCOT

STID	COID	TR	YR
⋮	⋮	⋮	⋮
888	co2	1	87
888	co3	1	87
444	co2	1	87

طراحی منطقی با TDS – رابطه چند به چند (ادامه)

مثال نظیر [DATE]:



مساله: تبدیل به TDB [با TDS]

- چهار نوع جدول داریم:
  - برای هر نوع موجودیت یک نوع جدول
  - برای نوع ارتباط یک نوع جدول

طراحی منطقی با TDS – رابطه چند به چند (ادامه)

Supplier

S#	SNAME	CITY	…
s1	…	c1	…
s2	…	c1	…
⋮	⋮	⋮	⋮

Part

P#	PNAME	CITY	…
p1	…	c1	…
p2	…	c2	…
⋮	⋮	⋮	⋮

Project

J#	JNAME	CITY	…
j1	…	c2	…
j2	…	c1	…
⋮	⋮	⋮	⋮

طبق قواعد معنایی محیط ممکن است تاریخ هم جزو کلید بشود.

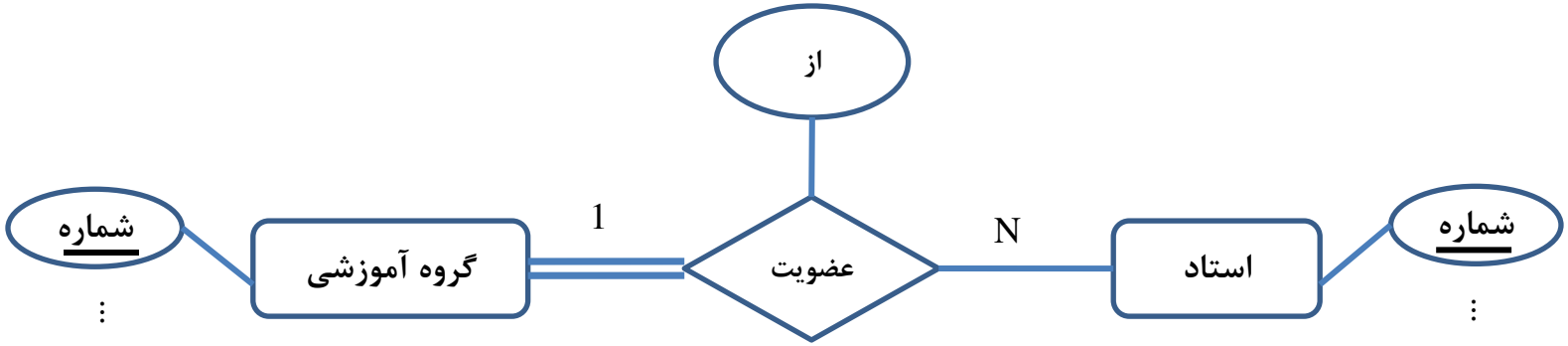
(در واقع اگر صفت چند مقداری باشد، جزو کلید محسوب می‌شود.)

SPJ

S#	P#	J#	Date	QTY
s1	p1	j1	d1	100
s1	p1	j1	d2	50
⋮	⋮	⋮	⋮	⋮

طراحی منطقی با TDS – رابطه یک به چند

مثال چندی 1:N



- دو نوع جدول داریم:
  - یکی برای نوع موجودیت سمت 1
  - یکی برای نوع موجودیت سمت N و نیز خود ارتباط

طراحی منطقی با TDS – رابطه یک به چند (ادامه)

DEPT

DEID	DETITLE	…	DEPHONE
D11	Phys	…	…
D12	Math	…	…
⋮	⋮	⋮	⋮

PROF

PRID	PRNAME	RANK	…	FROM	DEID
Pr100	…	استاد	…	d1	D13
Pr200	…	استادیار	…	d2	D11
Pr300	…	دانشیار	…	?	?

* ستون DEID در جدول PROF **کلید خارجی** است و با خط‌چین مشخص می‌شود.

کلید خارجی[کاربردی]: ستون c از جدول T1 در جدول T2 کلید خارجی است هرگاه در جدول T1 کلید اصلی باشد.

در چه حالاتی استفاده از سه نوع جدول قابل توجیه است؟

طراحی منطقی با TDS – رابطه یک به یک

مثال چندی 1:1

- دو نوع جدول داریم:
  - یکی برای نوع موجودیت سمت 1 غیرالزامی
  - یکی برای نوع موجودیت سمت 1 الزامی و نیز خود ارتباط

طراحی منطقی با TDS – رابطه یک به یک (ادامه)

- یک طرز طراحی ممکن:

DEPT

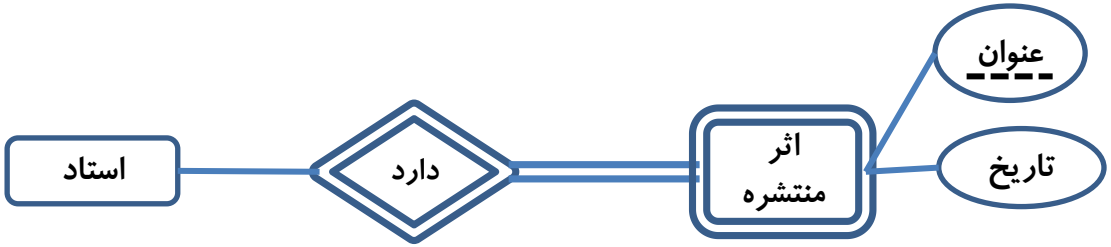
DEID	DETITLE	…	DEPHONE	PRID
D11	Phys	…	…	…
D12	Math	…	…	…
⋮	⋮	⋮	⋮	⋮

PROF

PRID	PRNAME	RANK	…
Pr100	…	استاد	…
Pr200	…	استادیار	…
Pr300	…	دانشیار	…
⋮	⋮	⋮	⋮

کنجکاوی؟ طرزهای دیگر طراحی؟

مثال رابطه شناسا (رابطه موجودیت ضعیف)



- دو نوع جدول داریم:
  - یکی برای نوع موجودیت قوی
  - یکی برای نوع موجودیت ضعیف و رابطه (حاوی شناسه موجودیت قوی)

طراحی منطقی با TDS – رابطه شناسا (ادامه)

PROF

PRID	PRNAME	RANK	…
Pr100	…	استاد	…
Pr200	…	استادیار	…
Pr300	…	دانشیار	…
⋮	⋮	⋮	⋮

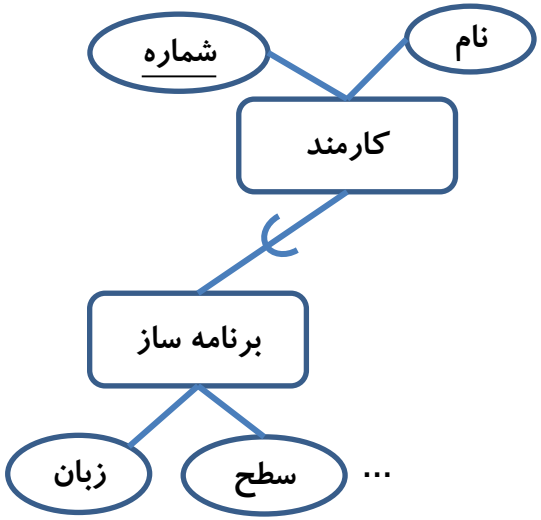
PUB

PRID	PTITLE	…	PDATE
Pr100	Data Encryption…	…	…
Pr100	Semantic Analysis of …	…	…
⋮	⋮	⋮	⋮

❊ دو صفت PRID (کلید خارجی از جدول PROF) و TITLE، کلید اصلی جدول انتشارات را تشکیل می‌دهند.

کنجکاوی؟ حذف و بروزرسانی در جدول PROF چه تاثیری بر PUB باید داشته باشد.

مثال رابطه IS-A



- دو نوع جدول داریم:
  - یکی برای زبرنوع موجودیت (حاوی صفات عام یا مشترک)
  - یکی برای نوع زیرنوع موجودیت (حاوی صفات خاص زیرنوع و شناسه زبرنوع)

طراحی منطقی با TDS – رابطه IS-A (ادامه)

EMP

EID	ENAME	EBDATE	…	EPHONE
E100	…	…	…	…
E101	…	…	…	…
E102	…	…	…	…
⋮	⋮	⋮	⋮	⋮

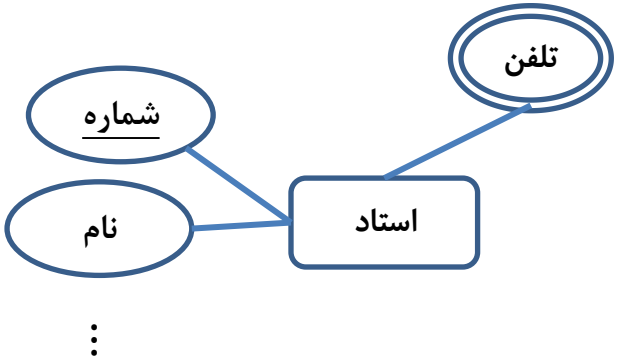
PROG

EID	LANG	…	LEVEL
E100	C++	…	…
E102	Java	…	…
⋮	⋮	⋮	⋮

❊ EID (کلید خارجی از جدول EMP) کلید اصلی جدول PROG نیز هست.

کنکاوی؟ حذف و بروزرسانی در جدول EMP چه تاثیری بر PROG باید داشته باشد (و بالعکس)؟

مثال صفت چندمقداری



- دو نوع جدول داریم:
  - یکی برای نوع موجودیت (حاوی صفات تک‌مقداری)
  - یکی برای صفت (ساده یا مرکب) چندمقداری

طراحی منطقی با TDS – صفت چندمقداری (ادامه)

PROF

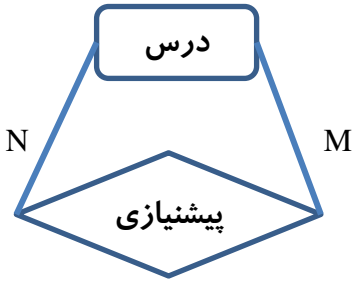
PRID	PNAME	RANK	…	
Pr100	…	…	…	…
Pr101	…	…	…	…
Pr102	…	…	…	…
⋮	⋮	⋮	⋮	⋮

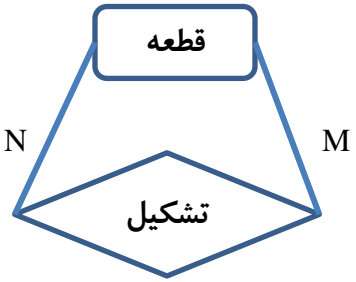
PROFTEL

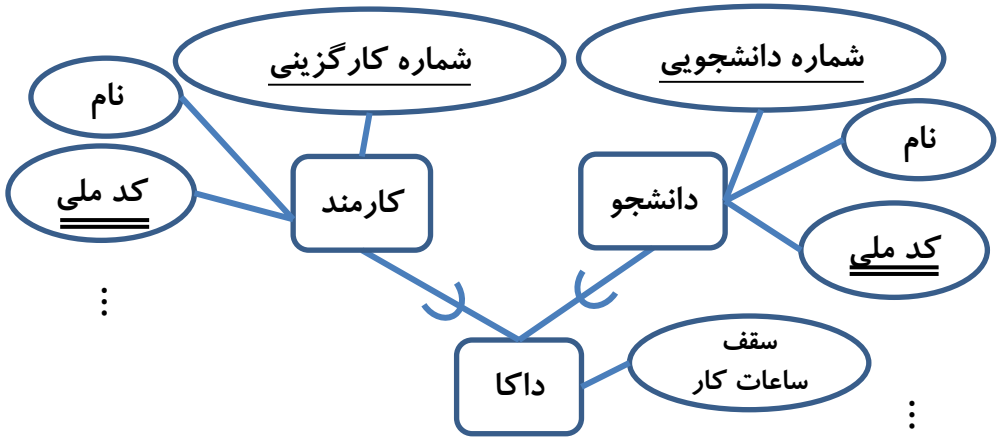
PRID	TEL
Pr100	09121234567
Pr100	02177889911
Pr101	09352348762
⋮	⋮

طراحی منطقی با TDS (ادامه)

- **تمرین:** TDB را برای مدلسازی‌های زیر طراحی کنید.

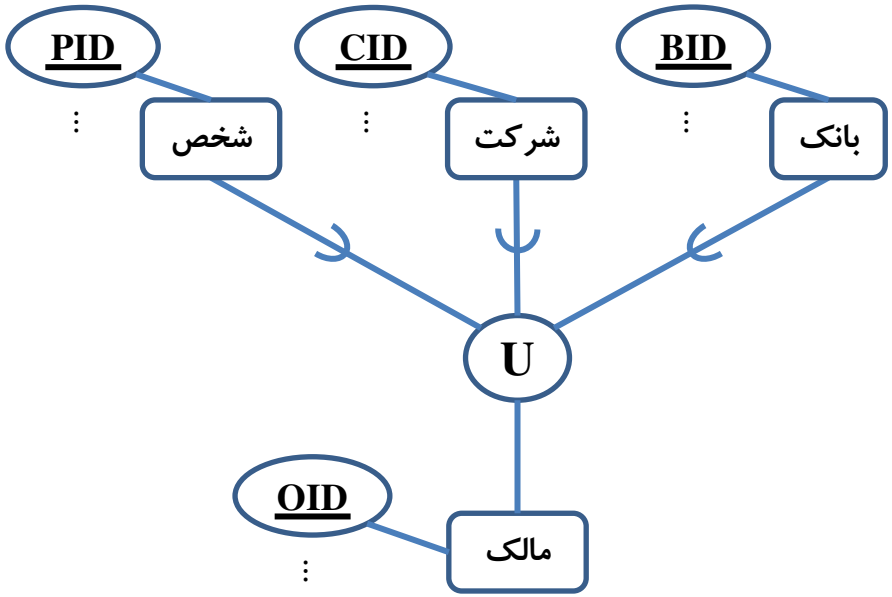


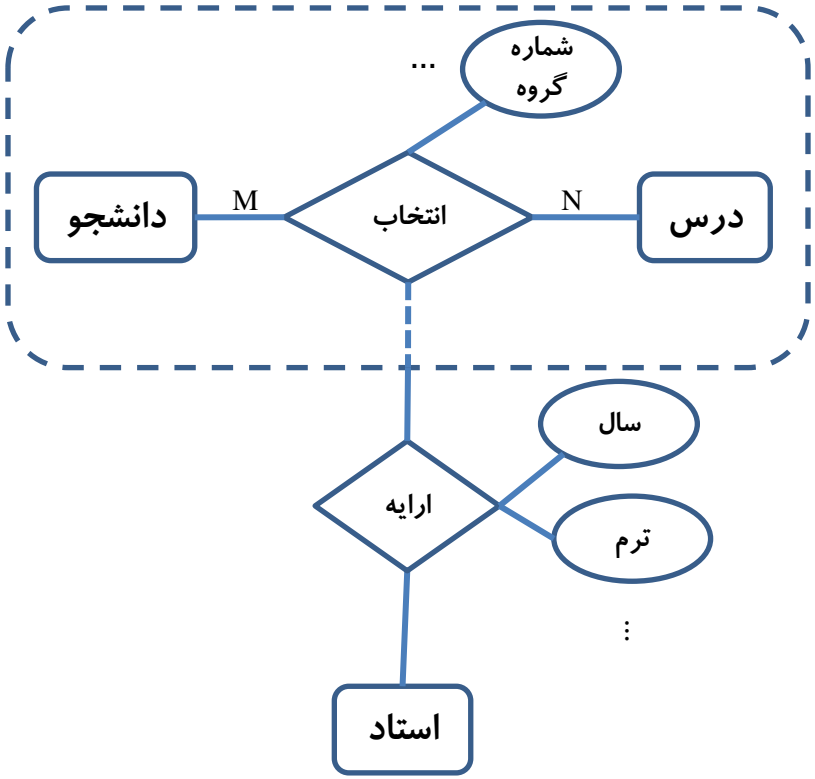




طراحی منطقی با TDS (ادامه)

- **تمرین:** TDB را برای مدلسازی‌های زیر طراحی کنید.





پرسش و پاسخ . . .

amini@sharif.edu

به نام آنکه جان را فکرت آموخت

بخش چهارم:
مقدمات پیاده‌سازی و SQL

مرتضی امینی

نیمسال دوم ۹۴-۹۵

(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)

- برای پیاده‌سازی طراحی منطقی انجام شده در یک سیستم مدیریت پایگاه داده‌ها نیاز به یک زبان پایگاهی داریم.
- زبان SQL زبان استاندارد انجام عملیات پایگاهی در پایگاه داده‌های رابطه‌ای (از دیدگاه کاربردی: جدولی) است.
- دستورهای Structured Query Language (SQL):
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
  - Data Control Language (DCL)
- چند دستور از DDL:
  - ایجاد جدول CREATE TABLE
  - حذف جدول DROP TABLE
  - تغییر جدول ALTER TABLE
- **نکته:** در دستورات SQL در دو طرف مقادیر متنی یا رشته‌ای از single quote استفاده می‌شود (بسیاری از سیستم‌های پایگاه داده double quote را هم می‌پذیرند) ولی در اطراف مقادیر عددی چیزی قرار نمی‌گیرد.

تعریف و حذف پایگاه داده و شِما

- **دستور تعریف پایگاه داده**

CREATE DATABASE *DatabaseName*

- **دستور حذف پایگاه داده**

DROP DATABASE *DatabaseName*

- در اغلب سمپادها می‌توان در یک پایگاه داده چند شما تعریف کرد.
- **دستور تعریف و حذف شِما**

CREATE SCHEMA *SchemaName*

DROP SCHEMA *SchemaName*

- **شمای پایگاه داده‌ها** عبارت است از تعریف (توصیف) ساختهای منطقی طراحی شده و نوعی برنامه است شامل تعدادی دستور برای تعریف و کنترل داده‌ها.
- در واقع شِما شامل همه جداول، نوعها، دامنه‌ها، دیدها و محدودیتهای مرتبط با یک برنامه کاربردی است.

تعریف جدول

تعریف جدول‌ها: شِمای پایگاه جدولی

- دستور تعریف جدول CREATE TABLE

CREATE TABLE *TableName*
{ **(** *columnName* *dataType* [**NOT NULL** | **UNIQUE**]
[**DEFAUTL** *defaultOption*][***CHECK*** (*searchCondition*)] **[, …])**}
[**PRIMARY KEY** (*listOfColumns*),]
{[**UNIQUE** (*listOfColumns*),][, …]}
{[**FOREIGN KEY** (*listOfForeignKeyColumns*)
REFERENCES *ParentTableName* [(*listOfCandidateKeyColumns*)],
[**ON UPDATE** *referentialAction*]
[**ON DELETE** *referentialAction*]][, …]}
{[**CHECK** (*searchCondition*)][, …]}

- می‌توان جدول را به صورت موقت نیز (با استفاده از CREATE TEMPORARY TABLE) ایجاد کرد. جدول موقت حاوی داده‌های ناپایا است و پس از اینکه برنامه کاربر (SQL Session) اجرایش تمام بشود، این جدول توسط سیستم حذف می‌شود.

انواع داده‌ای

- انواع داده‌های قابل استفاده در تعریف ستون‌ها عبارتند از:
  - کاراکتری: CHAR(n), VARCHAR(n)
  - بیتی: BIT [VARYING] (n)
  - عددی: NUMERIC(p, q), REAL, INTEGER, SMALLINT, FLOAT(p), DOUBLE PRECISION
  - زمانی: DATE, TIME, TIMESTAMP, INTERVAL
  - ....
- در برخی DBMSها، نوع داده‌های خاصی پشتیبانی می‌شود که امکان ذخیره، بازیابی و پردازش داده‌های از آن نوع را برای کاربر تسهیل می‌نماید. به طور مثال نوع داده جغرافیایی در PostgreSQL.

محدودیتهای صفتی (ستونی)

- **Default**: تعیین مقدار پیش‌فرض یک ستون
- **Not Null**: ستون ناهیچ‌مقدار
- **Unique**: یکتایی مقادیر ستون(ها)
- **Primary Key**: کلید اصلی (می‌توان تعدادی از ستونها را با یکدیگر به عنوان کلید اصلی تعریف کرد)
- **Foreign Key .... References ....**: کلید خارجی (می‌توان تعدادی از ستونها را با یکدیگر به عنوان کلید خارجی تعریف کرد)
- **Check**: تعیین محدودیت مقداری برای مقادیر ستون

مثالی از تعریف جدول

مثال

شِمای پایگاه داده جدولی:

```
CREATE TABLE STT
( STID        CHAR(8) NOT NULL ,
STNAME        CHAR(25) ,
STLEV         CHAR(12) ,
STMJR         CHAR(20) ,
STDEID        CHAR(4)
)
PRIMARY KEY (STID) ;
CHECK STMJR IN { 'bs' , 'ms' , 'doc' , '???' }
```

```
CREATE TABLE COT
( COID        CHAR(6) NOT NULL ,
COTITLE       CHAR(16) ,
CREDIT        SMALLINT ,
COTYPE        CHAR(1) ,
CODEID        CHAR(4) ,
)
PRIMARY KEY (COID) ;
```

محدودیت صفتی (ستونی) [کلاز کنترلی]

مثالی از تعریف جدول (ادامه)


```
CREATE TABLE SCT
( STID        CHAR(8)  NOT  NULL ,
COID          CHAR(6)  NOT  NULL ,
TR            CHAR(1) ,
YR            CHAR(5) ,
GRADE         DECIMAL(2 , 2)
)
PRIMARY  KEY ( STID, COID )
CHECK  0 ≤ GRADE ≤ 20
FOREIGN  KEY  (STID)  REFERENCES  STT  (STID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
FOREIGN  KEY  (COID)  REFERENCES  COT (COID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
```

محدودیت صفتی (ستونی) [کلاز کنترلی]

حذف جدول

دستور حذف جدول DROP TABLE

DROP TABLE ***tablename*** **[CASCADE| RESTRICT]**

- **CASCADE** باعث می‌شود که همه اشیاء وابسته به جدول (مانند دیدهای تعریف شده بر روی آن یا محدودیتهایی مانند کلید خارجی وابسته به آن) نیز به صورت خودکار حذف شود.
- **RESTRICT** در صورت وجود دیگر اشیاء وابسته به جدول، از حذف آن جلوگیری می‌کند. پیش‌فرض این دستور، RESTRICT است.

