پایگاه داده ها

دکتر روحانی رانکوهی

جلسه اول شنبه 1388/04/20 ساعت ۷:۳۰

مقدمه :

هر سیستم نرم افزاری از مجموعه ای از داده های ذخیره شده استفاده می کند.

سیستم های نرم افزاری در یک تقسیم‌بندی کلی به چهار دسته تقسیم می شوند:

- ✓ بنیادی: OS
- ✓ نیمه بنیادی: DMS , DBMS
- ✓ کاربردی: Applications
- ✓ ابزاری: tools

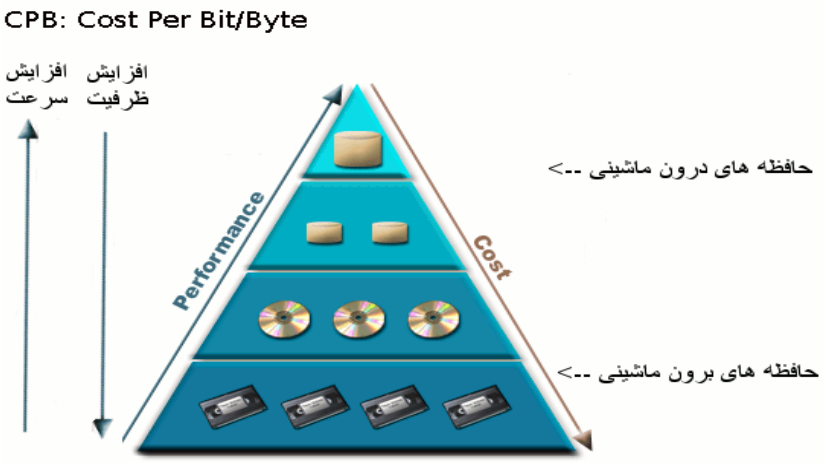
داده های ذخیره شده در یک تقسیم‌بندی به سه دسته تقسیم می شوند:

- ✓ ساختمند structured : داده‌هایی که فرمت از پیش تعریف شده دارند.
- ✓ نیم ساختند semi structured
- ✓ نا ساختند un structured

داده های ذخیره شده(دسته بندی دیگر):

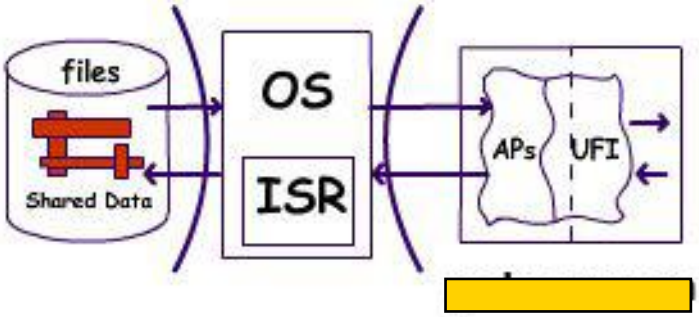
- ✓ ذخیره شده در تعدادی فایل
- ✓ ذخیره شده در یک سلسله مراتب حافظه ها Storage Hierarchy

یادآوری : رسانه های ذخیره سازی (حافظه های ذخیره سازی)



داده‌های ذخیره شده در تعدادی فایل

به مجموعه ای از فایل های فیزیکی (که روی device ذخیره شده اند) اصطلاحاً محیط فیزیکی «ذ–ب – ا = ذخیره و بازیابی اطلاعات» می گوییم (ISR=Information Storage and Retrieval) . این محیط باید ایجاد – مدیریت– بهره برداری شود .



انواع کاربر:

- برنامه ساز

AP : Application Programs
UFI : User Friendly Interface

- نا برنامه ساز

سیستم واسط ISR

نسل ها:

- مشی فایلینگ (Filing Approach)

۱. نسل سیستم های فایلینگ FS

۲. نسل سیستم مدیریت داده‌ها DMS

- مشی پایگاهی

۳. نسل سیستم مدیریت پایگاه داده‌ها DBMS

تأکید : برای ایجاد سیستم های کاربردی در اساس دو مشی داریم :

- مشی فایلینگ
- مشی پایگاهی ← ابزار ما DBMS است.