DROP TABLE SCT

دستور تغییر جدول ALTER TABLE

اضافه کردن ستون، تغییر تعریف ستون، حذف ستون و ...

ALTER TABLE ***tableName***

[ADD [COLUMN] ***columnName*** ***dataType*** **[NOT NULL] [UNIQUE]**

[DEFAULT ***defaultOption*****] [CHECK (*****searchCondition*****)]]**

[DROP [COLUMN] ***columnName*** **[RESTRICT | CASCADE]]**

[ADD [CONSTRAINT [***constraintName*****]]** ***tableConstraintDefinition*****]**

[DROP [CONSTRAINT ***constraintName*** **[RESTRICT | CASCADE]]**

[ALTER [COLUMN] SET DEFAULT ***defaultOption*****]**

[ALTER [COLUMN] DROP DEFAULT]

. . .

مثال

اضافه کردن ستون «وضعیت» به جدول اطلاعات دانشجو

ALTER TABLE STT

ADD COLUMN STATE CHAR(10)

- در شِمای پایگاهی ← دستورات
  - Data Definition (DD)
  - Data Controller (DC)

و نه دستورات
Data Manipulation (DM)

کجای؟ این جدایی چه مزایایی دارد؟

کجای؟ سیستم با شِمای پایگاهی چه می‌کند؟

- اطلاعات موجود در آن را در جایی به نحوی ذخیره می‌کند. ← در تعدادی جدول

کاتالوگ سیستم
دیکشنری سیستم
مِتا داده‌ها
← حاوی فراداده‌ها و داده‌های کنترلی در مورد داده‌های کاربران

مثال

مثالی از جدول‌های کاتالوگ:

SysTables

نام جدول	ایجاد کننده	تاریخ	تعداد ستون	...
STT	C1	D1	5	
COT	C1	D2	5	
SCT	C2	D2	5	
⋮	⋮	⋮	⋮	

جدولی که جدول‌ها را مدیریت می‌کند.

SysCols

نام ستون	نام جدول	نوع	طول	...
STID	STT	CHAR	8	
STNAME	STT	CHAR	25	
⋮	⋮	⋮	⋮	
GR	SCT	DEC	2,2	

جدولی که ستون‌ها را مدیریت می‌کند.

آشنایی با کاتالوگ (ادامه)

کنجکاوی؟ آیا برنامه ساز می تواند محتوای کاتالوگ را مستقیما تغییر دهد؟ (با دستورات INSERT, DELETE,UPDATE)

- تمرین: حداقل سه جدول دیگر برای کاتالوگ طراحی کنید.
- تمرین: چه اطلاعاتی در کاتالوگ ذخیره می شود؟

- عملیات در TDB : **دستور های DML**

بازیابی	**SELECT**
درج	**INSERT**
حذف	**DELETE**
بهنگام سازی	**UPDATE**

دستور بازیابی SELECT

SELECT [ALL | DISTINCT] *item(s) list*
FROM *table(s) expression*
[WHERE *condition(s)*]
[ORDER BY *Col(s)*]
[GROUP BY *Col(s)*]
[HAVING *condition(s)*]

- خروجی دستور SELECT یک جدول است.
- از DISTINCT برای حذف سطرهای تکراری در جدول نتیجه استفاده می‌شود.
- در شرط WHERE می‌توان از =، <>، <، >، >=، <=، BETWEEN، LIKE و IN استفاده کرد.

مثال

```
SELECT  STT.STID     AS    SN ,
        STT.STNAME   AS    SName
        FROM     STT
        WHERE    STT.STMJR='phys'
                 AND
                 STT.STLEV='bs'
```

مثال

```
SELECT   STT1. STID          AS    SN ,
         STT1. STNAME    AS    SName
        FROM     STT    AS    STT1
        WHERE    STT1. STMJR='phys'
                 AND
                 STT1. STLEV='bs'
```

مثال

یک کپی از جدول با نام جدید، نام‌گذاری جدول جواب:

```
(SELECT   S.*
          FROM   S )   AS   MyS
ORDER   BY   SNAME   یا   2
```

شماره ستون

- مرتب شده:
- پیش فرض صعودی: (Ascending)
- نزولی (Descending): باید قید شود.

مثال

قابلیت‌های پیشرفته (Advanced features):

```
SELECT   S#, CITY
         FROM    S
         WHERE   SNAME { LIKE / NOT LIKE }
```

- *'%N'* → با N تمام شود
- *'M%'* → با M شروع شود
- *'_ _ A _ _'* → دقیقا ۵ کاراکتر، کاراکتر سوم A

BETWEEN

مثال

```
SELECT P#
        FROM P
        WHERE   WEIGHT BETWEEN (5,15)
```

یا

```
        WHERE   WEIGHT >=5   AND WEIGHT <=15
```

- شماره قطعاتی را بدهید که وزن آنها بین ۵ و ۱۵ است.

مثال NULL

```
SELECT  S#, CITY
        FROM    S
        WHERE   STATUS { IS NULL
                       { IS NOT NULL
```

کجای؟ بررسی برخورد یک package با NULL؟

عملگرهای جبر مجموعه‌ها

tablename1 **op** ***tablename2*** **[CORRESPONDING [BY {*column*, [, *column* …]}]]**

$$op \in \left\{ \begin{array}{c} \textbf{UNION [ALL]} \\ \textbf{INTERSECT [ALL]} \\ \textbf{EXCEPT [ALL]} \end{array} \right\}$$

- اگر از گزینه CORRESPONDING BY استفاده شود، عمل درخواست شده روی ستون‌های تصریح شده انجام می‌شود.
- اگر CORRESPONDING بدون BY استفاده شود، عمل درخواست شده روی ستون‌های مشترک انجام می‌شود.
- اگر از این گزینه استفاده نشود، عمل روی تمام ستون‌های دو جدول انجام می‌شود.
  - **شرط استفاده:** برابری Heading: هم‌نامی و هم نوعی ستون (های) دو جدول
- **توجه:** تکراری‌ها در نتیجه اجرای عملگرهای جبر مجموعه‌ها حذف می‌شوند مگر آنکه از ALL استفاده شود.

عملگرهای جبر مجموعه‌ها (ادامه)

مثال

شماره تهیه کنندگانی را بدهید که حداقل یک قطعه تولید می‌کنند.

```
SELECT  S.S#,
        FROM    S
        INTERSECT
SELECT  SP.S#,
        FROM    SP
```

مثال

تست سازگاری پایگاه داده‌ها: هر فردی که قطعه ای تولید کرده باید یکی از افراد ثبت شده در جدول تولیدکنندگان باشد.

```
SELECT  SP.S#,
        FROM    SP
        EXCEPT
SELECT  S.S#,
        FROM    S
```

مدل دیگر

SP *EXCEPT* S **Using** S# یا **Corresponding by** S#

عملگرهای جبر مجموعه‌ها (ادامه)

مثال: شماره تهیه کنندگانی را بدهید که هیچ قطعه‌ای تولید نمی‌کنند.

```
SELECT  S.S#,
        FROM    S
        EXCEPT
SELECT  SP.S#,
        FROM    SP
```

- تمرین: این مثال‌ها به طرز دیگر هم نوشته شود.

توابع جمعی (گروهی)

- **Aggregation Functions**
  - AVG ← میانگین
  - MIN ← مینیمم
  - MAX ← ماکزیمم
  - SUM ← جمع
  - COUNT / COUNT(*) ← تعداد عبارات ناهیچمقدار / تعداد کل سطرها

مثال: بیشینه وضعیت تهیه کنندگان در شهرهای c1 یا c2

```
SELECT  MAX ( STATUS )   AS   SMAX
        FROM     S
        WHERE    CITY='c1'
                 OR
                 CITY='c2'
```

توابع جمعی (ادامه)

مثال: تعداد انواع قطعات تولیدی توسط تولیدکنندگان

```
SELECT  COUNT (DISTINCT P#)   AS   N1
        FROM   SP
```

مثال: تعداد انواع قطعات قابل تولید

```
SELECT  COUNT (*)   AS   N2
        FROM   P
```

مثال: تعداد کل قطعات تولیدی توسط s2

```
SELECT  SUM (QTY)   AS   N3
        FROM   SP
        WHERE   S# = 's2'
```

کجای؟ NULL و توابع جمعی؟ (در سه پکیج بررسی شود)

GROUP BY

- سطرهای جدول داده شده در کلاز FROM را گروه بندی می‌کند، به نحوی که مقدار ستون(های) گروه‌بندی در گروه یکسان است.

مثال

تعداد کل قطعات تولیدی توسط هر تولیدکننده

```
SELECT  S#  AS  SN , SUM (QTY)  AS  SQ
        FROM   SP
        GROUP BY   S#
```

SP
گروه‌بندی شده

S#	P#	QTY
s1	p1	…
s1	p2	…
s1	p4	…
s2	p2	…
s2	p3	…
s3	p5	…
…	…	…

جدول جواب

SN	SQ
s1	280
s2	100
s3	203
…	…

نکته در کلاز SELECT نمی توان نام ستونی را آورد که در کلاز GROUP BY نباشد، غیر از ستون‌هایی که با توابع جمعی به دست آمده‌اند.

- **HAVING**
  - امکانی است برای دادن شرط یا شرایط ناظر به گروه سطرها

مثال شماره تهیه‌کنندگانی را بدهید که بیش از ۱۰۰ قطعه تولید کرده‌اند.

```
SELECT  S#
        FROM SP
        GROUP BY S#
        HAVING SUM(QTY) > 100
```

گروه‌بندی (ادامه)

- تمرین : شماره دانشجویانی را بدهید که در ترم دوم سال ۸۷-۸۸ بیش از ۲۰ واحد گرفته باشند.
- تمرین : شماره دانشجویانی را بدهید که در ترم دوم سال ۸۷-۸۸ بیش از ۷ درس گرفته باشند.

نکته: GROUP BY و HAVING در SQL افزونه‌اند، اما نوشتن QUERY بدون آنها پیچیده است.

کجای؟: HAVING بدون GROUP BY؟

کجای؟: به چند روش می‌توان یک کپی از جدول ساخت؟

بازیابی از بیش از یک جدول

مثال

نام تهیه کنندگان قطعه ‘p2’ را بدهید:

در جدول S

در جدول SP

روش اول

```
SELECT  SNAME
        FROM    S, SP
        WHERE   SP.S# = S.S#   AND   SP.P# = 'p2'
```

شبیه سازی عملگر پیوند

نکته

ضرب دکارتی در SQL

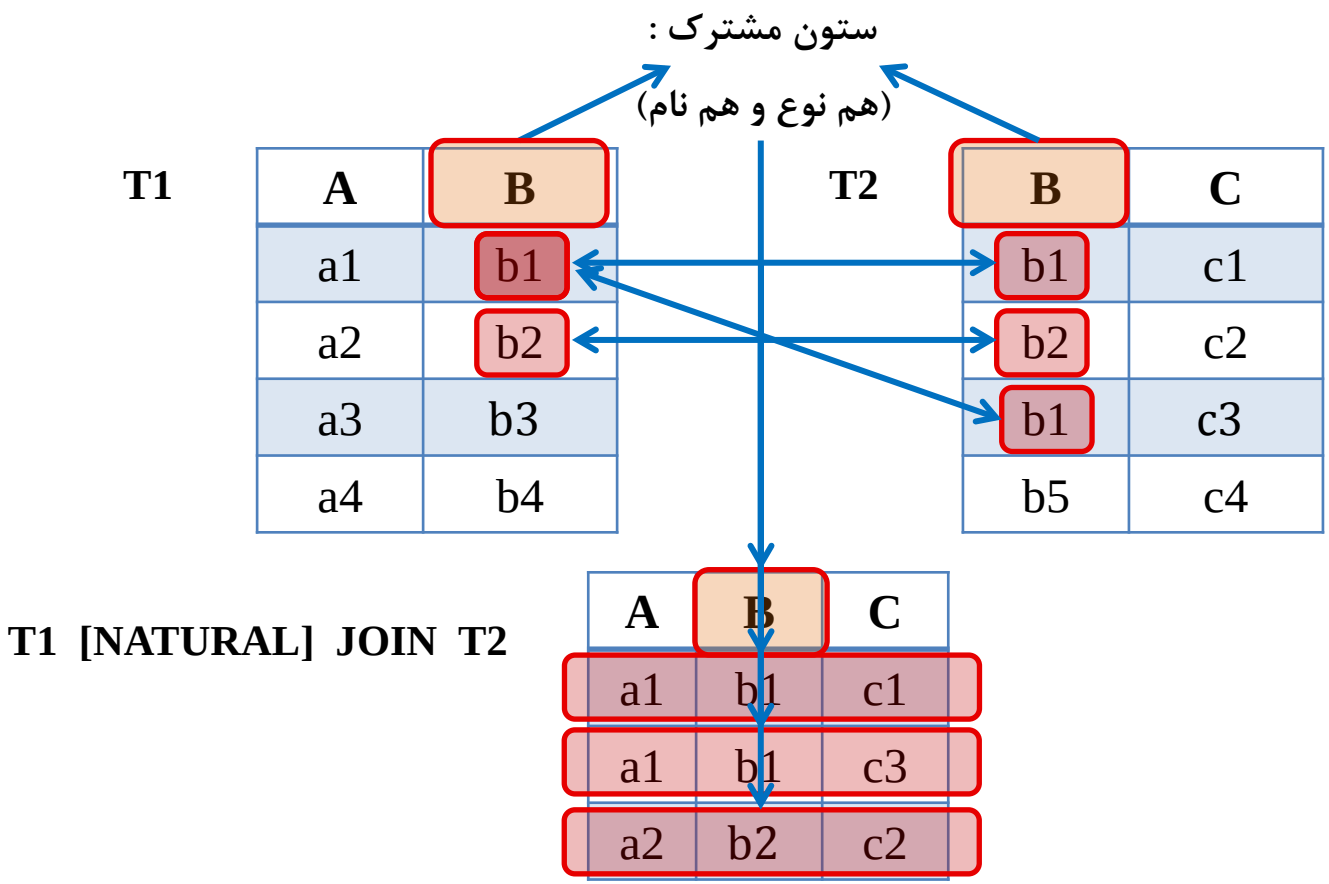
```
SELECT  T1.* , T2.*
        FROM   T1, T2
```

مکانیزم اجرا از دید برنامه‌ساز:

- به ازای هر سطر جدول S، بررسی می‌کند که آیا #S آن در SP وجود دارد یا نه و #P آن سطر در SP، p2 است یا نه. اگر درست بود SNAME آن سطر جزو جواب است.

بازیابی از بیش از یک جدول – عملگر پیوند یا JOIN

- **پیوند:** ارائه مقدماتی (غیر ریاضی)
- T1 [NATURAL] JOIN T2



بازیابی از بیش از یک جدول – عملگر پیوند یا JOIN (ادامه)

- توضیح مقدماتی عملگر **پیوند**:
  - صرف نظر از جزئیات تئوریک، سطرهای دو جدول را که مقدار ستون(های) مشترکشان یکسان است، به هم پیوند می‌زند.

مثال

نام تهیه کنندگان قطعه ‘p2’ را بدهید:

روش دوم

```
SELECT  SNAME
        FROM   S  [NATURAL]  JOIN  SP
        WHERE  P# = ‘p2’
```

S

S#	SNAME	…
s1	sn1	…
s2	sn2	…
s3	sn3	…
s3	sn4	…
…	…	…

SP

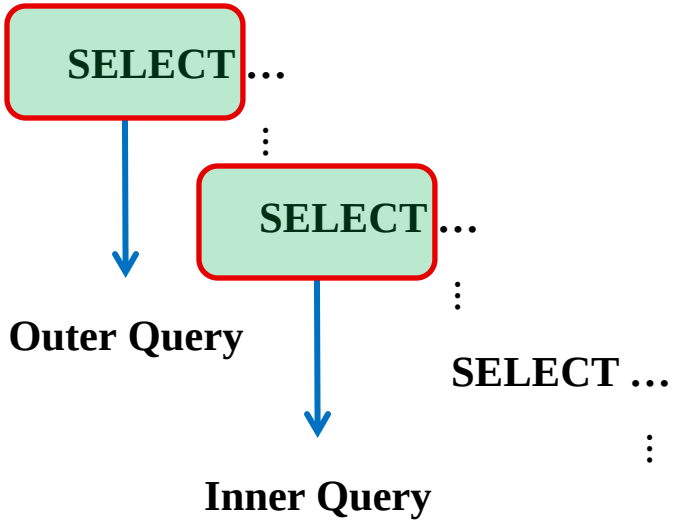
S#	P#	QTY
s1	p1	100
s1	p2	120
s1	p3	500
s2	p1	50
…	…	

S [NATURAL] JOIN SP

S#	SNAME	…	P#	QTY
s1	sn1	…	p1	100
s1	sn1	…	p2	120
s1	sn1	…	p3	500
s2	sn2	…	p1	50
…	…	…		

زیر پرسش یا SubQuary

تعریف: یک SELECT است در درون SELECT دیگر. ← پرسش تو در تو



❑ IN و NOT IN : عملگر تعلق

مثال

روش سوم

```
SELECT  SNAME
        FROM  S
        WHERE  S#  IN  (SELECT  S#  FROM  SP
                        WHERE  P# = 'p2' )
```

عملگر تعلق

روش چهارم

یا

= ANY

روش پنجم

یا

= SOME

❑ مکانیزم اجرا:

- سیستم ابتدا SELECT درونی را اجرا می‌کند، آنگاه به ازای هر سطر S بررسی می‌کند که S# در مجموعه جواب SELECT درونی هست یا نه.

بازیابی از بیش از یک جدول – پرسش های بهم بسته

تعریف: دو پرسش درونی و بیرونی (در یک پرسش تو در تو) را **بهم بسته (Correlated)** گوییم هرگاه در کلاز WHERE پرسش درونی به ستونی از جدول موجود در کلاز FROM پرسش بیرونی، ارجاع داشته باشیم.

- **توجه:** نحوه اجرای پرسش‌های **بهم‌بسته** با طرز اجرای پرسش‌های **نابهم‌بسته** متفاوت است: در حالت بهم‌بسته، سیستم پرسش درونی را به ازای هر سطر از جدول پرسش بیرونی یک بار اجرا می‌کند.

مثال

روش ششم

```
SELECT  SNAME
        FROM  S
        WHERE  'p2'  IN  ( SELECT  P#  FROM  SP
                           WHERE  SP.S# = S.S# )
```

روش هفتم: یا = ANY

روش هشتم: یا = SOME

زیرپرسش بهم‌بسته یا CORRELATED

بازیابی از بیش از یک جدول (ادامه)

- امکان
  - theta ANY
  - theta SOME
  - theta ALL

theta ∈ { = , ≠ , < , ≤ , ≥ , > }

مثال: شماره تهیه کنندگانی را بدهید که مقدار وضعیت آنها بیشینه نباشد.

```
1- SELECT   S#
        FROM   S
        WHERE   STATUS < ANY ( SELECT  DISTINCT   STATUS   FROM   S )
```

چون جواب SELECT تک مقداری است نیازی به ANY نیست.

```
2- SELECT   S#
        FROM   S
        WHERE   STATUS < ( SELECT   MAX (STATUS)    FROM    S)
```

بازیابی از بیش از یک جدول (ادامه)

مثال

روش نهم

```
SELECT  SNAME
        FROM   S
        WHERE  0 < ( SELECT  COUNT(*)
                     FROM   SP
                     WHERE  SP.S#  =  S.S#
                            AND
                            SP.P# = 'p2' )
```

EXISTS و NOT EXISTS

- امکان بررسی وجود یا عدم وجود سطر در جدول بازگشتی

مثال

روش دهم

```
SELECT  SNAME
        FROM   S
        WHERE   EXISTS   ( SELECT *
                           FROM    SP
                           WHERE   SP.S# = S.S#
                                   AND
                                   SP.P# = 'p2' )
```

کنجکاوی؟ روش‌های دیگر ؟

- دستورهای INSERT, UPDATE, DELETE

- درج INSERT:

INSERT INTO *table-name* ***[****(col1,col2, ...)****]***
VALUES (*one row*) | *subquery*

- بهنگام‌سازی UPDATE:

UPDATE *table-name*
SET *col = value / experession [, col = value / experission]…*
⋮
WHERE *condition(s) / subquery*

- حذف DELETE:

DELETE FROM *table-name*
WHERE *condition(s) / subquery*

مثال

درج سطری (سطر کامل – سطر ناقص):

```
INSERT   INTO   STT
         VALUES    ( '222' , 'st2' , 'IT' , 'bs' , 'D17' )
INSERT   INTO   STT
         VALUES   ( '333' , 'st3' , Null , 'ms' , Null )
```

مثال

درج گروهی:

اطلاعات دانشجویان مقطع کارشناسی ارشد

رشته کامپیوتر در جدول موقت T1 درج شود.

```
CREATE   TEMPORRAYR   TABLE   T1
         ( STN,   ....  )

INSERT    INTO    T1
         ( SELECT   STT.*
                  FROM   STT
                        WHERE   STJ = 'comp'
                                AND
                                STL = 'ms'  )
```

مثال

بهنگام‌سازی چند سطر:

- تعداد واحد تمام درس‌های عملی گروه آموزشی D11 را برابر یک کن.

```
UPDATE    COT
    SET    CREDIT = '1'
        WHERE    COTYPE = 'p'   AND    CODEID = 'D11'
```

مثال

بهنگام‌سازی در بیش از یک جدول:

```
UPDATE    STT
    SET    STID = 88104444
        WHERE    STID = 88107777

UPDATE    STCOT
    SET    STID = 88104444
        WHERE    STID = 88107777
```

کنجکاوی؟ اگر دستور دوم اجرا نشود؟

مثال

نمره دانشجویان گروه آموزشی D111 در درس ‘com222’ در ترم دوم سال ۸۵-۸۶ را ناتمام اعلان کن.

```
UPDATE    STCOT
    SET    STCOT.GRADE = ‘U’
        WHERE    STCOT.TR = ‘2’   AND   STCOT.YRYR = ‘85-86’
                AND    STCOT.COID = ‘COM222’
                AND    STID   IN  ( SELECT    STID
                                    FROM    STT
                                        WHERE    STT.STDEID = ‘D111’) ;
```

حذف

مثال

حذف تکدرس: درس com111 را برای دانشجوی 88104444 حذف کنید.

```
DELETE   FROM   STOCOT
         WHERE   STID = 88104444
                 AND
                 COID = 'COM111'
```

کجای؟

آیا این حذف باید انتشار یابد؟

مثال

حذف از بیش از یک جدول:

```
DELETE   FROM   DEPT
         WHERE   DEID = 'D333'

UPDATE   STT
    SET   DEID = 'Null'
         WHERE   DEID = 'D333'
```

دیگر امکانات SQL

- مطالعه شود :
  - پرسش بازگشتی (Recursive)
  - SQL ادغام شده
  - SQL پویا
  - نوشتن رویّه
  - نوشتن تابع
  - امکانات شیئ- رابطه‌ای
  - مدیریت تراکنش

پرسش و پاسخ . . .

amini@sharif.edu

به نام آنکه جان را فکرت آموخت

بخش پنجم:
معماری پایگاه داده‌ها

مرتضی امینی

نیمسال دوم ۹۴-۹۵

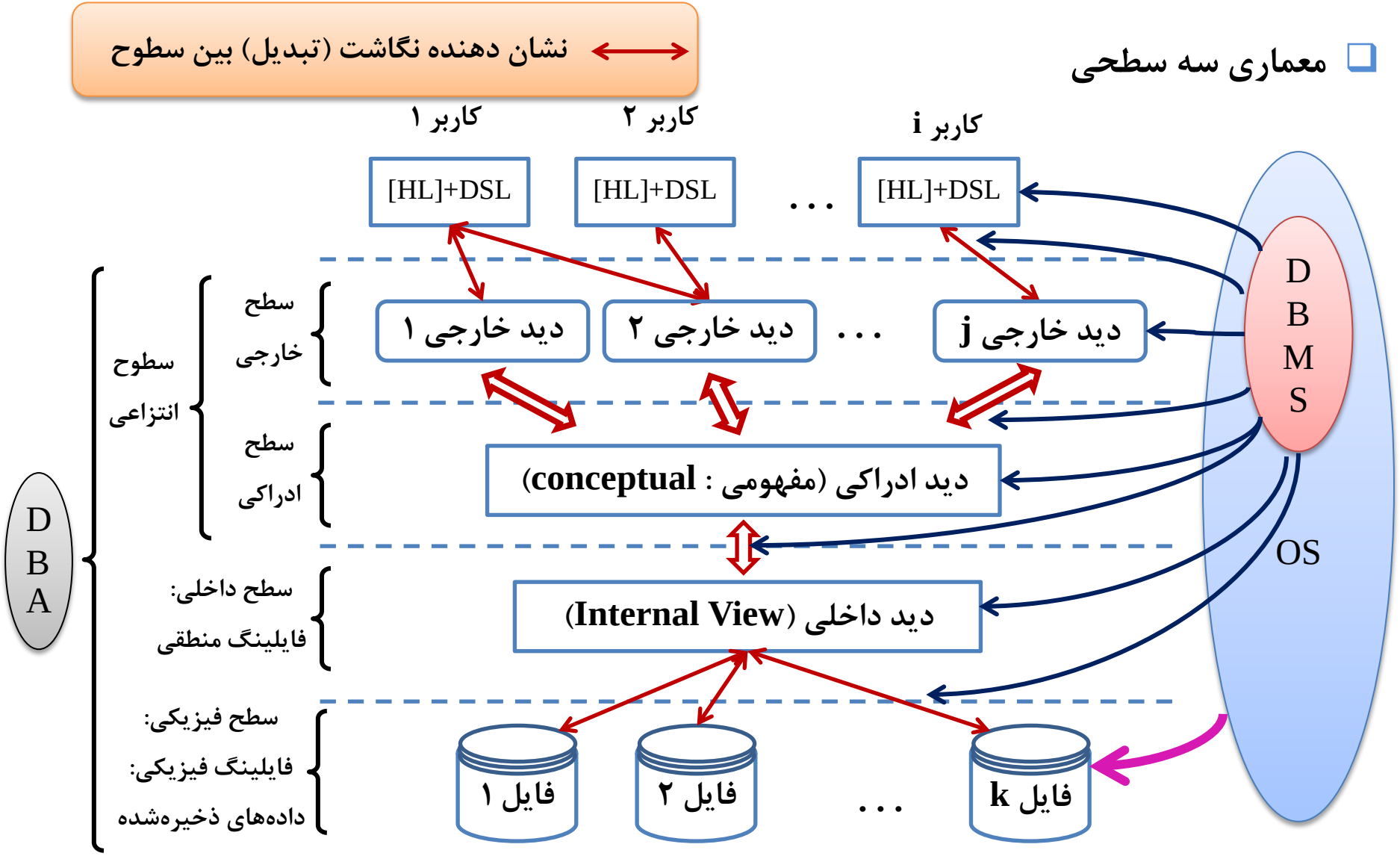
(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)

معماری پایگاه داده‌ها

- نیاز به یک معماری واحد از دیدگاه **داده شناسانه** (و نه دیدگاه عملکردی یا دیدگاه مولفه-مبنا) که در آن داده‌ها به گونه‌ای قابل فهم (مستقل از پیچیدگی‌های سطح سمپاد) به کاربر نمایش داده شود.
- عدم وجود اتفاق نظر در چگونگی معماری پایگاه داده‌ها در سالهای آغازین ایجاد
- **پیشنهاد معماری سه سطحی از سوی ANSI / SPARC**
  - سه سطح معماری ANSI، در واقع سه سطح **تعریف و کنترل داده‌ها** است.
  - دو سطح در محیط انتزاعی و یک سطح در محیط فایلینگ منطقی.

معماری سه سطحی [پیشنهادی ANSI]

- معماری سه سطحی



اجزای معماری سه سطحی پایگاه داده‌ها:

۱- کاربر User

۲- زبان میزبان HL: مانند زبانهای جاوا، C# و دلفی

۳- زبان داده‌ای فرعی (زیرزبان داده‌ای) DSL: زبانهای داده‌ای ادغام شده در زبانهای میزبان

۴- دید خارجی (نمای خارجی) → **سطح خارجی**

۵- دید ادراکی (فرایافتی یا مفهومی) → **سطح ادراکی**

۶- دید داخلی → **سطح داخلی**

۷- فایل‌های فیزیکی

۸- سیستم مدیریت پایگاه داده‌ها (کوتاهتر: سمپاد)

۹- مدیر پایگاه داده‌ها (DBA)

دید ادراکی

- دید (نمای) ادراکی (فرایافتی یا مفهومی)

تعریف: دید طراح نسبت به داده های ذخیره شدنی (و نهایتا ذخیره شده) در پایگاه داده‌ها

- ✓ دیدی جامع: دربرگیرنده نیازهای همه کاربران محیط
- ✓ مطرح در محیط انتزاعی (فرافایلی) ← مبتنی بر یک ساختار داده مشخص
- ✓ طراحی با عنصر (عناصر) ساختاری اساسی
- ✓ پس از طراحی ← توصیف شود ← **شمای ادراکی (Conceptual Schema)**

تعریف: **نوعی برنامه حاوی دستورات** { **DDL**، **DCL** } **و نه دستورات DML**

- ✓ شمای ادراکی به سیستم مدیریت داده می شود و در کاتالوگ آن نگهداری می‌شود.

دید ادراکی (ادامه)

مثال

CREATE TABLE S1 **...**

CREATE TABLE S2 **...**

CREATE TABLE S3 **...**

- **دید ادراکی:** جدول‌های مبنای S1 و S2 و S3
- **شمای ادراکی:** تعریف جدول‌ها است.
- به این جدول‌ها **جدول‌های مبنا** می‌گوییم.

در کجا ؟ ← **کاتالوگ سیستم**

چگونه ؟ ← **در سیستم‌های جدولی: در تعدادی جدول که خود سیستم ایجاد می‌کند.**

- اطلاعات شِمای ادراکی به سیستم مدیریت داده می‌شود و در کاتالوگ آن نگهداری می‌شود.

یادآوری

کاتالوگ سیستم : متا داده‌ها (Data Dictionary : Meta Data)

- حاوی :
  - ✓ تمامی اطلاعات شمای ادراکی
  - ✓ داده های کنترلی
  - ✓ Data About Data

دید ادراکی (ادامه)

مثال

جدول systables:

systables

نام جدول	ایجاد کننده	تاریخ ایجاد	تعداد ستون	کلید اصلی	...
...	...	...	...	...	...

کاربر پیاده‌ساز : **CREATE TABLE** STT **...**

سیستم : **INSERT INTO** SYSTABLES

VALUES (‘STT’ , ‘c1’ , ‘d1’ , 5 , ‘STID’ , … **)**

کاربر پیاده‌ساز : **DROP TABLE** STCOT **...**

سیستم : **DELETE FROM** SYSTABLES

WHERE TNAME = ‘STCOT’

دید ادراکی (ادامه)

- اضافه کردن ستون به یک جدول :

کاربر پیاده‌ساز : **ALTER TABLE** STT
ADD SADDRESS CHAR (80)

سیستم : **UPDATE** SYSTABLES
SET ColN = 6
WHERE TNAME = ‘STT’

سیستم برای جدولی که تعداد ستون‌های آن تغییر می‌کند در سطح فایلینگ چگونه عمل می‌کند؟

آیا با دستور DELETE در سطح جدول‌های مبنا، جدول کاتالوگ تغییر می‌کند؟

DELETE FROM STT
WHERE STID=‘777’

دید (نمای) داخلی

تعریف: دید خود DBMS [و نیز طراح پایگاه داده‌ها، در مرحله طراحی فیزیکی] ، نسبت به داده‌های ذخیره‌شده

- مطرح در سطح فایلینگ منطقی (و گاه مجازی)
- مبتنی بر یک [یا چند] **ساختار فایل** ←
  - **1:1** پیش فرض (یک جدول : یک فایل)
  - **1:N** (چند جدول : یک فایل)
  - **N:1** (یک جدول : چند فایل)
- سطحی که فایل‌های منطقی پایگاه داده‌ها تعریف می‌شود.
- تناظر بین «ساخت» های سطح ادراکی و «ساخت» های سطح داخلی

مثال: تناظر 1:1 بین ساخت‌های سطح ادارکی و سطح داخلی

Table	TableFile
STT	STTFile
COT	COTFile
STCOT	STCOTFile
…	…

دید داخلی (ادامه)

- توصیف دید داخلی ← **شمای داخلی (Internal Schema)** ← دستورهای { تعریف فایل‌ها / کنترل فایل‌ها }

تعریف: **نوعی برنامه که توسط خود DBMS (و گاه براساس اطلاعاتی که طراح – پیاده‌ساز به سیستم می‌دهد) تولید می شود و شرح و وصف فایلینگ منطقی پایگاه داده‌ها است.**

- **توجه:** در شِمای داخلی انواع رکوردها تعریف می‌شوند و دستورهای لازم جهت ایجاد فایل‌ها و کنترل آنها در این شِما وجود دارد.

مثال: شمای داخلی ساده شده در یک زبان شبه پاسکال

```
TYPE    STUDENT = RECORD
        STUDENT-ID        : String ;
        STUDENT-NAME      : String ;
        STUDENT-LEV       : String ;
        STUDENT-MJR       : String ;
        STUDENT-DEPT      : String ;
```

دید داخلی (ادامه)

- اطلاعاتی که طراح-پیاده ساز به سیستم می‌دهد (مانند شاخص) در دید داخلی تاثیر می‌گذارد.

- در سیستم‌های جدولی: خود سیستم روی کلید اصلی (PK) شاخص خودکار (Automatic Index) ایجاد می‌کند. (عمدتا B-Tree)

- برای ایجاد شاخص روی دیگر ستون ها پیاده ساز باید درخواست کند.

مثال

ایجاد شاخص بر روی ستون STNAME که PK نیست:

CREATE INDEX SNX

ON STT (STNAME)

[CLUSTERED] ? **خوشه‌بندی**

دید داخلی (ادامه)

ویژگی‌های ستون شاخص؟

- ✓ تغییر ناپذیر (حتی الامکان)
- ✓ پرکاربرد در کلاز WHERE
- ✓ ... ؟

حذف شاخص: **DROP INDEX** SNX

با اجرای دستور { **DROP TABLE** / **DROP INDEX** } در سیستم چه اتفاقی می‌افتد؟

مثالی از وضعیتی بیان کنید که براساس آن طراح-پیاده ساز تصمیم به ایجاد شاخص می‌گیرد.

دید منطقی DBMS نسبت به داده‌های ذخیره شده

- DBMS
  - **می‌داند**: جنبه های فایلینگ منطقی [مجازی]
    - ✓ چه فایل‌هایی دارد
    - ✓ نگاشت سطح ادراکی به سطح داخلی
    - ✓ صفحات (Pages) فضای پایگاه داده کاربر
    - ✓ فرمت رکورد هر فایل [رکورد داخلی]
    - ✓ ساختار هر فایل
    - ✓ کلید(ها)
    - ✓ استراتژی دست‌یابی به رکوردها
    - ✓ توالی منطقی رکوردها در صفحات
    - ✓ اندازه جاری هر فایل
    - ✓ اندازه گسترش فایل
    - ✓ اطلاعات همگانی
      - BOF
      - EOF
      - R/W
      - ⋮
    - ✓ ارتباط منطقی بین فایل‌ها
    - ✓ ....
  - **نمی‌داند**: جنبه های فایلینگ فیزیکی

دید منطقی DBMS نسبت به داده‌های ذخیره شده (ادامه)

- DBMS
  - **می‌داند**: جنبه های فایلینگ منطقی [مجازی] ....
  - **نمی‌داند**: جنبه های فایلینگ فیزیکی
    - چگونگی نشست فایل‌ها روی دیسک
    - استراتژی دستیابی چگونه پیاده‌سازی شده‌اند.
    - اندازه بلاک (Block) فیزیکی
    - جزئیات تکنیک‌های Blocking
    - Localityرکوردهای فایل‌ها
    - توالی منطقی رکوردها چگونه پیاده‌سازی شده‌اند
    - ...

کجای؟ Locality چیست و بر کدام یک از عملیات روی فایل‌ها تاثیر می‌گذارد؟

فایلینگ مجازی

- در بعضی از سیستم‌های مدیریت جدید، سیستمِ مدیریت، کل فضای پایگاه داده را به صورت **مجموعه‌ای از مجموعه صفحات** می‌بیند، یعنی نوعی **نمای مجازی** از داده‌های ذخیره شده در پایگاه داده دارد.

مثال

در سطح فایلینگ مجازی $DB = \{ Pages \}$

شماره صفحات	تعداد صفحات	نام جدول
p1 … p10	10	STT
p15 … p29	15	COT
P101 … P1000	900	STCOT

SELECT STT.*

FROM STT

WHERE STID = ‘444’

DBMS : **READ** P1

(فرض کنید ‘444’ در P1 است)

دید خارجی

دید (نمای) خارجی

تعریف: دید کاربر (برنامه ساز) خاص است نسبت به داده‌های ذخیره شده [مثلا دید یک AP نویس]

- **دید جزئی (Partial):** دربرگیرنده نیازهای داده‌ای یک کاربر مشخص [برای یک AP مشخص]
- مطرح در سطح انتزاعی ← مبتنی بر یک ساختار داده‌ای مشخص

کنجکاوی؟: آیا این ساختار داده همان ساختار داده سطح دید ادراکی است؟

- روی دید ادراکی طراحی و تعریف می‌شود.
- یک کاربر ← چند دید متفاوت
  چند کاربر ← یک دید مشترک

دید خارجی (ادامه)

- توصیف دید خارجی ← **شِمای خارجی (External Schema)**

تعریف: **نوعی «برنامه»که کاربر سطح خارجی می‌نویسد، حاوی دستوارت «تعریف داده‌ها» و معدود دستورات «کنترل داده‌ها»** (**کجای؟** **چرا معدود؟**)

- شِمای خارجی ← **ذخیره در کاتالوگ**

نکته: در سیستم‌های جدولی، دید خارجی خود نوعی جدول است، اما **مجازی** (Virtual Table) و نه ذخیره‌شده

نکته: دید خارجی در واقع پنجره‌ای است که از آن کاربر خارجی محدوده‌ی داده‌ای خود را می‌بیند و نه بیشتر.

دید خارجی (ادامه)

مثال

کاربر ۱

V1

STID	STNAME
777	st7
444	st4
⋮	⋮

چند ستون از یک جدول

V2

CONUM	COTITLE

دگرنامی ستون

STT

STID	STNAME	STLEV	STMJR	STDEID
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

COT

COID	…

STCOT

STID	COID …

تناظر یک به یک

دید خارجی (ادامه)

مثال

کاربر ۲

V1

STID	STNAME
777	st7
444	st4
⋮	⋮

دید مشترک با کاربر ۱

STT

STID	STNAME	STLEV	STMJR	STDEID
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

COT

COID	…

STCOT

STID	COID …

دید خارجی (ادامه)

کاربر ۳

V3

STNUM	STNAME	COTITLE	TR	YR

دید روی بیش از یک جدول

STT

STID	STNAME	STLEV	STMJR	STDEID
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

COT

COID	…

STCOT

STID	COID …

دید خارجی (ادامه)

مثال

کاربر ۴

V4

STID	STNAM	TR	YR	AVG

صفت مجازی

STT

STID	STNAME	STLEV	STMJR	STDEID
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

COT

COID	…

STCOT

STID	COID …

تعریف شمای خارجی

- از این مثال‌ها نتیجه می‌گیریم که تعریف، طراحی و توصیف دید خارجی در سیستم‌های جدولی از پویایی بالایی برخوردار است.
  - یعنی انواع جدول‌های مجازی را می‌توان روی لایه‌های زیرین تعریف کرد.
- **تعریف شمای خارجی کاربر ۱ (با استفاده ار مفهوم دید):**

```
CREATE   VIEW   V1 [(STID,  STNAME)]
        AS   SELECT   STT.STID,   STT.STNAE
                FROM   STT;

CREATE   VIEW   V2   [(SN, SJ, SL)]
        AS   SELECT   STID, STJ, STL
                FROM   STT
                WHERE   STJ   !=   'phys;
                [WITH  CHECK  OPTION]
```

شرط تعریف دید

- در شرط تعریف دید می‌توان از نام ستونی که در محدوده دید نیست استفاده کرد.

تعریف شمای خارجی (ادامه)

نکته: هر کاربر با اجازه Admin می‌تواند دید (View) خودش را داشته باشد (Sub-database).

نکته: دستور SELECT در متن دستور تعریف دید اجرایی نیست بلکه اعلانی است

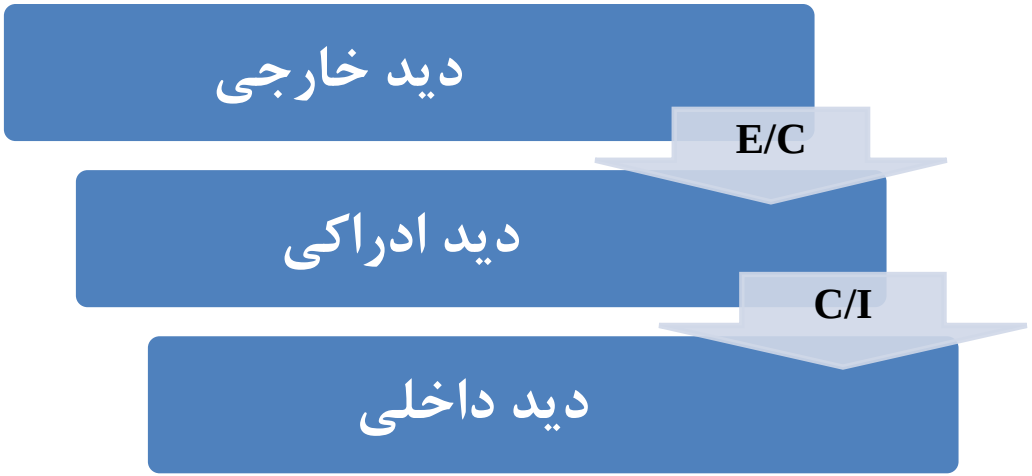
- یعنی هیچ داده‌ای بازیابی نمی‌شود و صرفا برای اعلام محدوده داده‌ای کاربران است.

نکته: تا آنجا که به تعریف دید مربوط است هر دستور SELECT معتبر با هر میزان پیچیدگی را می‌توان در CREATE VIEW نوشت.

- **تمرین:** مثال کاتالوگ پیش‌دیده را به نحوی گسترش دهید که اطلاعات (نه داده‌ها) شمای داخلی و شمای خارجی دیده شده را بتوان در آن ذخیره کرد (جدول دیگری برای کاتالوگ تعریف کنید که بتوان این شماها را در آن ذخیره کرد).

نگاشت بین سطوح در عملیات سطح شمای خارجی

- نگاشت یا تبدیل بین سطوح (عملیات از دید خارجی در DB):
  - **E/C**: External to Conceptual Mapping
  - **C/I**: Conceptual to Internal Mapping



کنجکاوی؟ آیا تبدیل دیگری هم متصور است؟

عملیات در شمای خارجی

- عملیات در شمای خارجی
  - **بازیابی:** کاربر حق دارد در محدوده دید خود عمل بازیابی انجام دهد.
  - **ذخیره‌سازی:** به تشخیص Admin مجاز به انجام است.
    - درج
    - حذف
    - بروزرسانی

- هر دستور [حکم] عمل‌کننده در شمای خارجی (روی دید خارجی)،
  - تبدیل می‌شود به دستور(های) عمل کننده در شمای ادراکی (روی دید ادراکی)
  - و سپس به قطعه برنامه‌ای عمل کننده در شمای داخلی (روی دید داخلی)
  - و نهایتاً به عملیاتی در فایل‌های فیزیکی.