نسل DBMS ها که از 1965 آغاز شده است تقسیم بندی درون نسلی دارد:

- HDBMS : Hierarchy
- NDBMS : Network
- RDBMS : Relational
- OODBMS : Object Oriented
- ORDBMS : Object Relational

ضابطه این دسته بندی مفهوم بسیار مهم مدل داده ای (Data Model یا طرز نمایش داده ها) است.

بیش از 20 رده بندی DBMS داریم :

- از نظر تعداد کاربر هایی که می پذیرند
  - ✓ یک کاربری
  - ✓ چند کاربری
- از نظر OS ها
  - ✓ Portable
  - ✓ Non Portable
- از نظر زبان ها

تعریف پایگاه داده ها (*DB*):

مجموعه ای است از داده های ذخیره شده، پایا (مانا[1])، به هم مرتبط[2]، مجتمع[3]، حتی الامکان فاقد افزونگی[4] دارای معماری خاص خود، مبتنی بر یک مدل داده ای تحت مدیریت یک سیستم کنترل متمرکز، مورد استفاده ی یک یا چند کاربر ، از یک محیط(سازمان) به طور اشتراکی و همزمان .

1 persistent
2 Inter Connected
3 integrated
4 Redundancy

داده های ذخیره شده در *DB* موسومند به داده های عملیاتی ← داده هایی که کاربران روزانه با آن سر و کار دارند .

- داده‌های ذخیره شده در DB بر دو نوعند :

داده‌های کاربری ← مورد استفاده کاربر

داده های سیستمی ← خود سیستم ایجاد می کند Meta Data

- داده های ذخیره شده در DB لزوما همان داده های ورودی /خروجی نیستند . یعنی :

همه داده‌هایی که سیستم نگهداری می کند آن داده هایی نیست که data entryمی شود.

داده هایی که سیستم در خروجی به کاربر می دهد هم لزوما stored نیستند .

۳۹:۰۰

مثال : فرض S جدول تهیه کنندگان، P جدول قطعات، SP جدول تهیه می کند/تهیه می شود باشند:

Query ← شماره هر قطعه و کل تعداد تهیه شده از آن را بدهید .

S :

S#	Sname
S1	…
S2	…
S3	…

P :

P#	Pname
P1	…
P2	…
P3	…

SP :

#S	#P	QTY
S1	p1	100
S1	p2	70
S1	p3	80
S2	p1	60
S2	p2	90
S3	p1	70

جدول جواب(آنچه که سیستم در خروجی به کاربر می‌دهد که لزوماً stored نیست.)

#P: شماره قطعه	تعداد کل
p1	230
p2	160
p3	80

QUERY مورد نیاز برای رسیدن به پاسخ فوق را بنویسید:

نکته : همانطور که می‌بینید آنچه که سیستم در خروجی به user می‌دهد لزوماً stored نیست و می‌تواند یکی از انواع داده‌های زیر باشد:

Calculated Attribute
Virtual Attribute
Derived Attribute

پاسخ :

```
select P# as PN , SUM (QTY) as SQ
        from SP
        group by P#
```

نکته : داده‌های ذخیره شده در DB را پایدار یا persistent گویند. اما صفتهایی چون SQ که از محاسبه ستون‌های دیگر به دست می‌آید را صفت محاسبه شده، مجازی یا مشتق می گویند.

تعریف : داده پایا داده ای است که پس از اجرای برنامه ی کاربر (مثل یک SQL Session) کماکان در سیستم باقی می ماند. وجودش درسیستم با اتمام اجرای برنامه کاربر از بین نمی رود . به بیان دیگر مثل داده های محیط برنامه نویسی نیست. (متغیر های محیط برنامه نویسی را ناپایا گویندچون با پایان یافتن برنامه از بین می روند) این بدان معنا نیست که اساساً تغییر نمی کنند. بلکه پیرو درخواست کاربر مجاز می تواند تغییر کنند.