عملیات بازیابی از دید

مثال

عملیات بازیابی: چون دید خارجی در سیستم‌های جدولی، به هر حال نوعی جدول است، برای بازیابی از همان دستور SELECT استفاده می‌کنیم.

```
SELECT V2.SN
        FROM V2
        WHERE SL='ms'
```

E/C

- سیستم در نگاشت E/C، شرط یا شرایط داده شده در تعریف دید را AND می‌کند با شرط یا شرایط داده شده در پرس‌وجوی روی دید. به این عمل، گاه **محاسبه دید** (View Computation) هم می‌گویند.

```
SELECT STT.STID
        FROM STT
        WHERE STL='ms'
          AND STJ != 'phys'
```

C/I

عملیات بازیابی از دید (ادامه)

در محیط فایلینگ منطقی

به واحد رکورد

ناحیه پیام بافر سیستم

```
OPEN STFILE (R,  SysBuf,  MessageArea, …)
LREAD STFILE ON STLINDEX.value='ms';
        …
        IF SysBuf.STJ != 'phys'
           MOVE SysBuf.STID INTO UBuf[SN]
            …
           LOOP Control;
```

در محیط فایلینگ فیزیکی

PSEEK جستجوی فیزیکی

PREAD خواندن فیزیکی

به واحد بلاک

عملیات ذخیره‌سازی از دید

- لزوماً از همه انواع دیدها نمی‌توان عملیات ذخیره‌سازی در DB انجام داد.
  - همه انواع دیدها قابل بروزرسانی (Updatable) نیستند.
  - محدودیتهایی هم در عمل و تاحدی در تئوری وجود دارد.

- **دید از نظر قابلیت عملیات ذخیره‌سازی** (بستگی دارد به ساختار دید و مکانیزم تعریف آن)
  - **پذیرا** (Updatable): می‌توان از آنها عملیات ذخیره‌سازی انجام داد ولی گاه مشکلاتی دارند.
  - **ناپذیرا** (Non Updatablen): تبدیل E/C انجام شدنی نیست.

- دید
  - تعریف شده روی یک جدول مبنا
  - تعریف شده روی بیش از یک جدول مبنا ← در عمل ناپذیرا، اما در تئوری بعضی‌ها پذیرا هستند.

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا

- دید تعریف شده روی یک جدول مبنا

- دید دارای کلید جدول مبنا (Key Preserving) ← پذیرا (در عمل و تئوری) اما مشکلاتی هم دارد.
- دید فاقد کلید جدول مبنا (Non Key Preserving) ← ناپذیرا
- دید دارای ستون [صفت] مجازی (دیدهای آماری) ← ناپذیرا

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

- دید حافظ کلید تعریف شده روی یک جدول مبنا

V2

SN	SL	SJ
888	ms	math
444	bs	comp
⋮	⋮	

STT

STID	STNAME	STL	STJ	STD
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

```
CREATE  VIEW  V2  [(SN, SJ, SL)]
        AS  SELECT  STID, STJ, STL
                FROM  STT
                WHERE  STJ  !=  'phys'
                [WITH  CHECK  OPTION]
```

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

- فرض بر مجاز بودن کاربر به انجام عمل داریم و لذا صرفاً شدنی بودن را بررسی می‌کنیم.
- در **دید حافظ کلید** انجام عملیات سطری امکان‌پذیر است.
  - زیرا تناظر یک به یک بین سطرهای دید و سطرهای جدول مبنا برقرار است.

مثال

حذف سطر در دید حافظ کلید

DELETE FROM V2
WHERE SN=‘444’

E/C

DELETE FROM STT
WHERE STID=‘444’ **AND** STJ != ‘phys’

- الان این سطر از جدول STT حذف می‌شود و اگر کاربر دیگری این سطر را در دیدش داشته باشد، دیگر به این سطر دسترسی ندارد.

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

مثال بروزرسانی در دید حافظ کلید

UPDATE V2

SET SJ=‘IT’

WHERE SN=‘444’

E/C

UPDATE STT

SET STJ=‘IT’

WHERE STID=‘444’ ’ **AND** STJ != ‘phys’

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

مثال: از نظر تئوریک درخواست زیر به دلیل عدم رعایت محدودیت دید باید رد شود.

```
UPDATE V2
        SET SJ='phys'
        WHERE SN='888'
```

- در عمل: اگر از عبارت [with check option] استفاده کنیم، سیستم رد می‌کند، وگرنه درخواست انجام می‌شود اما ...

E/C

```
UPDATE STT
        SET STJ='phys'
        WHERE STID='888' '  AND  STJ != 'phys'
```

- حال اگر بنویسیم:

```
SELECT V2.*  FROM V2
```

- سطر با کلید 888 دیگر در دید کاربر نمی‌آید!

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

مثال

INSERT INTO V2
VALUES (‘555’, ‘chem’, ‘bs’)

E/C

INSERT INTO STT
VALUES (‘555’, ?, ‘chem’, ‘bs’, ?)

- اگر هر کدام از ستون‌های نهان از دید کاربر، محدودیت هیچ‌مقدارناپذیری داشته باشند، درخواست رد می‌شود.
- حال اگر به جای 555 بنویسیم 777، درخواست رد می‌شود (تبدیل E/C انجام نمی‌شود) به دلیل عدم رعایت محدودیت یکتایی مقادیر کلید.
- حال اگر به جای chem بنویسیم phys، همان پیش می‌آید که در مثال UPDATE دیدیم.

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

- دلایل رد شدن درخواست عمل ذخیره‌سازی در دید تک جدولی حافظ کلید:
  - عدم رعایت محدودیت دید
  - عدم رعایت محدودیت یکتایی مقادیر کلید
  - عدم رعایت محدودیت هیچ‌مقدارناپذیری ستون‌های نهان
  - ...

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

- **دید تعریف شده روی یک جدول مبنا و فاقد کلید**

مثال

چون این دید فاقد کلید است، امکان انجام عملیات سطری وجود ندارد.

```
CREATE VIEW V3
        AS SELECT STNAME, STJ
        FROM STT
```

- درخواست زیر انجام نمی‌شود، چون معلوم نیست کدام سطر از رابطه باید حذف شود. پس تبدیل E/C ناممکن است، مگر اینکه بپذیریم این درخواست به صورت مکانیکی انجام شود؛ یعنی تمام سطرهای حائز شرط داده شده (مجموعه‌ای از سطرها) حذف شوند.

```
DELETE FROM V3
        WHERE STNAME='ali'
```

- اگر کاربر این پیامد را بپذیرد مشکلی نیست، اما در عمل سیستم‌ها نمی‌پذیرند!
- در دید V3 انجام INSERT نیز غیرممکن است.

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

مثال

حال اگر در تعریف V3، DISTINCT بزنیم چه پیش می‌آید؟

```
CREATE VIEW V3
        AS SELECT DISTINCT STNAME, STJ
        FROM STT
```

- فرقی نمی‌کند، باز هم همان مشکل پابرجاست:

```
DELETE FROM V3
        WHERE STNAME='a'
```

E/C

تبدیل می‌شود به حذف مجموعه‌ای از سطرها

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

- دید تعریف شده روی یک جدول مبنا دارای ستون مجازی
  - این دیدها هم در عمل و هم در تئوری ناپذیرا هستند.

مثال

V4

PN	SQ
P1	100
P2	210
P3	80

```
CREATE VIEW V4 (PN, SQ)
        AS SELECT P#, SUM(QTY)
        FROM SP
        GROUP BY P#
```

SP

S#	P#	QTY
S1	P1	100
S1	P2	140
S2	P3	80
S2	P2	70

عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

- انجام عملیات سطری در دید V4 غیرممکن است.

DELETE FROM V4
WHERE PN='p1'

→ سطری نیست، با نوعی تفسیر می توان گفت که مجموعه‌ای از سطرها را حذف می‌کند.

- از لحاظ تئوریک هم دید V4 نباید پذیرا باشد.
- زیرا جدول V4 (که مجازی است) و جدول مبنای SP با هم تعارض معنایی (Semantic Conflict) دارند. یعنی مسند بیانگر معنای رابطه V4 اساساً با مسند بیانگر رابطه SP تفاوت دارد.

[دربحث رابطه‌ای خواهیم دید که هر رابطه (جدول) یک معنا دارد و در اینجا این دو رابطه هیچ ربطی از نظر معنایی با هم ندارند.]

عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا

- دیدهای تعریف شده روی بیش از یک جدول
  - در عمل این دیدها ناپذیرا هستند و دخالت خود برنامه‌ساز لازم است.

V5: T1 **JOIN** T2 **دید پیوندی (پیوند طبیعی)**

V6: T1 **UNION** T2

V7: T1 **INTERSECT** T2

V8: T1 **EXCEPT** T2

- دید پیوندی
  - PK-PK: ستون پیوند در هر دو جدول PK است. پذیرا و بدون مشکل
  - PK-FK: پذیرا به شرط پذیرش پیامدها
  - FK-FK
  - (Non-Key) NK-NK

عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

دید پیوندی PK-PK

مثال

V5 همان STT است اما این بار به صورت یک دید تعریف شده است.

V5

STID	STNAME	STL	STJ	STD
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

```
CREATE VIEW V5
        AS SELECT ST1.*, ST2.*
        FROM ST1  JOIN  ST2
```

ST1

STID	STNAME	STL
777	st7	bs
888	st8	ms
444	st4	bs
⋮	⋮	⋮

ST2

STID	STJ	STD
777	phys	d11
888	math	d12
444	comp	d14
⋮	⋮	⋮

عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

- یک دستور اجراشونده در شمای خارجی تبدیل می‌شود به دو دستور در شمای ادراکی.

INSERT INTO V5
VALUES (‘999’, ‘St9’, ‘chem’, ‘bs’, ‘D15’)

E/C

INSERT INTO ST1
VALUES (‘999’, ‘St9’, ‘bs’)

INSERT INTO ST2
VALUES (‘999’, ‘chem’, ‘D15’)

- عمل DELETE در این دید تبدیل می‌شود به دو عمل حذف از جدول‌های مبنایی زیرین و عمل UPDATE (بسته به ستونی که می‌خواهیم بروز کنیم) به یک یا دو عمل بهنگام‌سازی در جدول‌های زیرین تبدیل می‌شود.

عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

- دید پیوندی PK-FK

مثال

```
CREATE VIEW V6
        AS SELECT STT.STID, STT.NAME, STCOT.*
        FROM STT JOIN STCOT
```

- درج در این دید تبدیل می‌شود به درج یک تاپل ناقص در STT به شرط آنکه شماره دانشجویی تکراری نباشد. ولی در STCOT حتما یک تاپل درج می‌شود.

```
INSERT INTO V6
        VALUES ('9212345', 'Amir', '40638', 15)
```

E/C

```
INSERT INTO STT
        VALUES ('9212345', 'Amir', ?, ?, ?)

INSERT INTO STCOT
        VALUES ('9212345', '40638', 15)
```

عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

- حذف از این دید مشکل دارد.
- اگر از هر دو جدول حذف شود، منجر به حذف داده‌های ناخواسته می‌شود.
- با حذف یک سطر از جدول STT، برای حفظ جامعیت ارجاعی نیز لازم است یک تعداد سطر دیگر از STCOT حذف شود، مگر آنکه فقط از STCOT حذف کنیم و از STT سطر مربوطه را حذف نکنیم.
- عمل بهنگام‌سازی نیز مساله مشابه حذف ممکن است داشته باشد.

ستون پیوند در هر دو جدول کلید خارجی است.

ستون پیوند در هیچ کدام کلید نیست (نه اصلی و نه خارجی).

کجای؟ دید حاصل از پیوند FK-FK و دید حاصل از پیوند NK-NK چه رفتاری در عملیات ذخیره‌سازی دارند؟

انجام عملیات ذخیره‌سازی در این دیدها دارای عوارض بسیاری است که در عمل آنها را ناپذیرا می‌کند.

عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

- دید حاصل از اجتماع، اشتراک، و تفاضل
  - این دیدها از لحاظ تئوری مشکلی در عملیات ذخیره‌سازی ندارند، هرچند نظرات مختلفی مطرح است.

عمل / دید	درج	حذف	بهنگام‌سازی
$R_1 \cup R_2$	درج تاپل در R_1 و/یا R_2	حذف تاپل از R_1 و/یا R_2	بهنگام‌سازی تاپل در R_1 و/یا R_2
$R_1 \cap R_2$	درج تاپل در R_1 و R_2	حذف تاپل از R_1 و/یا R_2	بهنگام‌سازی تاپل در R_1 و R_2
$R_1 - R_2$	درج تاپل در R_1 (به شرط عدم وجود در R_2)	حذف تاپل در R_1	بهنگام‌سازی تاپل در R_1

دیدهای پذیرا در SQL

- موضوع دیدهای پذیرا در SQL استاندارد چندان روشن نیست. در SQL 2003 دیدهایی که تمام شرایط زیر را داشته باشند، قابل بهنگام‌سازی (درج، حذف و بروزرسانی) هستند.

  [توجه: ممکن است برخی دیگر از دیدها هم قابل بهنگام‌سازی باشند.]

  ۱- عبارت تعریف‌کننده دید، یک عبارت SELECT ساده باشد (یعنی شامل عملگرهای JOIN، UNION، INTERSECT و EXCEPT نباشد).

  ۲- در عبارت SELECT گزینه DISTINCT وجود نداشته باشد.

  ۳- در کلاز FROM عبارت SELECT، فقط یک جدول وجود داشته باشد.

  ۴- جدول قید شده در کلاز FROM، یک جدول مبنا یا یک دید قابل بهنگام‌سازی باشد.

دیدهای پذیرا در SQL (ادامه)

۵- در لیست نام ستون‌ها در عبارت SELECT، ستون‌های موردنظر باید در جدول مبنا متناظر داشته باشند و به یک ستون از جدول مبنا بیش از یک بار ارجاع وجود نداشته باشد. ضمناً حاوی ستون کلید باشد.

۶- در عبارت SELECT، کلاز GROUP BY و/یا کلاز HAVING وجود نداشته باشد.

۷- کلاز WHERE در عبارت SELECT حاوی کلاز FROM نباشد به گونه‌ای که در آن به همان جدولی ارجاع داده شده باشد که در کلاز FROM ذکر شده در شرط ۴.

- نتیجه اینکه عملاً دیدهایی که یک زیرمجموعه افقی-عمودی دارای کلید از یک جدول مبنا (یا از دید قابل بهنگام‌سازی) باشند، قطعاً قابل بهنگام‌سازی هستند.

[**توجه:** به شرط رعایت محدودیت‌های جامعیتی مانند یکتایی کلید و هیچمقدارناپذیری]

مزایا و معایب مفهوم دید خارجی

- **معایب مفهوم دید:**
  - محدودیت [و مشکلات] در عملیات ذخیره‌سازی
  - فزونکاری (overhead) برای انجام تبدیل E/C (محاسبه دید) . راه حل: *استفاده از تکنیک دید ذخیره شده*

مزایا و معایب مفهوم دید خارجی (ادامه)

- **مزایای مفهوم دید خارجی:**
  - فراهم‌کننده محیط انتزاعی فرافایلی برای کاربران با پویایی بالا
  - اشتراک داده‌ها (Data Sharing) ← داده‌ها یک بار ذخیره می‌شوند و کاربران بسته به نیاز خود از داده‌های ذخیره شده به صورت همروند استفاده می‌کنند.
  - تامین امنیت برای داده‌های زیرین. ← از طریق مفهوم داده مخفی (Hidden Data)، زیرا کاربر خارج از محدوده دید خود هیچ نمی‌بیند (داده‌های نهان امن هستند).
  - تامین‌کننده استقلال داده‌ای (مفهوم اساسی در تکنولوژی DB؛ هم مزیت و هم از اهداف مهم تکنولوژی DB).
  - امکانی است برای کوتاه‌نویسی یا ماکرونویسی پرسش‌ها.

دید ذخیره شده

- **تکنیک دید ذخیره شده [ساخته شده] (Materialized View)**
- در این تکنیک، دید در سیستم ذخیره می‌شود؛ یعنی دیگر مجازی نیست و جدول ذخیره شده است. تا در هر بار مراجعه به دید لازم نباشد تبدیل E/C انجام شود.
- **هدف:** برای افزایش سرعت عملیات بازیابی.
- **شرط استفاده:** در عمل از این تکنیک وقتی استفاده می‌کنیم که داده‌های ذخیره شده در جدول‌های مبنای زیرین حتی‌الامکان تغییر نکنند. به بیان دیگر، نرخ عملیات ذخیره‌سازی در جدول‌های زیرین پایین باشد. زیرا اگر جدول‌های زیرین تغییر کنند، تغییرات متناسباً در جدول‌های دید باید اعمال شوند و این خود سربار ایجاد می‌کند.
- **کاربرد:** در برنامه‌های آماری، گزارش‌گیری‌ها و برنامه‌های داده‌کاوی (Data Mining)
- دید ذخیره شده (Stored View) در SQL چگونه پیاده‌سازی می‌شود؟ با CREATE SNAPSHOT.

دلایل عدم استفاده از دید

- چه زمانی از مفهوم دید استفاده نمی‌کنیم؟
  - هنگامی که سیستم تک‌کاربره باشد.
  - هنگامی که به تشخیص admin برای افزایش کارایی سیستم، برخی برنامه‌ها را مستقیماً روی شمای ادراکی (جداول مبنایی) بنویسیم.
  - هنگامی که کاربر نیازمند انجام عملیات ذخیره‌سازی باشد و از طریق دید امکان آن وجود نداشته باشد.

استقلال داده‌ای

- **مفهوم استقلال داده‌ای [DI] (جدایی برنامه‌ها از داده‌ها):**
  - مصونیت (تاثیرناپذیری) برنامه‌های کاربران [در سطح خارجی] در قبال تغییرات در سطوح زیرین معماری DB.

- چرا نباید برنامه‌ها تغییر کنند؟
  - چون هر تغییر در برنامه‌ها، هزینه تولید و پشتیبانی و بازتولید برنامه‌ها را بالا می‌برد.

- استقلال داده‌ای (DI)
  - استقلال داده‌ای فیزیکی (PDI)
  - استقلال داده‌ای منطقی (LDI)

استقلال داده‌ای فیزیکی

- **استقلال داده‌ای فیزیکی (PDI)**
  - مصونیت برنامه‌های کاربران در قبال تغییرات در شمای داخلی DB
  - تغییرات در شمای داخلی شامل تغییر در جنبه‌های فایلینگ پایگاه
    - ساختار فایل، طول رکورد، طرز ذخیره‌سازی فایل روی دیسک، گاه با دخالت طراح فیزیکی و گاه فقط توسط DBMS.
  - استقلال داده‌ای فیزیکی در پایگاه داده‌ها به طور کامل تامین است: زیرا کاربران با مفهوم دید کار می‌کنند که اساساً در سطح فرافایلی مطرح است و برنامه‌ها درگیر جنبه‌های فایلینگ نیستند.

- **استقلال داده‌ای منطقی (LDI)**
  - مصونیت برنامه‌های کاربران در قبال تغییرات در شمای ادراکی DB.
  - در سیستم‌های پایگاهی تا حد زیادی این استقلال تامین است ولی نه صددرصد.
- **تغییر در شمای ادراکی**
  - رشد پایگاه داده‌ها (DB Growth)
  - تغییر سازمان پایگاه داده‌ها [سازماندهی مجدد DB] (DB Restructuring)
- **نکته:** نغییراتی که مورد بررسی قرار می‌دهیم، تغییراتی است که از داده‌ها و ساختار موجود نمی‌کاهد، چرا که تغییرات کاهشی، قطعاً بر روی برنامه‌های سطح خارجی تاثیر می‌گذارد و استقلال داده‌ای حفظ نمی‌شود.

استقلال داده‌ای منطقی (ادامه)

- **چرا رشد DB:** مطرح شدن نیازهای جدید
  - اضافه شدن ستون(های) جدید به جدول(ها)
  - ایجاد جدول‌های جدید
- استقلال داده‌ای منطقی (LDI) در قبال رشد DB، به کمک مفهوم دید تقریباً صددرصد تامین است، زیرا کاربرِ دارای یک دید، خارج از محدوده آن دید هیچ نمی‌بیند.

استقلال داده‌ای منطقی (ادامه)

مثال

دیدهای پیش‌تر تعریف شده را روی جدول STT در نظر می‌گیریم. حال نیاز جدیدی برای کاربر مطرح شده است.

کاربر ۱

V1

STID	STNAME
⋮	⋮

V2

⋮	⋮

کاربر ۲

V9

STID	STADR
⋮	⋮

STT

STID	STNAME	STL	STJ	STD	STADR
777	st7	bs	phys	d11	
888	st8	ms	math	d12	
444	st4	bs	comp	d14	
⋮	⋮	⋮	⋮	⋮	

ALTER TABLE STT
ADD COLUMN STADR CHAR(70)

- این گسترش در سطح فایلینگ چگونه انجام می‌شود؟

استقلال داده‌ای منطقی (ادامه)

- آیا پیرو نیاز جدید کاربر در حد ستون، طراح همیشه جدول مبنا را گسترش می‌دهد؟
  - خیر، زیرا ممکن است آن ستون مجازی (محاسبه شدنی) باشد.
- **سازماندهی مجدد DB** یعنی طراح به هر دلیلی طراحی منطقی DB را تغییر دهد. مثلاً یک جدول مبنای موجود را به دو جدول تجزیه عمودی کند و طبعاً شمای ادراکی هم تغییر می‌کند. می‌خواهیم ببینیم LDI در قبال این تغییر تا چه حد تامین است.
- در این حالت، LDI به کمک مفهوم دید و امکان تعریف دید روی دید (View Definition on View)، تا حدی تامین است.