داده های ذخیره شده در سیستم پایگاهی براساس یک ساختار داده‌ای مشخص (DS) به هم مرتبط اند. مثلاً بر اساس یک DS درختی.

ارتباط ساختاری بین داده‌ها بر دو نوعند:

- منطقی : ارتباط منطقی بین داده ها را به این وسیله نمایش می دهیم :

  HDS → سلسله مراتبی

  NDS → شبکه ای

  RDS → رابطه ای

- فیزیکی : ساختار فایل های فیزیکی(بیش از ۴۰ نوع ساختار فایل وجود دارد)

نکته : اساساً DS برای [نمایش داده ها / ارتباط بین آن ها] درسطح طراحی منطقی DB می باشد.

مجتمع بودن (یکپارچگی) به معنی این است که مدلسازی ، طراحی منطقی و توصیف داده‌ها(Schema) واحد است، اما کاربران متعدد هر یک دید (view) خاص خود را نسبت به داده‌ها دارند .

مجتمع بودن دو وجه دارد:

- به طور منطقی ← به معنای دیده شده
- به طور فیزیکی ← ذخیره شده در یک سایت

وقتی می‌گویند داده‌ها در سیستم پایگاهی مجتمع هستند به این معنی است که داده‌ها حداقل به طور منطقی مجتمع باشند (در یک DB باشند) اما از نظر فیزیکی می‌توانند مجتمع باشند یا نباشند.

۱:۰٤:۰۰

افزونگی

افزونگی در معنای عام به معنی تکرار ذخیره سازی داده‌ها می باشد.

افزونگی درون فایلی یا Intra File Redundancy به معنی تکرار ذخیره سازی داده ها در حد یک فیلد یا بیشتر درخود نمونه‌ای از فایل‌های کمکی فایل شاخص[5] است که برای سرعت دست یابی به رکورد ها می باشد.

افزونگی درون فایلی خود بر دو نوع است :

- طبیعی یا natural : ناشی می شود از ماهیت دادهای محیط

  مثال: رشته دانشجو، که صد ها دانشجو رشته یکسان دارند.

- تکنیکی یا Technical : به دلیل استفاده از یک تکنیک ایجاد می شود مثال Indexing (نمایه گزاری)

[5] Index File

کنجکاوی : برای کاهش مصرف حافظه در حالت طبیعی چه تکنیک هایی وجود دارد ؟

راهنمایی : در اساس دو دسته تکنیک داریم:

۱. تکنیک های فشرده سازی داده‌ها ← 10 یا 15 روش.

۲. تکنیک ها ی مبتنی بر ساختار فایل ← ساختار فایل را این گونه طراحی می کنیم.

تأکید:

معایب افزونگی:

۱. مصرف حافظه

۲. خطر بروز overhead (فزونکاری) درعملیات ذخیره سازی : فزون کاری به دلیل حفظ سازگاری داده‌ها

مزیت افزونگی:

۱. افزایش سرعت در بازیابی

۲. برای ترمیم داده ها ، ایمنی بیشتر داده ها ، عملیات کنترلی سیستم ، ...

افزونگی در مباحث *Data Base* (افزونگی درمعنای گسترده)

تکرار ذخیره سازی مقادیر یک یا چند صفت از یک یا چند نوع موجودیت[6] از یک محیط یا سازمان (در چند سیستم معمولاً مجزا)

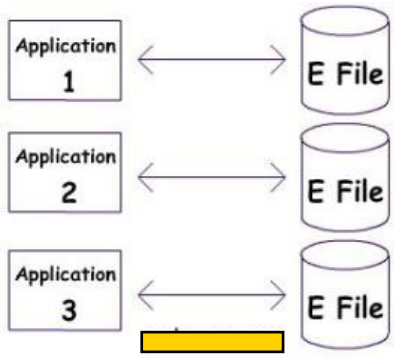
مثلاً: مشخصات نوع موجودیت کارمند در یک سازمان درچند سیستم ذخیره شده باشد.

- پدیده ای که در مشی فایلینگ بسیار رایج است و نیز در گونه ای از مشی پایگاهی.

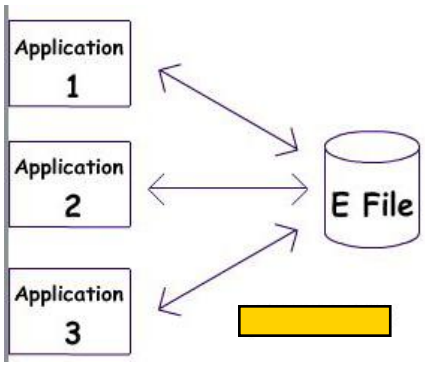
در شکل زیر در یک سازمان چندین سیستم وجود دارد مثل سیستم حقوق و دستمزد ، سیستم کارگزینی و ... که در این سیستم ها برخی اطلاعات مثل مشخصات سجلی در تمامی سیستم ها تکرار شده است.

رایج در مشی ناپایگاهی و گونه ای از مشی پایگاهی. DBMS ها جدا از هم و Non Integrated هستند.

6 Entity Type



شکل زیر نمونه‌ای از سیستم پایگاهی است:



کدام نوع افزونگی در مشی پایگاهی به حداقل می رسد ؟

افزونگی ناشی از مشی طراحی سیستم کاربردی که در اساس دو گونه است:

- مشی فایلینگ
- مشی پایگاهی

کنجکاوی : دلایل دیگر بروز افزونگی کدام است ؟

جواب : رجوع شود به مشی مربوطه

← افزونگی ناشی از طراحی بد (Bad Design) . مثل طراحی جدول ها با فرم نرمال ضعیف

← افزونگی های عملیاتی (لازم برای عملیات سیستم) . مثل TLF[7] و Back Up

[7] Transaction Log File

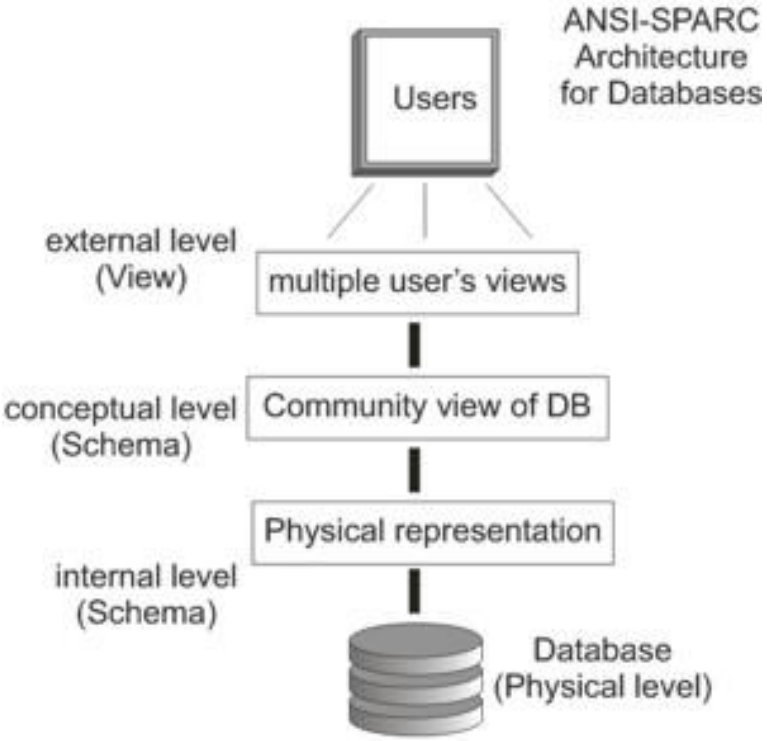
ابتدای صدای اول ابتدای صدای ۲

معماری سیستم پایگاهی

معماری سیستم پایگاهی موسوم است به معماری سه سطحی (پیشنهادی ANSI)