استقلال داده‌ای منطقی (ادامه)

مثال

برنامه‌ها

V1 V2

STT

STID	…	STD
777		
888		
444		
⋮	⋮	⋮

V1 V2

E/E

STT

STID	…	STD
777		
888	جدول مجازی: دید	
444		
⋮	⋮	⋮

E/C

ST1

STID	STNAME	STL
777	st7	bs
888	st8	ms
444	st4	bs
⋮	⋮	⋮

ST2

STID	STJ	STD
777	phys	d11
888	math	d12
444	comp	d14
⋮	⋮	⋮

استقلال داده‌ای منطقی (ادامه)

- **شمای جدید:**
- البته در عمل مشکلات دیگری هم وجود دارد و صرفاً با این اَعمال مشکل برطرف نمی‌شود.

```
CREATE TABLE ST1
        (STID …,
        …
        STL …)
        PRIMARY KEY STID

CREATE TABLE ST2
        (STID …,
        …
        STD …)
        PRIMARY KEY STID

INSERT INTO ST1
        (SELECT STID, STNAME, STL
        FROM STT)

INSERT INTO ST2
        (SELECT STID, …, STD
        FROM STT)
```

مهاجرت داده‌ها (Data Migration)

استقلال داده‌ای منطقی (ادامه)

- دلایل این نوع تجزیه (که کلید در هر دو جدول باشد) چه می‌تواند باشد؟
- افزایش کارایی سیستم در رده فایلینگ با فرض 1-Table:1-File برای بعض برنامه‌ها (مثلاً برنامه‌هایی با فرکانس بالاتری نسبت به ستون‌های ST1 و با فرکانس پایین‌تری به ستون‌های ST2 ارجاع داشته باشد، فایل‌ها را جدا می‌کند).
- توزیع داده‌ها در سایت‌ها وقتی پایگاه داده توزیع شده (DDB) داشته باشیم.
- کاهش حجم Null Value
- بهینه‌سازی طراحی (رجوع شود به بحث نرمال‌سازی رابطه‌ها)
- ...

استقلال داده‌ای منطقی (ادامه)

- با حذف جدول مبنای STT، دیدهای قبلاً تعریف شده روی آن نامعتبر می‌شوند و در نتیجه برنامه‌هایی که روی آنها کار می‌کردند، دیگر اجرا نمی‌شوند و LDI دیگر تامین نیست مگر اینکه طراح و پیاده‌ساز تدبیری بیندیشد.

↓

- ✓ جدول STT را با همان نام و ساختار به شکل یک دید تعریف می‌کنیم، با مکانیزم پیوند (**دید روی دید**):

```
CREATE VIEW STT
        AS SELECT STID, …, STD
        FROM  ST1 JOIN ST2

DROP TABLE STT
```

- تعریف این دید وارد کاتالوگ سیستم می‌شود. ← دیدهای قبلاً تعریف شده معتبر می‌شوند.

استقلال داده‌ای منطقی (ادامه)

- با این تدبیر، LDI برای برنامه‌هایی که بازیابی انجام می‌دهند، صددرصد تامین می‌شود، به قیمت افزایش سربار برای انجام تبدیل E/E علاوه بر E/C و C/I. زیرا از تکنیک **دید روی دید** استفاده کرده‌ایم.

- اما LDI برای برنامه‌هایی که عملیات ذخیره‌سازی انجام می‌دادند، ممکن است تامین نباشد. زیرا این بار STT خود یک دید است و دیدها در عملیات ذخیره‌سازی عمدتاً مشکل دارند. ولی در این مثال خاص از نظر تئوریک مشکلی بروز نمی‌کند. ← چون STT یک دید پیوندی PK-PK است.

پرسش و پاسخ . . .

amini@sharif.edu

به نام آنکه جان را فکرت آموخت

بخش ششم:
مفاهیم اساسی مدل داده رابطه‌ای

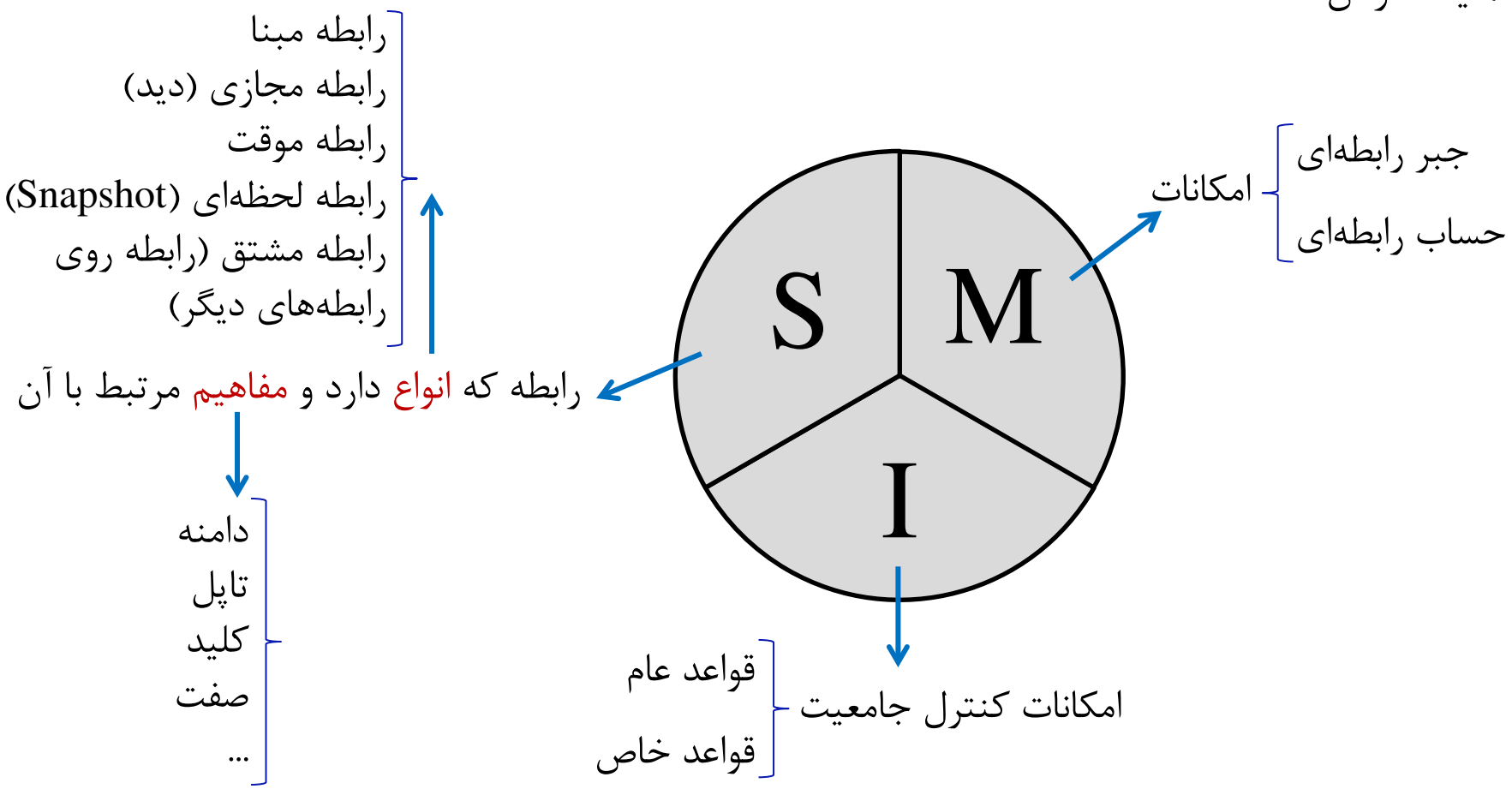
مرتضی امینی

نیمسال دوم ۹۴-۹۵

(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)

- RDM مبنای تئوریک RDB و RDBMS
- واضع مدل: F. Codd
- مفاهیم زیر در طی سه بخش باقیمانده از این درس مرور می‌شوند:
  - رابطه (Relation)
  - دامنه (میدان)
  - رابطه نرمال و غیرنرمال
  - کلید در مدل رابطه‌ای
  - قواعد جامعیت رابطه‌ای
  - عملیات در RDB ← جبر رابطه‌ای / حساب رابطه‌ای
  - طراحی RDB ← روش بالا به پایین / روش نرمال‌ترسازی (سنتز)

✓ **مدل داده** مجموعه‌ای است از امکانات برای طراحی منطقی و تعریف پایگاه داده‌ها، کنترل آن و نیز انجام عملیات در آن.



در ریاضی: هر زیر مجموعه از ضرب کارتزین چند مجموعه

(۱) با فرض وجود m مجموعه از مقادیر موسوم به دامنه [میدان] D_1، ...، D_m:

رابطه R با صفات A_1، ...، A_m تعریف شده روی این m دامنه

مجموعه‌ای است از عناصر، هر یک به صورت $\langle d_{1i}, d_{2i}, \ldots, d_{mi} \rangle$ موسوم به m-تاپل (m-tuple)

به نحوی که $d_{1i} \in D_1$، . . . ، $d_{ji} \in D_j$

مثال

STUD (STID,	STNAME,	STJ,	STL,	STD)
777	st7	bs	phys	d11
⋮	⋮	⋮	⋮	⋮
444	st4	bs	comp	d14

یک تاپل ۵-تایی

رابطه (ادامه)

(۲) [Date] با فرض وجود m مجموعه از مقادیر موسوم به دامنه [میدان] D_1، ...، D_m نه لزوماً متمایز، رابطه R تعریف شده روی این m دامنه:

دومجموعه:
- عنوان [سرآیند] (Heading): مجموعه‌ای است نامدار از اسامی صفات یعنی $\{A_1,\ldots,A_m\}$ که با $R(A_1,\ldots,A_m)$ نمایش داده می‌شود.
- بدنه [پیکر] (Body): مجموعه ای است از تاپل‌ها [همان مجموعه در تعریف اول].

مثال: رابطه دانشجو

STUD (STID, STNAME, STJ, STL, STD)

- **درجه رابطه:** کاردینالیتی عنوان یا تعداد صفات رابطه

m	اصطلاح
۱	رابطه یگانی
۲	رابطه دوگانی
n	رابطه nگانی

رابطه (ادامه)

- مجموعه عنوان را با H_R یا R(H) نیز نمایش می‌دهیم. به R(H)، Intention (ذات، جوهر یا **چکیده**) رابطه هم گفته می‌شود.
  - R(H) ثابت در زمان است. یعنی اگر مجموعه صفات را عوض کنیم، از نظر ریاضی یک رابطه دیگر است.
  - همین R(H) برای تعریف رابطه در سیستم کافی است.

مثال

CREATE RELATEION STUD
(STID, STNAME, STJ, STL, STD)

- هررابطه یک معنا دارد، بیانگر واقعیتی از یک محیط مشخص. به عنوان مثال وقتی می‌گوییم رابطه STUD را داریم، معنایش این است که در خردجهان واقع، نوع موجودیتی با نام STUD و با صفات STID و STNAME و ... و STD وجود دارد.

رابطه (ادامه)

- **کاردینالیتی رابطه:** همان کاردینالیتی بدنه؛ تعداد تاپل‌ها (بزرگتر مساوی صفر؛ صفر در بدو تعریف)
  - بدنه رابطه، متغیر در زمان است.
  - به یک مقدار بدنه در یک لحظه مشخص instance گویند.
  - به بدنه رابطه، Extension (**بسط** یا گسترده) یا حالت رابطه گویند.

رابطه (ادامه)

تعریف (۳) [از نظر تئوری زبان‌های برنامه‌سازی] تشکیل شده است از یک متغیر رابطه‌ای و در هر لحظه از یک مقدار رابطه‌ای.

- R(H): متغیر رابطه‌ای، متغیری از جنس رابطه Relation Variable [RELVAR]
- بدنه (r): مقدار رابطه‌ای Relation Value [RELVAL]

R(A, B) → متغیر رابطه‌ای

a_1	b_1
a_2	b_2
⋮	⋮
a_n	b_n

→ یک مقدار رابطه‌ای
(در لحظه بعد ممکن است مقدارش فرق کند.)

تناظر بین مفاهیم رابطه‌ای و اصطلاحات جدولی

مفهوم رابطه‌ای	اصطلاح
رابطه	جدول (صرفاً امکانی است برای نمایش مفهوم رابطه‌ای و تفاوت‌های متعددی با رابطه دارد.)
تاپل	سطر
صفت	ستون
دامنه	مقادیر مجاز ستون
درجه	تعداد ستون‌ها
کاردینالیتی	تعداد سطرها
کلید	؟ (به معنایی که در مدل رابطه‌ای داریم، در بحث‌های جدولی مطرح نیست.)

ویژگی‌های رابطه:

۱- صفات در عنوان رابطه نظم (مکانی) ندارند. [چون مجموعه است] $R(A, B) = R(B, A)$

در حالی که در جدول، ستون‌ها می‌توانند نظم مکانی داشته باشند.

در مدل رابطه‌ای، تنها راه ارجاع به صفت رابطه، نام صفت است.

۲- تاپل‌ها [در بدنه] نظم ندارند (مرتب نیستند) [چون مجموعه است].

۳- رابطه، تاپل تکراری ندارد [چون مجموعه است].

۴- تمام صفات رابطه (نرمال)، تک مقدار هستند [رجوع شود به مفهوم رابطه نرمال] (این ویژگی دلیل تکنیکی دارد و از ذات رابطه نتیجه نمی‌شود). یعنی در هر تاپل دقیقاً یک مقدار برای هر صفت وجود دارد.

- در RM هیچ یک از مفاهیم فایلینگ مطرح نیستند (مثل نظم، فیلد، رکورد، اشاره‌گر، آدرس که در سطح طراحی و فایلینگ فیزیکی مطرح است).

مدل رابطه‌ای و مدل جدولی (ادامه)

تفاوت‌های مفهوم رابطه و اصطلاح جدول

- ۳ ویژگی اول رابطه، ۳ تفاوت

۴- در رابطه m>=0 (درجه)، یعنی از نظر تئوری رابطه می‌تواند از نظر درجه، صفر باشد.

۵- رابطه می‌تواند بیش از دو بُعد داشته باشد (مثلا Data Cube).

۶- نمایش دقیق عنوان رابطه به صورت زیر است حال آنکه عنوان جدول چنین نیست.

عنوان رابطه مجموعه‌ای است از دوتایی‌های ⟨دامنه:صفت⟩ $R(H): \{\langle D_1: A_1\rangle, \langle D_2: A_2\rangle, \ldots\}$

۷- نمایش دقیق تاپل رابطه به صورت زیر است حال آنکه سطر در جدول چنین نیست.

تاپل مجموعه‌ای است از سه‌تایی‌های ⟨دامنه، صفت، مقدار⟩ $TUPLE: \{\langle D_1: A_1:V_1\rangle, \langle D_2: A_2:V_2\rangle, \ldots\}$

۸- رابطه نمی‌تواند هیچ‌مقدار داشته باشد، ولی جدول می‌تواند (البته در این خصوص اختلاف نظر وجود دارد).

دامنه [میدان]

- مفهوم دامنه (میدان)
  - مجموعه‌ای است نامدار از مقادیر هم نوع، که حداقل یک صفت از رابطه، از آن **معنا**، **نوع** و **مقدار** می‌گیرد.
  - معادل است با مفهوم Data Type در تئوری انواع.
  - دامنه‌هایی که یک رابطه روی آن‌ها تعریف می‌شود، لزوماً متمایز نیستند.

مفروض $R(H)$

if $A_i \in H,\ A_j \in H,\ A_i \neq A_j \Rightarrow (D_i \neq D_j$ لزوما چنین نیست که$)$

دامنه [میدان] (ادامه)

- **تمرین:** مثالی از یک رابطه ۵-تایی که
  - دوصفت آن از یک دامنه باشد.
  - سه صفت آن از یک دامنه باشد.
- اگر m درجه رابطه و n تعداد دامنه‌ها باشد، داریم: $n \leq m$.
- برای تعریف یک رابطه در سیستم رابطه‌ای، از لحاظ تئوریک، ابتدا باید دامنه‌هایش را تعریف کرد.

دامنه [میدان] (ادامه)

مثال

مثالی از شمای پایگاه رابطه‌ای (در مدل تئوریک)

```
CREATE DOMAIN  SN        CHAR(8)  DEFAULT '00000000'
CREATE DOMAIN  SNAME     CHAR(20) DEFAULT 'noname'
CREATE DOMAIN  SJ        CHAR(4)  DEFAULT '?...?'
CREATE DOMAIN  SL        CHAR(3)  DEFAULT '?...?'
CREATE DOMAIN SD         CHAR(4)  DEFAULT '?...?'
CREATE DOMAIN CN         CHAR(6)  DEFAULT '?...?'
CREATE DOMAIN GRADE      DEC(2, 2) DEFAULT '?...?'

CREATE RELATEION STT
        (STID  DOMAIN  SN,
         STNAME  DOMAIN  SNAME,
         STJ  DOMAIN  SJ,
         STL  DOMAIN  STL,
         STD  DOMAIN  SD)

CREATE RELATION COT ….

CREATE RELATION STCOT …
```

دامنه [میدان] (ادامه)

کنجکاوی؟ دستورات زیر در SQL مطالعه شود.

- CREATE DOMAIN
- ALTER DOMAIN
- DROP DOMAIN

کنجکاوی؟ مزایای مفهوم دامنه از دیدگاه مهندسی نرم‌افزار بررسی شود.

رابطه نرمال و غیرنرمال

- **رابطه نرمال (بهنجار- عادی Flat Relation):**

تعریف: رابطه‌ای که تمام صفات آن تک‌مقداری (حداکثر دارای یک مقدار در هر تاپل) باشند.

- **رابطه غیرنرمال (Nested Relation):**

تعریف: رابطه‌ای که حداقل یک صفت آن چندمقداری باشد.

- **توجه:** تعریف زیر درست **نیست:**
  - رابطه‌ای نرمال است که مقادیر تمام صفات آن اتمیک (تجزیه نشدنی یا ساده) باشند.
  - **تذکر:** ساده یا مرکّب بودن صفت نقشی در نرمال بودن و نبودن آن ندارد.

رابطه نرمال و غیرنرمال (ادامه)

مثال

صفت چندمقداری ساده

NNCOPRECO (COID , PRECOID)

COID	PRECOID
c01	{c11, c17, c08}
c02	{c03, c09}
c03	c10

یک تاپل

تبدیل به

رابطه نرمال

COPRECO (COID , PRECOID)

COID	PRECOID
c01	c11
c01	c17
c01	c08
c02	c03
c02	c09
c03	c10

یک تاپل

رابطه نرمال و غیرنرمال (ادامه)

مثال

صفت چندمقداری مرکب
P# , QTY

NNSP (S# , PQTY)

S#	PQTY
s1	{p1 100, p2 90, p3 50}
s2	{p1 60, p2 90}
s3	p1 150

یک تاپل

تبدیل به رابطه نرمال

SP (S# , P# , QTY)

S#	P#	QTY
s1	p1	100
s1	p2	90
s1	p3	50
s2	p1	60
s2	p2	90
s3	p1	150

یک تاپل

- دلیل نرمال بودن رابطه در RM:
  - سادگی در
    - ۱- نمایش ظاهری رابطه
    - ۲- دستورات DSL
    - ۳- اجرای دستورات

DSL → DDL, DML, DCL

مثال برای درک موارد ۲ و ۳

- درج کن
  - I_1 : $< s4 , p4 , 40 >$: در هر دو رابطه NNSP و SP منجر می شود به درج «تاپل در رابطه» با همان دستور ساده «درج کن تاپل را».
  - I_2 : $< s2 , p3 , 30 >$: با همان دستور ساده درج می شود در SP و نه NNSP .

- ادامه مثال

I_1 : **INSERT INTO** $\begin{Bmatrix} \text{NNSP} \\ \text{SP} \end{Bmatrix}$

TUPLE (S4 , P4 , 40);

I_2 : **INSERT INTO** SP

TUPLE (S2 , P3 , 30);

→ امکان پذیر

I_2 : **INSERT INTO** NNSP

TUPLE (S2 , P3 , 30);

→ امکان ناپذیر

- دلیل: تاپلی با کلید S2 وجود دارد.

کجای؟ برای درج I_2 در NNSP منطقا چه باید کرد؟

✓ در رابطه غیرنرمال دستورات ساده‌ی تاپلی کار نمی‌کنند.

مزایا و معایب رابطه نرمال و غیرنرمال

نوع رابطه	مزایا	معایب
نرمال	سادگی (۱- ... ۲-... ۳-...)	طولانی شدن کلید
	تقارن صفات (پیاده‌سازی در سطح فایلینگ ساده‌تر) (نقش تمام صفات در عبارت WHERE وقتی که شرط جستجو را با theta می‌دهیم، یکسان است، زیرا همه تک‌مقداری‌اند. SELECT…. FROM …. WHERE A<(=)(>) ‘Single Value’ چنین تقارنی در رابطه غیرنرمال وجود ندارد.)	افزونگی (ادراکی یا منطقی) (این نوع افزونگی که در مرحله طراحی پیدا شده ممکن است منجر به افزونگی فیزیکی بشود یا نشود؛ بستگی دارد به نحوه پیاده‌سازی رابطه در سطح فایلینگ. اگر تناظر یک به یک باشد، که هر تاپل هم با یک رکورد پیاده‌سازی شود، افزونگی فیزیکی نیز پیش می‌آید.)
		سنگین و زمانگیر کردن کار طراحی منطقی پایگاه داده‌ها
		کاهش سرعت بازیابی در بعضی از پرسش‌ها
		دشواری در نمایش طبیعی ارتباط سلسله مراتبی بین اشیاء و وراثت
غیرنرمال	[عکس معایب رابطه نرمال]	پیچیدگی (۱- ... ۲-... ۳-...)
		عدم تقارن صفات

مزایا و معایب رابطه نرمال و غیرنرمال (ادامه)

- **در عمل با کلید طولانی چه باید کرد؟**

- از یک کلید ساختگی استفاده می‌کنیم؛ یعنی یا خودمان به صورت دستی و یا خود سیستم به صورت خودکار به هر سطر یک شماره می‌دهد.

کجای؟ این تکنیک چه مزایا و چه معایبی دارد؟

کلید در مدل رابطه‌ای

- اصطلاح **کلید**، یک اصطلاح عام است و گونه‌هایی دارد:

۱- سوپرکلید (اَبَر کلید): SK

۲- کلید کاندید (کلید نامزد): CK

۳- کلید اصلی: PK

۴- کلید بدیل: AK

۵- کلید خارجی: FK

کلید در مدل رابطه‌ای – سوپرکلید

- رابطه $\underbrace{R(A_1, A_2, \ldots, A_m)}_{H_R}$ را در نظر می‌گیریم.
- **سوپرکلید (Super Key)**

تعریف: هر زیر مجموعه $S \subseteq H_R$ که یکتایی مقدار داشته باشد.

- اگر t_i و t_j دو تاپل دلخواه و متمایز از R باشند و $t_i(S) \neq t_j(S)$، آنگاه S یک سوپرکلید است.
- اگر N تعداد SKهای رابطه R باشد، $N \geq 1$ است، زیرا در بدترین حالت خود H سوپرکلید می‌شود.

چون بدنه، مجموعه است و تاپل تکراری نداریم.

$$1 \leq N \leq 2^m - 1$$

- **کاربرد سوپرکلید:**
  - در عمل، فاقد کاربرد مستقیم، در تئوری در بحث طراحی.
  - در SQL: با UNIQUE محدودیت یکتایی مقدار را اعمال می‌کنیم.

کلید کاندید (Candidate Key)

تعریف: هر زیرمجموعه $K \subseteq H_R$ که دو ویژگی داشته باشد:

۱- یکتایی مقدار

۲- کاهش‌ناپذیری (Irreducibility) یا کمینگی (Minimality)

- $K \subseteq H_R$ کاهش‌ناپذیر است هرگاه هر زیرمجموعه محض از K، خود یکتایی مقدار نداشته باشد.
- هر زیرمجموعه از H_R به نحوی که یک صفت را از آن حذف کنیم دیگر یکتایی مقدار نداشته باشد.

مثال:

رابطه	کلید کاندید
STT	STID
COT	COID
STCOT	(STID, COID)
S	S#
P	P#
SP	(S#, P#)

کلید در مدل رابطه‌ای – کلید کاندید (ادامه)

- CKها بر اساس قواعد معنایی محیط به دست می‌آیند.

مثال

دو حالت مختلف:

- هر کارمند در بیش از یک پروژه می‌تواند شرکت داشته باشد.

شماره‌کارمند شماره‌پروژه شماره‌ملی

EMPROJ (E#, J#, ENC, …)

CK CK

- هر کارمند در حداکثر یک پروژه می‌تواند شرکت داشته باشد.

EMPROJ (E#, J#, ENC, …)

CK CK

کلید در مدل رابطه‌ای – کلید کاندید (ادامه)

- **خصوصیات کلید کاندید:**
  - هر CK، SK هم هست ولی عکس این مطلب صادق نیست.
  - هر رابطه حداقل یک CK دارد، زیرا در بدترین حالت، خود H_R می‌شود CK.
  - رابطه می‌تواند بیش از یک CK داشته باشد.
- رابطه R حداکثر چند CK دارد؟
  - بیشترین تعداد CK زمانی است که به اندازه نصف تعداد صفات رابطه در CK شرکت کنند.
  - CKهای رابطه می‌توانند همپوشا باشند، یعنی حداقل در یک صفت مشترک باشند.
  - بنابراین اگر رابطه از درجه m باشد، بیشترین تعداد CK: $C_n^m = \frac{m!}{n!(m-n)!}$ به نحوی که $n = \left\lceil \frac{m}{2} \right\rceil$.

کلید در مدل رابطه‌ای – کلید کاندید (ادامه)

- **نقش کلید کاندید:** تضمین‌کننده عملیات تاپلی (و نه مجموعه‌ای) یا امکان ارجاع به تک تاپل در رابطه را فراهم می‌نماید.
- هر **زبر**مجموعه از CK، یک SK است (تفاوتشان در این است که CK با کمترین تعداد صفات یکتایی مقدار را می‌دهد).
- CK(های) رابطه باید به سیستم معرفی شوند.

مثال

```
CREATE RELATEION EMPROJ
        (E#   …  NOT NULL,
         J# …  NOT NULL,
         ENC …  NOT NULL)

CANDIDATE KEY  (E#, J#)
CANDIDATE KEY (J#, ENC)
```

- تئوری این را می‌گوید ولی در عمل، پکیج‌ها نمی‌پذیرند.

کلید در مدل رابطه‌ای – کلید اصلی

- کلید اصلی (Primary Key)

تعریف: کلید اصلی (PK) یکی از CKها است به انتخاب طراح.

- در عمل با عبارت PRIMARY KEY تعریف می‌شود.

- ضوابط انتخاب کلید اصلی:

۱- شناسه رایج در محیط باشد.

۲- مقادیرش همیشه معلوم باشد (نه هر CK، آنکه به عنوان PK انتخاب می‌شود)

۳- کوتاه‌تر بودن طول

۴- حتی‌الامکان مقادیرش تغییر نکند.

کلید در مدل رابطه‌ای – کلید اصلی (ادامه)

- **دلایل لزوم انتخاب کلید اصلی:**

۱- دلیل تاریخی: PK مفهوم آشناتر برای طراحان است.

۲- ایجاد شاخص اتوماتیک روی PK.

۳- در بحث جامعیت DB: چون محدودیت هیچ‌مقدارناپذیری را اگر به همه CKها بدهیم خیلی محدود کننده است. کلید CKای که این محدودیت را روی آن اعمال می‌کنند می‌شود PK.

- اصالت مفهومی در مدل رابطه‌ای با کلید کاندید (CK) است.

کلید در مدل رابطه‌ای – کلید بدیل

- کلید بدیل (Alternate Key)

تعریف: به هر کلید کاندید (CK) غیر از کلید اصلی (PK)، کلید بدیل (AK) گویند.

- در عمل متناظر ندارد.
- اگر N تعداد AKهای رابطه R باشد، داریم N >= 0.

ممکن است فقط یک CK داشته باشیم که آن هم می‌شود PK و دیگر AK نداریم.

کلید در مدل رابطه‌ای – کلید خارجی

کلید خارجی (Foreign Key)

- *در عمل:* $T_2.C$ در T_2، کلید خارجی است هرگاه در T_1، کلید اصلی باشد.
- *در تئوری:* صفت (ساده یا مرکب) $R_2.A_i$ در R_2 کلید خارجی است، هرگاه در R_1، نه لزوماً متمایز از R_2، کلید کاندید (CK) باشد.
- صفت (صفات) کلید خارجی باید هم‌میدان با صفت (صفات) کلید کاندید باشد و معمولا هم‌نام با کلید کاندید است، ولی گاه لازم می‌شود که نام دیگری داشته باشد.

مثال

رابطه	کلید خارجی	دلیل: CK در
STCOT	STID	STT
STCOT	COID	COT
SPJ	S#	S
SPJ	P#	P
SPJ	J#	J

کلید در مدل رابطه‌ای – کلید خارجی (ادامه)

- اگر N تعداد FKهای رابطه R باشد، داریم $N \geq 0$.
- معرفی کلید خارجی با عبارت FOREIGN KEY انجام می‌شود.
- **نقش کلید خارجی:** برای نمایش ارتباطهای صریح بین نوع موجودیت‌ها (و در نتیجه بین نمونه‌های آنها) به کار می‌رود. منظور از ارتباط صریح، ارتباطی است که در مدل ER با لوزی مشخص شده است.

مثال

S (S#, …) — CK

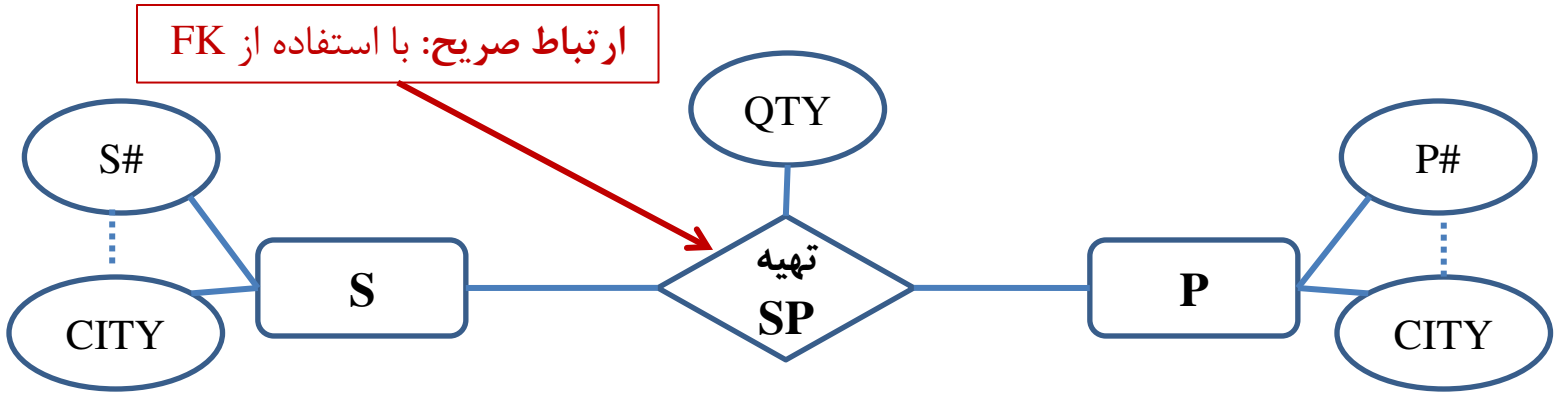
P (P#, …) — CK

SP (S#, P#, …) — FK: S#، FK: P#، CK: (S#, P#)

SCOT (STID, COID, …) — FK: STID، FK: COID، CK: (STID, COID)

کلید در مدل رابطه‌ای – کلید خارجی (ادامه)

- آیا FK تنها امکان نمایش ارتباط است یا امکان دیگری هم وجود دارد؟
  - FK تنها امکان نیست.
  - وجود هر صفت مشترک [هم دامنه و در عمل، هم‌نام (نه لزوماً)]، در عنوان مثلاً دو رابطه، نمایشگر نوعی ارتباط است بین دو نوع موجودیت که با آن دو رابطه نمایش داده‌ایم.



S (S#, … , CITY)

P (P#, … , CITY)

SP (S#, P#, …)

ارجاع

ارتباط ضمنی: از طریق هر صفت مشترک؛ صفت هم‌معنا (از یک میدان) و نه لزوماً هم‌نام

بحث تکمیلی: کلید خارجی – گراف ارجاع

مفهوم گراف ارجاع

- FK امکانی است برای ارجاع از یک رابطه به رابطه‌ای دیگر
- هر مقدار معلوم FK، امکانی است برای ارجاع مقداری از تاپل(هایی) از رابطه(هایی) به تاپلی از رابطه(هایی).

تعریف **گراف ارجاع** امکانی است برای نمایش ارجاعات بین رابطه‌ها که در آن هر گره، نمایانگر یک رابطه و هر یالِ جهت‌دار، نمایانگر ارجاع از یک رابطه (حاوی کلید خارجی) به رابطه دیگر (حاوی کلید کاندید) است.

مثال

$$P \longleftarrow SP \longrightarrow S$$

- **شکل کلی مسیر ارجاع:**

$$R_m \rightarrow R_{m-1} \rightarrow \cdots \rightarrow R_2 \rightarrow R_1$$

با این ارجاع می شود **چرخه ارجاع**

- مسیر ارجاع می‌تواند **چرخه‌ای** باشد.

گراف ارجاع (ادامه)

- چرخه ارجاع می‌تواند تک‌رابطه‌ای باشد و این در صورتی است که یک رابطه خود ارجاع (Self-Referencing) داشته باشیم.
- هنگامی که FK تعریف می‌کنیم باید معنایش را نیز بگوییم.

مثال

چرخه ارجاع بین دو رابطه کارمند و اداره.

شماره کارمند مدیر اداره

DEPT (D#, DTITLE, … , E#)

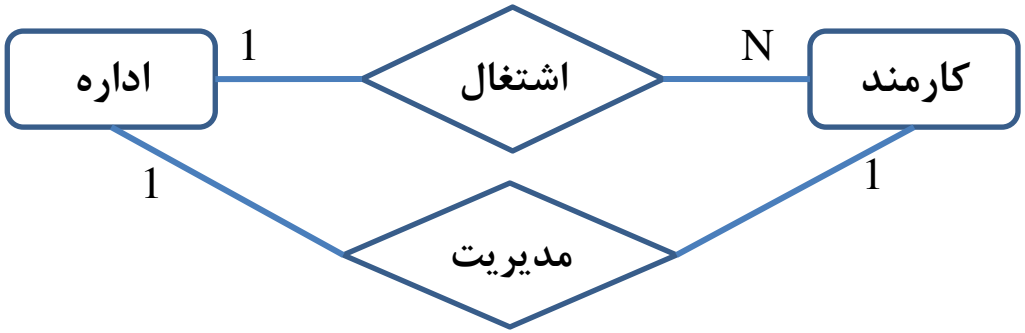
Unique

شماره اداره محل کار

EMPL (E#, ENAME, …, D#)

DEPT ⇄ **EMPL**

- بر اساس کدام مدلسازی این طراحی انجام شده است؟



گراف ارجاع (ادامه)

مثال: چرخه ارجاع تک‌رابطه‌ای کارمند با خودش.

شماره مدیر

EMPL (E#, ENAME, ENC, …, EPHONE, EMANAGER#)

EMPL

- **نکته‌های مثال اخیر:**
  - مثالی است از حالتی که در آن R1 و R2 در تعریف FK، لزوماً متمایز نیستند.
  - رابطه EMPL به خود رجوع کننده (خود ارجاع) است.
  - اگر m درجه EMPL باشد و n تعداد دامنه‌هایش باشد، داریم: $n \leq m-1$
  - لزوم دگر نامی شماره کارمندی مدیر، چون عنوان رابطه (Heading)، مجموعه‌ای از نام صفات است.
  - **تمرین:** این طراحی بر اساس کدام مدلسازی انجام شده است؟

گراف ارجاع (ادامه)

مثال: چرخه ارجاع سه رابطه‌ای

دانشکده استاد

PROF (PRID, PRNAME, …, DEID)

DEPT(DEID, DTITLE, …., UNID)

UNIV(UNID, UNAME, …, UNPRESNUM)

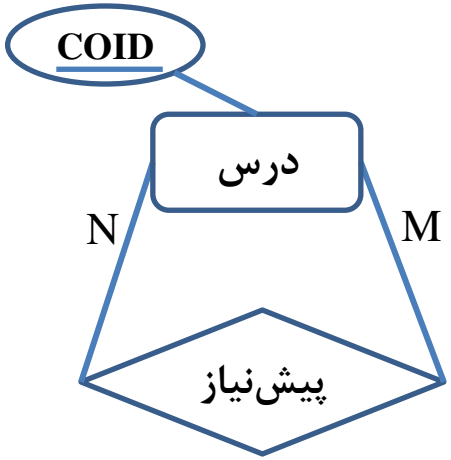
شماره استادی رئیس دانشگاه

تمرین: این طراحی بر اساس کدام مدلسازی انجام شده است؟

گراف ارجاع (ادامه)

نکته: صرف وجود دور در ERD چرخه ارجاع ایجاد نمی‌شود.

مثال



COT (COID, …)

COPRECO(COID, PRECO)

- **در چه وضعی چرخه ارجاع پدید می‌آید؟**
  - باید به چندی ارتباط‌ها توجه شود.

جامعیت پایگاه داده‌ها (DB Integrity)

تعریف: صحت، سازگاری [، دقت و اعتبار] داده‌های ذخیره شده در پایگاه داده‌ها

جنبه‌های کیفی داده (Data Quality Features)

- مسئولیت کنترل جامعیت DB با RDBMS است.
- بر اساس اطلاعاتی که کاربر [تیم طراح – پیاده‌ساز] به سیستم می‌دهد.

قواعد یا محدودیت‌های جامعیتی (Integrity Rules/Constraints)

- IRها [ICها] با استفاده از دستورات زبان پایگاهی به سیستم داده می‌شوند.

اعلانی: قواعد به نحوی اعلان می‌شوند.

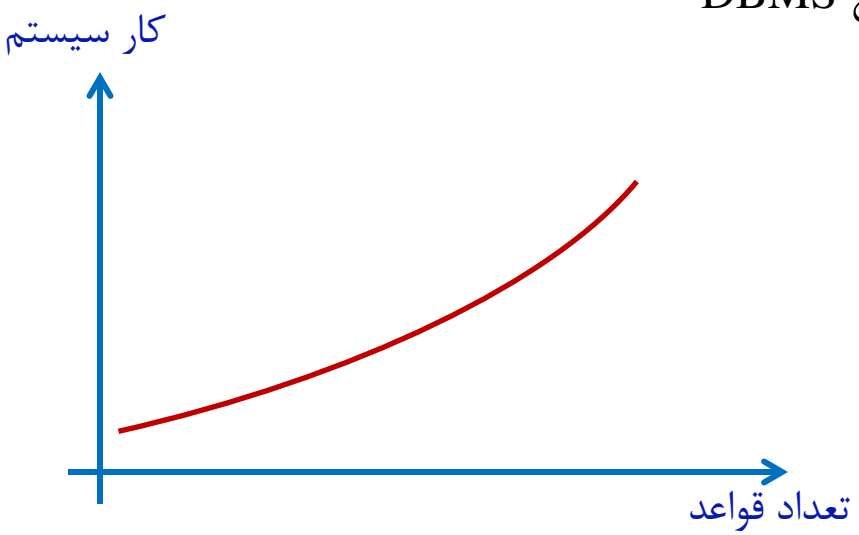
اجرایی: قواعد در یک رویه به سیستم داده می‌شوند.

جامعیت در مدل رابطه‌ای (ادامه)

- هر DBMSای باید بتواند جامعیت پایگاه داده‌ها را کنترل و تضمین کند.
- **دلیل:** زیرا همیشه ممکن است عواملی سبب نقض جامعیت شوند. از جمله:
  - اشتباه در برنامه‌های کاربردی (به ویژه اشتباهات معنایی)
  - اشتباه در وارد کردن داده‌ها
  - وجود افزونگی کنترل نشده
  - اجرای همروند تراکنشها به گونه‌ای که داده نامعتبر ایجاد شود.
  - خرابی‌های سخت‌افزاری و نرم‌افزاری

جامعیت در مدل رابطه‌ای (ادامه)

- اِعمال IRها برای سیستم سربار دارد.
- **منشأ سربار (دلایل بروز سربار) در DBMS**
  - انجام نگاشت‌ها (ناشی از معماری)
  - قواعد جامعیتی
  - اِعمال ضوابط و کنترلهای امنیت داده‌ها در سطح DBMS



قواعد جامعیت در مدل رابطه‌ای

- IRها [ICها] در مدل رابطه‌ای

۱- **قواعد [محدودیت‌های] عام:** ناوابسته به داده‌های محیط: فراقواعد (MetaRules)

۲- **قواعد [محدودیت‌های] خاص:** وابسته به داده‌های محیط: قواعد کاربری (User Defined)

یا قواعد فعالیت‌های محیط (Business Rules)

- قواعد عام در مدل رابطه‌ای
  - قاعده C1: جامعیت موجودیتی
  - قاعده C2: جامعیت ارجاعی

قواعد عام در مدل رابطه‌ای – قاعده جامعیت موجودیتی C1

- قاعده (محدودیت) C1 – قاعده جامعیت موجودیتی (Entity IR)
  - ناظر است به PK.
  - هیچ جزء تشکیل‌دهنده PK نباید هیچ‌مقدار (Null) داشته باشد.
  - دلیل:
    - ✓ PK عامل تمییز تاپل‌ها است.
    - ✓ تاپل در مدل رابطه‌ای نمایشگر نمونه موجودیت است.
    - ✓ PK عامل تمییز نمونه موجودیت‌ها است.

    عامل تمییز خود نمی‌تواند ناشناخته باشد.
  - مکانیزم اِعمال C1: اعلان PK به سیستم ← کنترل می‌کند
    - ۱- محدودیت یکتایی مقدار (با UNIQUE
    - فقط این محدودیت کنترل می‌شود)
    - ۲- محدودیت هیچ‌مقدارناپذیری

قواعد عام در مدل رابطه‌ای – قاعده جامعیت ارجاعی C2

- **قاعده (محدودیت) C2 – قاعده جامعیت ارجاعی (Referential IR)**
  - ناظر است به FK.
  - اگر $R_2.A_i$ در R_2، کلید خارجی باشد، مقدار A_i در هر تاپل از R_2 باید در R_1 مقدار قابل انطباق (Matchable Value) داشته باشد.
  - به عبارت دیگر باید هر مقدار معلوم A_i در R_2، در R_1 نیز وجود داشته باشد. یعنی در عمل می‌تواند در R_2 مقدار آن Null باشد (البته اگر جزء تشکیل‌دهنده کلید R_2 نباشد).
  - دلیل:
    - FK عامل ارجاع است؛ ارجاع به نمونه موجودیت (ارجاع مقداری و نه ارجاع از طریق اشاره‌گر).
    - در واقعیت نمی‌توان به نمونه موجودیت ناموجود ارجاع داد.

قواعد عام در مدل رابطه‌ای – قاعده جامعیت ارجاعی C2 (ادامه)

مثال

STT (STID, …)

777
888
444

STCOT (STID, COID, …)

777	CO1
…	…
444	CO4

INSERT INTO STCOT
VALUES (‘999’, ‘CO9’, …)

- چون برای 999 مقدار قابل انطباق در STT وجود ندارد، پس این درخواست رد می‌شود.