بحث اولیه

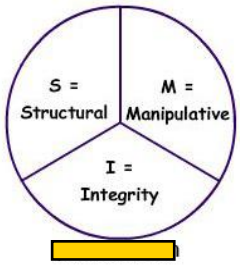
هر مجموعه از فایل ها DB نیست ، DB معماری خاص خودش موسوم به معماری پیشنهادی ANSI را دارد.



مفهوم مدل داده ای Data Model مجموعه ای است از امکانات :

- نمایش داده ها و ارتباطات بین آن ها (درمرحله طراحی منطقی) DS
- عملیات در داده ها (پردازش داده ها)
- کنترل داده‌ها (امکانات جانبی)

نمایش مفهوم مدل داده ای:



Manipulative : عملیاتی که به کمک آن ها در DB عملیات انجام می دهیم.

Integrity : مجموعه امکاناتی که به کمک آنها داده هایمان را کنترل می کنیم.

مثال : در TDM[8] (در تئوری : RDM)

- بخش S ← مفهوم جدول TDS
- بخش M ← مثلاً دستور های SQL
- بخش I ← قواعد C1[9] و C2[10] و قواعد جامعیتی

نکته تکمیلی : DS بخشی است از یک مفهوم گسترده تر به نام Data Model

دلایل لزوم DS :

۱. طراحی منطقی DB

۲. طراحی زبان پایگاهی (DB Language) یا DBL ← هر زبانی حول محوری طراحی می شود.

مثال SQL خاص DS های جدولی است.

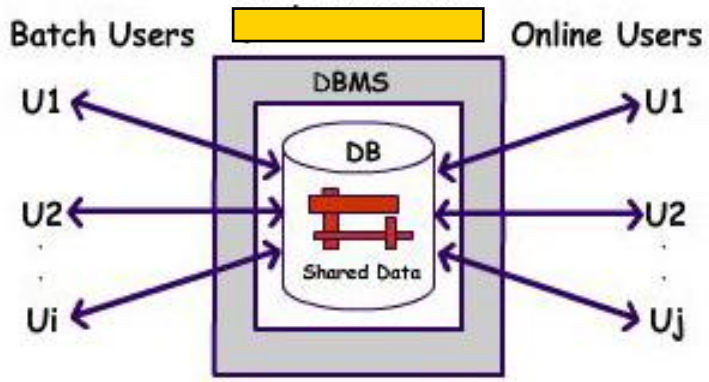
۳. طراحی خود DBMS

٤. لازم برای مقایسه DBMS ها

8 مدل داده ای جدولی
9 کلید اصلی تهی نباشد
10 کلید خارجی تهی نباشد

سیستم کنترل متمرکز

۱. تیم DBA
۲. DBMS



نکاتی که در شکل بالا می بینیم:

- ✓ نمایش مفهوم کنترل متمرکز توسط DBMS
- ✓ داده های ذخیره شده اشتراکی
- ✓ تعدد کاربران (هرکدام طبق دید و نیاز خود)
- ✓ همزمانی از دید کاربر
- ✓ هرنوع کاربر (batch و online) را می پذیرد. و نوع دیگر کاربری که می‌توانیم داشته باشیم کاربر Inter Active یا تعاملی است.

نکته : کاربر Inter Active حتماً online است اما هر کاربر online حتماً Inter Active نیست.