قواعد عام در مدل رابطه‌ای – قاعده جامعیت ارجاعی C2 (ادامه)

- برای اِعمال قاعده C2 در مدل رابطه‌ای لازم است:

۱- معرفی FKها به سیستم

۲- دادن گراف ارجاع

۳- مشخص کردن روش اِعمال در عملیات حذف و به‌هنگام‌سازی مقدار کلید اصلی

(در درج روش خاصی لازم نیست و در صورت عدم وجود تاپل مرجع، درخواست رد می‌شود.)

```
CREATE TABLE STCOT
        (STID     CHAR(6) NOT NULL
         COID     CHAR(6) NOT NULL
         TR       CHAR(1)
         YR       CHAR(5)
         GR       DEC(2, 2))
        CHECK (0 <= GR <= 20)
        PRIMARY KEY (STID, COID)
        FOREIGN KEY STID REFERENCES STT (STID)
                ON DELETE CASCADE
                ON UPDATE CASCADE
        FOREIGN KEY COID REFERENCES COT (COID)
                ON DELETE CASCADE
                ON UPDATE CASCADE
```

۱- معرفی FK

۲- گراف ارجاع

۳- روش اِعمال (انتشار عمل)

قواعد عام در مدل رابطه‌ای – قاعده جامعیت ارجاعی C2 (ادامه)

- روش‌های اِعمال C2 در حذف (بعضاً در به‌هنگام‌سازی):

۱- روش CASCADE: انتشاری یا تسلسلی

در این روش با حذف تاپل مرجع، تمام تاپل‌های رجوع کننده به آن حذف می‌شوند.

هر چه گراف ارجاع سنگین‌تر باشد، کار سیستم در اینجا بیشتر است.

```
DELETE FROM STT
        WHERE  STID='444'
```

↓

```
DELETE FROM STCOT
        WHERE STID='444'
```

۲- روش RESTRICTED: روش منوط به ... (یا مشروط به ...) یا روش تعویقی

در این روش اگر بخواهیم تاپل مرجع را حذف کنیم، ابتدا باید تاپل‌های ارجاع کننده به آن حذف شوند.

قواعد عام در مدل رابطه‌ای – قاعده جامعیت ارجاعی C2 (ادامه)

- روش‌های اِعمال C2 در حذف (و بعضاً در به‌هنگام‌سازی):

۳- روش SET TO NULL: روش هیچ‌مقدارگذاری یا Nullifying

در این روش باحذف تاپل مرجع، FK در تاپل‌های رجوع کننده Null می‌شود به شرط آنکه FK جزء سازنده PK نباشد.

۴- روش SET TO DEFAULT: روش درج پیش‌فرض

در این روش، با حذف تاپل مرجع، FK با مقدار پیش‌فرض جاگذاری می‌شود به شرط آنکه FK جزء سازنده PK نباشد.

قواعد عام در مدل رابطه‌ای – قاعده جامعیت ارجاعی C2 (ادامه)

- روش‌های اِعمال C2 در حذف (و بعضاً در به‌هنگام‌سازی):

۵- روش NO ACTION: عدم اقدام

برای این روش دو پیشنهاد داده شده است:

۵-۱- عدم اقدام مطلق: مثلاً مجاز نبودن عمل حذف تاپل مرجع و نمایش خطا.

۵-۲- انجام عمل خواسته شده و نه اقدام دیگر: تاپل مرجع حذف بشود ولی اقدام دیگری انجام نشود. در این مورد طراح-پیاده‌ساز می‌پذیرد که موقتاً (معمولا تا پایان یک تراکنش و نه بعد از آن) محدودیت C2 نقض شود.

- در حالت وجود چرخه ارجاع کدام روش انجام شدنی است؟
  - نمی‌توان روش RESTRICTED را در حالت کلی اِعمال کرد. با روش CASCADE هم ممکن است تاپل‌های ناخواسته حذف شود.
  - در این مواقع NO ACTION را انتخاب می‌کنیم.

قواعد خاص در مدل رابطه‌ای

- **قواعد خاص در مدل رابطه‌ای:**
  - محدودیت دامنه‌ای (میدانی)
  - محدودیت صفتی
  - محدودیت رابطه‌ای
  - محدودیت پایگاهی

قواعد خاص در مدل رابطه‌ای (ادامه)

محدودیت دامنه‌ای (میدانی)

- این محدودیت ناظر است به دامنه، مشخص‌کننده نوع و طیف مقادیر دامنه
- در همان دستور CREATE DOMAIN اعلان می‌شود.

مثال

CREATE DOMAIN GRADE DEC(2, 2) **DEFAULT** '?...?' دستور ایجاد دامنه

CONSTRAINT GRADECONST نام محدودیت (اختیاری)

CHECK VALUE BETWEEN (0, 20)

DROP DOMAIN GRADE دستور حذف دامنه

قواعد خاص در مدل رابطه‌ای (ادامه)

- **محدودیت صفتی [ستونی]**
  - این محدودیت ناشی می‌شود از محدودیت دامنه‌اش
  - صفت می‌تواند محدودیت‌های دیگری هم داشته باشد، به شرطی که ناقض محدودیت دامنه‌ای‌اش نباشد.

مثال

محدودیت‌های ناظر به صفت:

۱- صفت نمره باید بین ۰ تا ۲۰ باشد.

۲- صفت سن کاهش نمی‌یابد (محدودیت پردازشی).

محدودیت ۱، یک **محدودیت وضعیتی** است ولی محدودیت ۲، یک **محدودیت گذاری** است.

قواعد خاص در مدل رابطه‌ای (ادامه)

- محدودیت صفتی را چگونه می‌توان به سیستم اعلان کرد؟

۱- با تعریف دامنه‌اش اعلان می‌شود.

۲- در همان دستور CREATE TABLE با عبارت CHECK اعلان می‌شود.

مثال جدول انتخاب درس

```
CREATE TABLE STCOT
        (STID …
         COID …
         TR …
         GR …)
        CHECK (0 <= GR <= 20)
```

۳- با ASSERTION اعلان می‌شود.

۴- با TRIGGER به سیستم داده می‌شود (اجرایی).

قواعد خاص در مدل رابطه‌ای (ادامه)

محدودیت رابطه‌ای

- ناظر است به تاپل‌های یک رابطه (درون رابطه‌ای Intra-relational).
- حیطه اِعمالش یک رابطه است و مقادیر مجاز یک متغیر رابطه‌ای را مشخص می‌کند.
- باید در هر عملی که بر روی رابطه انجام می‌شود (که منجر به تغییر در متغیر رابطه‌ای می‌گردد) کنترل شود.

مثال: تعداد واحد درس‌های عملی قابل اخذ برای هر فرد در هر ترم، حداکثر ۲ واحد است.

مثال: تهیه‌کنندگان ساکن شهر C2 نمی‌توانند مقدار وضعیت بیش از ۱۵ داشته باشند.

قواعد خاص در مدل رابطه‌ای (ادامه)

محدودیت پایگاهی

- ناظر است به تاپل‌های بیش از یک رابطه که به نحوی با هم ارتباط معنایی [منطقی] دارند.

مثال: رابطه بین جداول STT و STCOT

یا رابطه بین جداول S و SP

مثال: دانشجوی رشته کامپیوتر نمی‌تواند درس آمار و احتمال را از گروه آموزشی D13 (دانشکده ریاضی) انتخاب کند. رابطه‌های دخیل: STT، COT و STCOT

مثال: تهیه‌کننده ساکن شهر C7 با وضعیت کمتر از ۱۵، نمی‌تواند قطعه آبی رنگ با وزن بیش از ۱۰ گرم به تعداد بیش از ۱۰۰ عدد تهیه کند.

- محدودیت‌های رابطه‌ای و پایگاهی چگونه اِعمال می‌شوند؟
  - با ASSERTION (اِعلانی)
  - با TRIGGER (اجرایی)

امکانات بیان محدودیت‌ها – اِظهار

- اِظهار – ASSERTION
  - امکانی است اِعلانی برای بیان محدودیت‌های رابطه‌ای و پایگاهی [و صفتی]

CREATE ASSERTION *name*
[BEFORE|AFTER *action*
ON *tablename* **]**
CHECK *condition(s)*

- در قسمت *condition(s)* می‌توان یک شرط ساده، یک عبارت بولی شامل چند شرط و نیز یک عبارت SELECT معتبر نوشت (همانطور که بعد از عبارت WHERE نوشته می‌شود).

- دستور حذف اِظهار

DROP ASSERTION ***name***

امکانات بیان محدودیت‌ها – اِظهار (ادامه)

مثال با این اظهار، محدودیت یکتایی مقادیر صفت کد ملی STNATID اعلان می‌شود.

```
CREATE ASSERTION  UNC-CHECK
        CHECK (UNIQUE(SELECT  STNATID  FROM  STT))
```

مثال با این اظهار این محدودیت که «جمع واحدهای انتخابی دانشجو در هر ترم–سال نباید بیش از ۲۰ واحد باشد»، اعلان می‌شود.

```
CREATE ASSERTION  TOTCRED-CHECK
        CHECK (NOT EXISTS (SELECT STID
                                FROM COT  JOIN  STCOT
                                GROUP BY  (STID, TR, YR)
                                HAVING SUM(CREDIT) > 20) )
```

امکانات بیان محدودیت‌ها – اِظهار (ادامه)

مثال

همه دانشجویان دانشکده مهندسی کامپیوتر (CE) باید درس مبانی برنامه‌سازی (با کد ۴۰۱۱۱) را اخذ کرده باشند.

```
CREATE ASSERTION   ELEM-CHECK
      CHECK (NOT EXISTS
                    ( SELECT  *  FROM  STT
                          WHERE  DEPT='CE'  AND
                                NOT EXISTS
                                ( SELECT * FROM  STCOT
                                  WHERE  STCOT.STID = STT.STID
                                        AND  STCOT.COID='40111'  ) )
```

رهانا [راه‌انداز] – TRIGGER

- امکانی است اجرایی برای اِعمال محدودیت‌های [صفتی،] رابطه‌ای و پایگاهی قبل یا بعد از بروز یک رویداد و یا به جای یک رویداد (معمولا تغییر دهنده داده‌ها).

CREATE TRIGGER *name*
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE OF *columnlist*
ON *tablename*
[REFERENCING { OLD ROW | NEW ROW | OLD TABLE | NEW TABLE} AS *name* **]**
[FOR EACH {ROW | STATEMENT}]
{(WHEN condition(s)
SQL 2003 Procedure
)}

- مفهوم نظری TRIGGER: مفهوم قاعده فعال [مفهوم محوری است در ADBMSها]

ساختار (قاعده ECA): if **E**vent on **C**ondition, then **A**ction

- Insert
- Delete
- Update

امکانات بیان محدودیت‌ها – رهانا (ادامه)

- با FOR EACH ROW بعد از بروز رویداد در هر سطر عبارت رهانا اجرا شود.
- با FOR EACH STATEMENT فقط یک بار پس از بروز رویداد (با هر تعداد سطر متاثر از آن)، عبارت رهانا اجرا شود.

مثال: این رهانا این محدودیت را که «حقوق کارمند هیچگاه کاهش نمی‌یابد» اِعمال می‌کند.

```
CREATE TRIGGER EMP-PAY-TRIG
        BEFORE UPDATE OF EMPSAL
        ON EMPL
        REFERENCING OLD AS OEMPL,  NEW AS  NEMPL
        FOR EACH ROW
        (WHEN OEMPL.EMPSAL > NEMPL.EMPSAL
                SIGNAL.SQL State '7005' ('salary cannot be decreased')
        )
```

امکانات بیان محدودیت‌ها – رهانا (ادامه)

مثال: این رهانا باعث حفظ سازگاری در جدول PROF می‌شود تا همواره صفت SALAUG حاوی آخرین میزان افزایش حقوق استاد باشد.

```
CREATE TRIGGER EMP-PAY-TRIG
        AFTER UPDATE OF  PSALARY
        ON  PROF
        REFERENCING OLD AS OPROF,  NEW AS  NPROF
        FOR EACH ROW
        (UPADATE   PROF
                SET   SALAUG=NPROF.PSALARY – OPROF.PSALARY
                WHERE   PROF.PID=OPROF.PID
        )
```

اگر بیش از یک عبارت باشد، آنها را داخل BEGIN و END قرار می‌دهیم.

امکانات بیان محدودیت‌ها – رهانا (ادامه)

مثال از کاربردهای رهانا، استفاده از آن در انجام عملیات ذخیره‌سازی از دید خارجی است (به خصوص در سمپادهایی که از عملیات در دید خارجی پشتیبانی نمی‌کنند).

STT1 (STID, NAME, MAJOR, LEVEL)
STT2 (STID, DEPT, BDATE, NATID)

```
CREATE  VIEW  CE-STT
        AS  SELECT  STID,  NAME,  MAJOR
        FROM  STT1  JOIN  STT2
        WHERE  DEPT='CE'  AND  LEVEL ='BS'

CREATE TRIGGER  INS-VIEW-TRIG
        INSTEAD  OF   INSERT   ON  CE-STT
        REFERENCING   NEW  AS NST
        FOR EACH ROW
        BEGIN
            INSERT  INTO  STT1  VALUES  ( NST.STID, NST.NAME, NST.MAJOR,  'BS')
            INSERT  INTO  STT1  VALUES  ( NST.STID, 'CE', NULL, NULL)
        END
```

امکانات بیان محدودیت‌ها – رهانا (ادامه)

مثال: این رهانا باعث اعمال قاعده C2 در عمل حذف می‌شود.

```
CREATE TRIGGER  DEL-TRIG
        BEFORE   DELETE
        ON  COT
        REFERENCING OLD AS OCOT
        FOR EACH ROW
        (DELETE   FROM   STCOT
                WHERE   STCOT.COID=OCOT.COID  )
```

- مطالعه مثالهای بیشتر از اظهار و رهانا در یادداشت‌های تکمیلی

پرسش و پاسخ . . .

amini@sharif.edu

به نام آنکه جان را فکرت آموخت

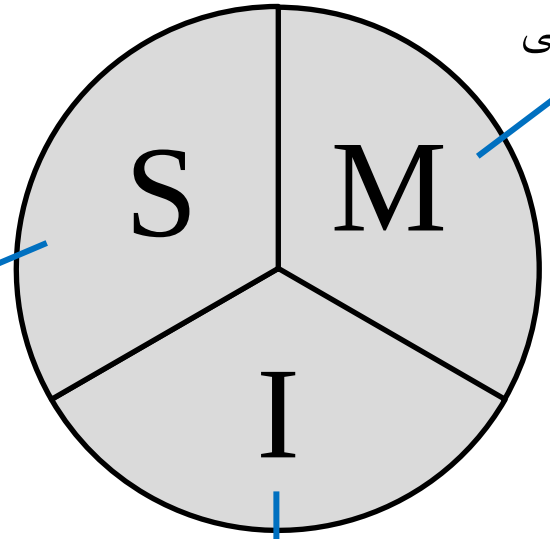
بخش هفتم:
عملیات در پایگاه داده رابطه‌ای

مرتضی امینی

نیمسال دوم ۹۴-۹۵

(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)

یادآوری: مدل داده‌ای



رابطه که انواع دارد و مفاهیم مرتبط با آن

- رابطه مبنا
- رابطه مجازی (دید)
- رابطه موقت
- رابطه لحظه‌ای (Snapshot)
- رابطه مشتق (رابطه روی رابطه‌های دیگر)

- دامنه
- تاپل
- کلید
- صفت
- ...

امکانات عملیاتی

- جبر رابطه‌ای
- حساب رابطه‌ای

امکانات کنترل جامعیت

- قواعد عام
- قواعد خاص

- عملگرها
  - عملگرهای متعارف
    - اجتماع - UNION
    - اشتراک - INTERSECT
    - تفاضل - MINUS
    - ضرب کارتزین - TIMES
    - عملگرهای دو عملوندی $R_1 \; op \; R_2$

      $op \in \{\cup, \cap, , -, \times\}$
  - عملگرهای خاص
    - گزینش یا تحدید - RESTRICT
    - پرتو یا تصویر - PROJECT
    - پیوند یا الصاق - JOIN

عملگرهای متعارف جبر رابطه‌ای

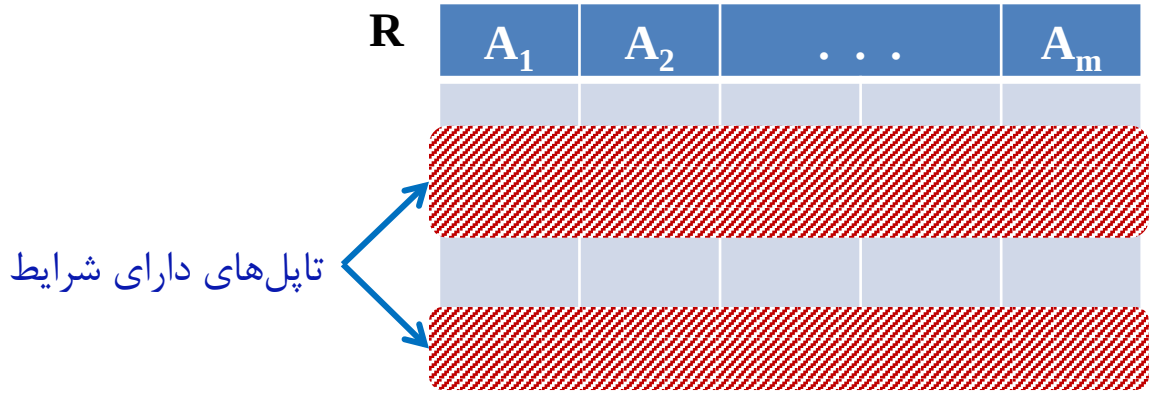
- **خاصیت بسته بودن:** حاصل ارزیابی هر عبارت جبر رابطه‌ای معتبر، باز هم یک رابطه است (که تاپل تکراری ندارد).
- برای **سه عملگر** $\cup$ ، $\cap$ **و** $-$ ، باید عملوندها نوع-سازگار (Type Compatible) باشند:
- **پیش شرط:** $H_{R_1} = H_{R_2}$
- $R_3 = R_1 \; op \; R_2 \longrightarrow H_{R_3} = H_{R_1} = H_{R_2} \qquad op \in \{\cup, \cap, , -\}$
  - بدنه نتیجه، حاصل انجام هر یک از اَعمال اجتماع، اشتراک و یا تفاضل دو مجموعه بدنه است.
- **در عملگر ضرب کارتزین (TIMES):**
  - **شرط:** در عنوان دو رابطه نباید صفت هم‌نام وجود داشته باشد. $H_{R_2} \cap H_{R_1} = \varnothing$
  - عنوان رابطه نتیجه برابر است با $H_{R_2} \cup H_{R_1}$ و بدنه نتیجه برابر ضرب کارتزین دو مجموعه بدنه است.
  - TIMES در SQL چگونه شبیه‌سازی می‌شود؟

عملگر گزینش

- **عملگر گزینش یا تحدید – RESTRICT**
- **نماد ریاضی:** σ_c ← شرط یا شرایط گزینش

یک عبارت بولی تشکیل شده از شرطهای ساده به صورت (A_i *theta* A_j) یا (A_i *theta* literal) که در آن *theta* یکی از عملگرهای =، ≠، <، >، ≤ و ≥ است و literal یک مقدار ثابت است.

- **شکل کلی:** $\sigma_c(R)$ **یا** R **WHERE** c **یا** **RESTRICT** R **WHERE** c
- تک عملوندی: Monadic
- **عملکرد** (در نمایش جدولی رابطه): زیرمجموعه‌ای افقی می‌دهد. ← عملگر تاپل(ها)یاب



عملگر گزینش (ادامه)

مثال: مشخصات کامل دانشجویان رشته فیزیک دوره کارشناسی را بدهید.

$$\sigma_{STJ='phys' \wedge STL='bs'}(STT)$$

```
SELECT STT.*
        FROM  STT
        WHERE  STJ='phys'  AND  STL='bs'
```

وقتی در شرط C (یا کلاز WHERE) بخشی از کلید را با شرط تساوی داده باشیم.

نکته: اگر $R' = \sigma_c(R)$ باشد آنگاه $CK_{R'} \subseteq CK_R$.

عملگر گزینش (ادامه)

- عملگر گزینش جابجایی‌پذیر است، یعنی:

$$\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R)) = \sigma_{c1 \wedge c2}(R)$$

- **عبارتهای جبری معادل:**

- R **WHERE** (C_1 **AND** C_2) $\equiv$ (R **WHERE** C_1) **INTERSECT** (R **WHERE** C_2)
- R **WHERE** (C_1 **OR** C_2) $\equiv$ (R **WHERE** C_1) **UNION** (R **WHERE** C_2)
- R **WHERE NOT** C $\equiv$ R **MINUS** (R **WHERE** C)

عملگر پرتو – PROJECT

- نماد ریاضی: **Π**
- شکل کلی: $\Pi_{\langle L \rangle}(R)$ یا $(R)[L]$ یا **PROJECT** R **OVER** (L)

  لیست صفات پرتو
- تک عملوندی: Monodic
- **عملکرد** (در نمایش جدولی رابطه): زیرمجموعه عمودی می‌دهد. ← عملگر ستون(ها)یاب

R

A_1	. . .	A_i	. . .	A_j	. . .	A_m

عملگر پرتو (ادامه)

- عملگر پرتو تکراری‌ها را حذف می‌کند. ← چون جواب رابطه است، پس یک مجموعه است و عضو تکراری ندارد.

مثال: شماره و رشته تمام دانشجویان را بدهید.

$$\Pi_{\langle STID,STJ\rangle}(STT)$$

SELECT STID, STJ **FROM** STT

مثال: شماره دانشجویانی که درسی انتخاب نکرده‌اند.

$$R := \Pi_{\langle STID\rangle}(STT) - \Pi_{\langle STID\rangle}(STCOT)$$

مثال: شماره و مقطع تحصیلی دانشجویان رشته IT را بدهید.

$$\Pi_{\langle STID,STL\rangle}\left(\sigma_{STJ='IT'}(STT)\right)$$

عملگر پرتو (ادامه)

- اگر $R' = \Pi_{\langle L \rangle}(R)$ باشد آنگاه:
  - اگر $CK_R \subseteq L$ آنگاه $CK_{R'} = CK_R$،
  - اگر نه در حالت کلی $CK_{R'} = L$.

کجای؟ اگر $R' = R_1 \; op \; R_2$ و $op \in \{\cup, \cap, , -, \times\}$، آنگاه $CK_{R'} = ?$.

- SELECT در SQL استاندارد، در حالت کلی ترکیبی از دو عملگر RESTRICT و PROJECT است.

عملگر پرتو گسترش یافته – EXTENDED PROJECT

- نماد ریاضی: Π
- شکل کلی: $\Pi_{\langle F1,F2,\ldots,Fn\rangle}(R)$

لیست صفات و یا توابع حسابی پرتو

- این عملگر امکان می‌دهد تا در لیست صفات پرتو، از توابع حسابی استفاده شود و صفت (صفاتی) با مقادیر حاصل از اجرای تابع (توابع) در رابطه جواب داشت.

مثال: رابطه‌ای با صفات شماره دانشجو، شماره درس و نمره دانشجو در درس، تغییریافته با فرمول G:=1.2*GRADE بدهید.

$$\Pi_{\langle STID,\ COID,\ (1.2*GRADE)\ RENAME\ AS\ G\rangle}(STCOT)$$

عملگر تغییر نام – RENAME

- نماد ریاضی: ρ
- شکل کلی: $\rho_R(E)$

نام رابطه حاصل از عبارت جبر رابطه‌ای E

- این عملگر برای نامیدن رابطه حاصل از یک عبارت جبر رابطه‌ای به کار می‌رود.
- **عملکرد:** $\rho_R(E)$ رابطه حاصل از عبارت جبر رابطه‌ای E را با نام R برمی‌گرداند.
- از عملگر RENAME برای دگرنامی صفت هم می‌توان استفاده کرد (مشابه آنچه در مثال اسلاید قبل آمد). مثلاً با دستور R **RENAME** A_i **AS** B_j، به صفت A_i از R، نام دیگر B_j داده می‌شود.

- **عملگر پیوند JOIN** (مدل ریاضی عمومی)
  - **نام عمومی:** Theta Join
  - **نماد ریاضی:** $\bowtie_{Cond(s)}$ ← شرط پیوند
  - **فرض:** دو رابطه R_1 و R_2 نام صفت مشترک ندارند.

$$\begin{cases} R_1(A_1, A_2, \ldots, A_n) \\ R_2(B_1, B_2, \ldots, B_m) \end{cases}$$

  - **شکل کلی:** $\mathbf{R_1 \bowtie_C R_2}$ یا $\mathbf{R_1\ \theta\text{-}JOIN_C\ R_2}$ یا فقط $\mathbf{R_1\ JOIN_C\ R_2}$
  - Theta
    - $=$ EQUI-JOIN
    - $\neq$ NOT EQUI-JOIN
    - $<$ LESS THAN-JOIN
    - $\leq$ LESS EQUI-JOIN
    - $>$ GREATER THAN-JOIN
    - $\geq$ GREATER EQUI-JOIN

عملگر پیوند (ادامه)

- **شرط پیوند (c):** $R_1.A_i$ **theta** $R_2.B_j$

صفات پیوند

که باید **هم‌دامنه** و **ناهم‌نام** باشند.

چون نتیجه JOIN رابطه است و در headingاش صفت تکراری نباید وجود داشته باشد.

- **نکته:** اگر صفات پیوند هم‌نام باشند، حداقل یکی را باید دگرنامی کرد (به دلیل وجود این راه حل، حساسیتی در عدم وجود صفت مشترک نداریم).

- در حالت کلی شرط پیوند می‌تواند به صورت زیر باشد که در آن c_1، ...، c_n قالب بالا (قالب شرط پیوند) را دارند.

$$<c_1> \text{ AND } <c_2> \text{ AND } \ldots \text{ AND } <c_n>$$

مثال

$$<R1.A1 = R2.B1> \text{ AND } <R1.A2 = R2.B2>$$

عملگر پیوند (ادامه)

مثال: مشخصات کامل جفت تهیه‌کننده–قطعه از یک شهر را بدهید.

$$R_1 := S \bowtie_{S.CITY=P.PCITY} (P \text{ RENAME CITY AS PCITY})$$

S (S#, SNAME, STATUS, CITY)

S1			C1
S2			C2
S3			C3
S4			C4
S5			C5
S6			C6

P (P#, . . . , W, CITY)

P1	5	C1
P2	6	C2
P3	4	C1
P4	7	C4
P5	10	C5

R_1 (S#, . . . , CITY, P#, . . . , W, PCITY)

S1	C1	P1	5	C1
S1	C1	P3	4	C1
S2	C2	P2	6	C2
~~S3~~	تاپل پیوندشدنی ندارد.			
S4	C4	P4	7	C4
S5	C5	P5	10	C5
~~S6~~	تاپل پیوندشدنی ندارد.			

عملگر پیوند (ادامه)

- **عملکرد:**

$$R_3 = R_1 \bowtie_C R_2$$

$$H_{R_3} = H_{R_1} \cup H_{R_2}$$

  - در بدنه R_3 تاپل‌های پیوندشدنی از دو رابطه قرار دارند.

- **خصوصیات:**

  - $R_1 \bowtie_C R_2 = R_2 \bowtie_C R_1$ چون صفات در heading رابطه نظم مکانی ندارند.
  - $R_1 \bowtie_C R_2 = \sigma_C(R_1 \times R_2)$ حاصل Theta-Join در حالت عمومی، زیرمجموعه‌ای افقی از ضرب کارتزین است که در آن تاپل‌هایی از حاصلضرب که حائز شرط پیوند هستند حضور دارند.

وقتی در شرط پیوند، تساوی بخشی از کلید هر دو رابطه را داده باشیم.

نکته

اگر $R' = R_1 \bowtie_C R_2$ باشد، آنگاه $CK_{R'} \subseteq CK_{R_1} \cup CK_{R_2}$

گونه‌های خاص عملگر پیوند – پیوند طبیعی

- پیوند طبیعی (Natural Join)
  - گونه‌ای از پیوند است که دو ویژگی دارد:

۱- Theta: =

۲- صفات پیوند یک بار در جواب می‌آیند. (صفت یا صفات پیوند باید هم‌نام هم باشند.)

مثال

$R_2 := S \bowtie_{S.CITY=P.CITY} P$

R_2 (S#, . . . , CITY, P#, . . . , W)

S1	C1	P1	5
S1	C1	P3	4
S2	C2	P2	6
S4	C4	P4	7
S5	C5	P5	10

گونه‌های خاص عملگر پیوند – پیوند طبیعی (ادامه)

- اگر صفت مشترک [هم‌نام و هم‌دامنه] یک صفت باشد، نیازی به قیدکردن نیست.

  اما اگر بیش از یک صفت باشد، باید صفت یا صفات پیوند را قید کنیم.

  اگر قید نکنیم، پیوند روی تساوی مقادیر تمام صفات مشترک انجام می‌شود.

$$R_1: (A, B, C)$$

$$R_2: (A, F, C)$$

$$R' = R_1 \bowtie R_2$$

$$R': (A, B, C, F)$$

- اگر $H_{R_1} \cap H_{R_2} = \emptyset$، آنگاه $R_1 \bowtie R_2 = R_1 \times R_2$.
- اگر $H_{R_1} = H_{R_2}$، آنگاه $R_1 \bowtie R_2 = R_1 \cap R_2$.

گونه‌های خاص عملگر پیوند – نیم‌پیوند

- **نیم‌پیوند (Semijoin)**
  - در شکل عمومی با هر Theta نوشته می‌شود.
  - **نماد:** $\ltimes_C$ (در چپ تعریف شده)
  - **مدل ریاضی:** $R_3 := R_1 \ltimes_C R_2 = \Pi_{\langle H_{R_1} \rangle}(R_1 \bowtie_C R_2)$
  - **عملکرد:**
    - $H_{R_3} = H_{R_1}$
    - در بدنه R_3: تاپل‌های پیوند شدنی از رابطه چپ

گونه‌های خاص عملگر پیوند – نیم‌پیوند (ادامه)

مثال

$R_3 := S \ltimes_{S.CITY=P.PCITY} (P \text{ RENAME CITY AS PCITY})$

R_3 (S#, . . . , CITY)

S1	C1
S2	C2
S4	C4
S5	C5

کجای؟ کاربرد این عملگر چیست؟

- **تمرین:** عملگر نیم‌پیوند در SQL شبیه‌سازی شود.

گونه‌های خاص عملگر پیوند – برون‌پیوند

برون‌پیوند (Outer Join)

- Theta هر چیزی می‌تواند باشد.
- سه گونه دارد:

۱- Left O. J. $⟕_C$

۲- Right O. J. $⟖_C$

۳- Full O. J. $⟗_C$

- عملکرد $R_4 := R_1 ⟗_C R_2$:
  - $H_{R_4} = H_{R_1} \cup H_{R_2}$
  - در بدنه R_4: تاپل‌های پیوند شدنی از دو رابطه و

تاپلهای‌های پیوندناشدنی از رابطه چپ گسترش‌یافته با هیچ‌مقدار (Null Value)

گونه‌های خاص عملگر پیوند – برون‌پیوند (ادامه)

مثال

$R_4 := S ⟗ P$

R_4 (S#, . . . , CITY, P#, . . . , W)

S1	C1	P1	5
S1	C1	P3	4
S2	C2	P2	6
S4	C4	P4	7
S5	C5	P5	10
S3	C3	?	?
S6	C6	?	?

کجای؟ کلید R_4 (CK_{R4}) چیست؟ بی‌تردید کلید اصلی ندارد.

- مشکل Outer Join:

۱- از نظر ریاضی رابطه نیست، چون کلید اصلی ندارد.

۲- مصرف حافظه زیاد

کجای؟ این عملگرها در عمل چه کاربردی دارند؟

کجای؟ آیا عملگرهای Outer Join خاصیت جابجایی دارند؟

عملگر نیم‌تفریق

- نیم‌تفریق (Semi Minus)
- R_1 **SEMIMINUS** R_2 = R_1 **MINUS** (R_1 **SEMIJOIN** R_2)
- عملکرد
  - $H_{R_5} = H_{R_1}$
  - در بدنه R_5: تاپل‌های پیوند نشدنی از رابطه چپ

عملگر تقسیم (Divide)

- مفروضند رابطه‌های:

$$\begin{cases} R_1(\overbrace{A_1, A_2, \ldots, A_n}^{X}, \overbrace{B_1, B_2, \ldots, B_m}^{Y}) \\ R_2(B_1, B_2, \ldots, B_m) \end{cases}$$

- شرط عمل:
  - $R_3(X) := R_1(X, Y) \div R_2(Y) \longrightarrow H_{R_2} \subseteq H_{R_1}$

- عملکرد:

۱- $H_{R_3} = X = H_{R_1} - H_{R_2}$

۲- در بدنه R_3: بخش X از تاپلهایی از R_1 که حاوی تمام مقادیر Y از R_2 باشند.

عملگر تقسیم (ادامه)

مثال

R_1 (S#, P#) ÷ R_2(P#) = R_3(S#)

S1	P1	P1	S1
S1	P2	P2	
S1	P3	P3	
S2	P1		
S2	P2		
S3	P1		

R_1 (S#, P#) ÷ R_4(P#) = R_5(S#)

S1	P1	P1	S1
S1	P2	P2	S2
S1	P3		
S2	P1		
S2	P2		
S3	P1		

کنجکاوی؟ Outer Divide چیست؟

عملگر تقسیم (ادامه)

- ضرب و تقسیم جبر رابطه‌ای لزوماً عکس هم نیستند.
- **تمرین:** عملگر تقسیم را در SQL شبیه‌سازی کنید.
- **تمرین:** Q3 و Q4 (صفحه A-3 از یادداشت‌های تکمیلی سری II) را بدون استفاده از عملگر DIVIDE بنویسید.

عملگر گسترش – EXTEND

- صفت یا صفاتی را به عنوان (heading) یک رابطه اضافه می‌کند. حاصل، رابطه دیگری است.

EXTEND STUD **ADD** STADDRESS

STUD (STID, …, STD, STADDRESS)

- در SQL با ALTER TABLE پیاده‌سازی شده ولی ALTER ستون(هایی) را به همان جدول اضافه می‌کند.

- با این عملگر می‌توانیم یک ستون محاسبه‌شدنی به رابطه اضافه نماییم.

عملگر تلخیص – SUMMARIZE

- تاپل‌های رابطه را گروه‌بندی می‌کند به نحوی که مقدار صفت (صفات) گروه‌بندی در هر گروه یکسان باشد؛ معمولاً با یک یا چند تابع جمعی استفاده می‌شود.
- این عملگردر SQL با GROUP BY پیاده‌سازی شده است.

SUMMARIZE STCOT **BY** (STID) **ADD AVG(**GRADE**) AS** AVER

- برای این پرسش‌ها، اول عنوان (Heading) رابطه جواب را تعیین می‌کنیم.
- به جای AVG می‌توانیم از توابع جمع و یا گروهی دیگر مانند MIN (حداقل)، MAX (حداکثر)، SUM (جمع) و یا COUNT (شمارشگر تاپل‌ها) استفاده کنیم.

عملگر GROUP

- عملگر GROUPپیشنهاد Date است، برای تبدیل رابطه نرمال به غیرنرمال (در SQL، NEST است).
- عکس آن UNGROUP (در SQL، UNNEST) است.

SP **GROUP** (P#, QTY) **AS** NNPQTY

NNSP (S#, NNPQTY[P#, QTY])

	P1	50
S1	P2	70
	P3	60
S2	P1	100
	P2	150

- با استفاده از UNGROUP، رابطه نرمال SP را می‌توانیم مجددا به دست آوریم.

NNSP **UNGROUP** NNPQTY

عملیات ذخیره‌سازی با جبر رابطه‌ای

- از لحاظ تئوریک می‌توان عملیات ذخیره‌سازی را هم با عملگرهای جبر رابطه‌ای انجام داد.

عمل	عملگر
درج	$\cup$
حذف	$-$
به‌هنگام‌سازی	اول $-$ بعد $\cup$

متغیر رابطه‌ای

مقدار رابطه‌ای

$$S := S \cup \{\{\langle S\#, S7\rangle, \langle SNAME, SN7\rangle, \langle STATUS, 17\rangle, \langle CITY, C7\rangle\}\}$$

عملگر انتساب رابطه‌ای

(یک مقدار رابطه‌ای را منتسب می‌کند به یک متغیر رابطه‌ای)

یک رابطه نوع-سازگار با S و کاردینالیتی یک

مقایسه دو رابطه

- دو رابطه R_1 و R_2 مقایسه‌شدنی (قابل قیاس) هستند، هر گاه نوع-سازگار باشند ($H_{R_2} = H_{R_1}$)
- در مقایسه رابطه R_1 با R_2 ، بدنه R_1 با بدنه R_2 مقایسه می‌شود از نظر هم مجموعگی، زیرمجموعگی و زبرمجموعگی

$$\Pi_{\langle STID \rangle}(STT) * \Pi_{\langle STID \rangle}(SCR)$$

$$* \in \{\subset, \supset, \subseteq, \supseteq, =, \neq\}$$

- پاسخ عمل مقایسه: یا T یا F. به طور مثال در رابطه فوق:
  - اگر $\supset$ باشد، پاسخ T است اگر حداقل یک دانشجو باشد که درسی انتخاب نکرده باشد.
  - اگر $\subset$ باشد، پاسخ T است اگر حداقل در یک عمل ذخیره‌سازی در این DB قاعده جامعیت C2 رعایت نشده باشد (حذف از دانشجو و یا درج در انتخاب درس).

کامل بودگی جبر رابطه‌ای

- جبر رابطه‌ای **زبانی** است از نظر رابطه‌ای **کامل** (Relational Completeness) یعنی هر رابطه معتبر متصور از مجموعه رابطه‌های ممکن را می‌توان به کمک یک عبارت جبر رابطه‌ای بیان کرد.
- جبر رابطه‌ای ضابطه تشخیصِ کامل بودن زبان‌های رابطه‌ای است.
  - اگر هر رابطه‌ای را که با جبر رابطه‌ای می‌توان نشان داد، با زبانی مدعی کامل بودن رابطه‌ای بتوان نشان داد، آن زبان از نظر رابطه‌ای **کامل** است.
- **کاربردهای جبر رابطه‌ای:**
  - عملیات بازیابی
  - عملیات ذخیره‌سازی
  - تعریف انواع رابطه‌های مشتق (رابطه مجازی، لحظه‌ای و ...) مثال: تعریف دید (View) در SQL
  - ....

مباحث تکمیلی در جبر رابطه‌ای

- برای نوشتن یک پرسش (Query):

  ۱- از چه رابطه‌هایی استفاده کنیم.

  ۲- از چه عملگرهایی استفاده کنیم (حتی‌الامکان با کمترین تعداد عملگر)

  ۳- چه ترتیبی از عملگرها استفاده کنیم.

- مثال‌هایی از کاربرد جبر رابطه‌ای را در عملیات در RDB (در یادداشت‌های تکمیلی سری II) (صفحه A-1 و A-2) مطالعه نمایید.

- روش‌های اجرای عملگر Join در DBMS کدامند؟

- **حساب رابطه‌ای** شاخه‌ای است از منطق ریاضی، منطق مسندات.
- حساب رابطه‌ای و جبر رابطه‌ای معادلند. یعنی هر رابطه‌ای را که بتوان با یک عبارت جبر رابطه‌ای نوشت، می‌توان با عبارتی از حساب رابطه‌ای هم نوشت و برعکس.
- حساب رابطه‌ای حالت **توصیفی** دارد ولی جبر رابطه‌ای حالت **دستوری** دارد.

Descriptive

به کمک عبارات منطقی، شرایط ناظر به رابطه را برای سیستم توصیف می‌کنیم.

Prospective

دستورات عملیاتی به سیستم می‌دهیم.

- حساب رابطه‌ای هم ضابطه تشخیص زبان‌های رابطه‌ای کامل است.

- **متغیر تاپلی (Tuple Variable) یا متغیر طیفی (Range Variable):**

  - متغیری است که مقادیر آن تاپل‌های یک رابطه است (هر لحظه یک تاپل).

RANGVAR SX **RANGES OVER** S;

RANGVAR PX **RANGES OVER** P;

RANGVAR SPX **RANGES OVER** SP;

RANGVAR C2X **RANGES OVER** (S **WHERE** CITY='C2');

طیف مقادیرش تاپل‌هایی از S است که شرط را داشته باشند.

سورها (Quantifiers)

- سور وجودی EXISTS X (F): حداقل یک مقدار برای متغیر X وجود دارد به نحوی که به ازای آن، فرمول F به درست ارزیابی شود.
- سور همگانی (عمومی) FOR ALL X (F): به ازای تمام مقادیر متغیر X، فرمول F به درست ارزیابی می‌شود.

مثال: با فرض اینکه X از مجموعه اعداد صحیح مثبت مقدار می‌گیرد.

EXISTS X (X<10) حاصل ارزیابی: TRUE

FOR ALL X (X<10) حاصل ارزیابی: FALSE

حساب رابطه‌ای – سورها (ادامه)

- **یادآوری:** بین این دو سور روابط زیر وجود دارد.

FOR ALL X (F) = NOT EXISTS X (NOT F)

EXISTS X (F) = NOT (FORALL X (NOT F))

FORALL X (F) $\Rightarrow$ EXISTS X (F)

NOT EXISTS X (F) $\Rightarrow$ NOT FORALL X (F)

- بر اساس روابط فوق می‌توان روابط پیچیده دیگری را نیز استنباط کرد مانند روابط هم ارزی زیر:

FORALL X (F AND G) = NOT EXISTS X (NOT(F) OR NOT(G))

FORALL X (F OR G) = NOT EXISTS X (NOT(F) AND NOT(G))

EXISTS X (F OR G) = NOT FORALL X (NOT(F) AND NOT(G))

EXISTS X (F AND G) = NOT FORALL X (NOT(F) OR NOT(G))

حساب رابطه‌ای – فرمول خوش‌ساخت

تعریف: یک **فرمول خوش ساخت (WFF)** به صورت زیر تعریف می‌شود:

- اگر R یک رابطه و T یک متغیر تاپلی تعریف شده روی R باشد، آنگاه R(T) یک فرمول اتمی است.

  [R(T) یعنی، T یک عنصر (تاپلی) از R است.]

- اگر T_i یک متغیر تاپلی روی رابطه R و A یک صفت از R باشد و T_j یک متغیر تاپلی بر روی S و B یک صفت از S باشد، آنگاه T_i.A theta T_j.B یک فرمول اتمی است (theta یک از عملگرهای متعارف مقایسه‌ای است).

- T_i.A theta C و C theta T_j.B نیز که در آن C یک مقدار ثابت است، فرمول اتمی هستند.

- اگر F_1 و F_2 فرمول باشند، آنگاه (F_1 AND F_2)، (F_1 OR F_2)، NOT(F_1) نیز فرمول هستند.

- اگر F یک فرمول و T یک متغیر تاپلی باشد، آنگاه EXISTS T(F) و FORALL T(F) نیز فرمول هستند.

حساب رابطه‌ای – عبارت حساب رابطه‌ای

تعریف: اگر X یک متغیرتاپلی روی رابطه $R(A_1, A_2, \ldots, A_n)$ باشد در اینصورت شکل کلی **عبارت حساب رابطه‌ای** بدین صورت است:

(target-items) [**WHERE** F]

که در آن target-items فهرستی از صفات متغیر تاپلی X به صورت $X.A_1, X.A_2, \ldots, X.A_n$ و F یک فرمول خوش‌ساخت است.

مثال:

- ST.STID شماره تمام دانشجویان در رابطه STT
- ST.STID **WHERE** ST.STDEID='D11' شماره دانشجویان گروه آموزشی D11
- (ST.STID, ST.STL) **WHERE EXISTS** STCO (ST.STID=STCO.STID **AND** STCO.COID='COM11')

شماره دانشجویی و مقطع تحصیلی آنهایی که درس COM11 را انتخاب کرده‌اند.

حساب رابطه‌ای – عبارت حساب رابطه‌ای (ادامه)

مثال

- SX.S# شماره همه تهیه کنندگان

- SX.SNAME **WHERE** SX.CITY='C2' **AND** SX.STATUS> 15

نام تهیه کنندگان شهرستان C2 که وضعیت آنها بزرگتر از 15 باشد.

- مثال‌های بیشتر در کتاب‌های مرجع و یادداشتهای تکمیلی سری II.

پرسش و پاسخ . . .

amini@sharif.edu