– با توجه به آن چه گفته شد عناصر اصلی سیستم پایگاهی کدامند؟

۱. سخت‌افزار

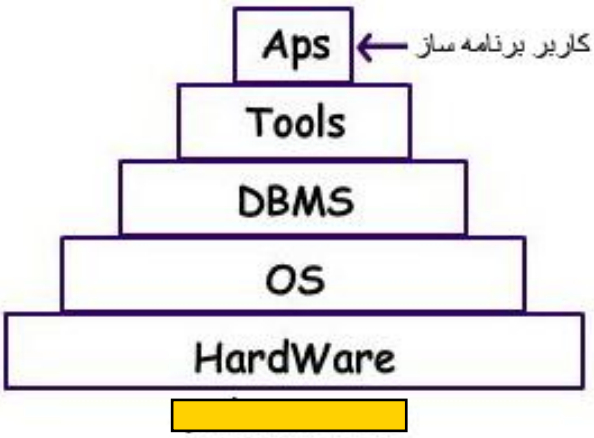
۲. نرم افزار

- OS
- DBMS
- APs
- تسهیلات نرم افزاری (لازم برای ایجاد سیستم کاربردی) مثل Tools و System Facilities
- نرم افزار شبکه (نه لزوماً همیشه) .

۳. داده

٤. کاربر (گونه هایی دارد، رجوع شود به پیوست)

نکته : اگر package های برنامه سازی نباشند برنامه نویسی مشکل تر، حجم برنامه بیشتر و ایجاد برنامه سازی کاربردی سخت تر می شود .



سخت افزار محیط پایگاهی:

۱. سخت افزار ذخیره سازی

- رسانه اصلی: دیسک (ظرفیت بالا – سرعت بالا)
- رسانه فرعی : نوار مغناطیسی (جهت انجام کار هایی از قبیل back up)
- همچنین تکنولوژی MMDB یا Main Memory Data Base Technology هم داریم که هدف آن این است که I/O Time را حتی الامکان به صفر برساند.

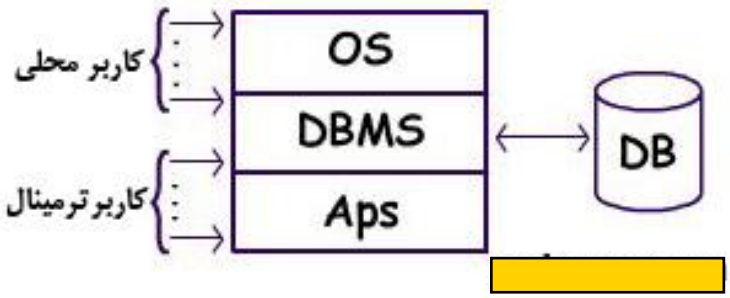
۲. سخت افزار پردازشگر

- کامپیوتر های معمولی از هر رده (Main , Mini , PC , ...)
- زیرساخت سخت افزاری مناسب باید موجود باشد.
- همچنین DBM هم داریم (Data Base Machine)

۳. سخت افزار همرسانی یا ارتباط (communication)

- امکانات محلی یا local: برای اتصال دستگاه های جانبی به کامپیوتر....
- امکانات شبکه ای : برای ایجاد سیستم پایگاهی توزیع شده DDB (Distributed DataBase Systems)

نکته تکمیلی: معماری یک سیستم پایگاهی



توجه: با بحث تحت عنوان معماری DB پیشنهادی ANSI اشتباه نشود

بحث اجمالی : ایده های اصلی

۱. سیستم های پایگاهی با معماری متمرکز Centralize Data Base Sys Arch

نمایش :

۱. یک پیکربندی H/S یا Hardware/Software
۲. یک DBMS
۳. m>=1 کاربر یا APs
٤. یک DB مجتمع به صورت فیزیکی

۲. سیستم های پایگاهی با معماری نامتمرکز

گونه ها:

۱. سیستم پایگاهی با معماری مشتری خدمتگزار C/S DB SYS Arch
۲. سیستم پایگاهی توزیع شده DDB
۳. سیستم پایگاهی چند پایگاهی MDB
٤. سیستم پایگاهی موازی PDB
٥. سیستم پایگاهی موبایل Mo-DB

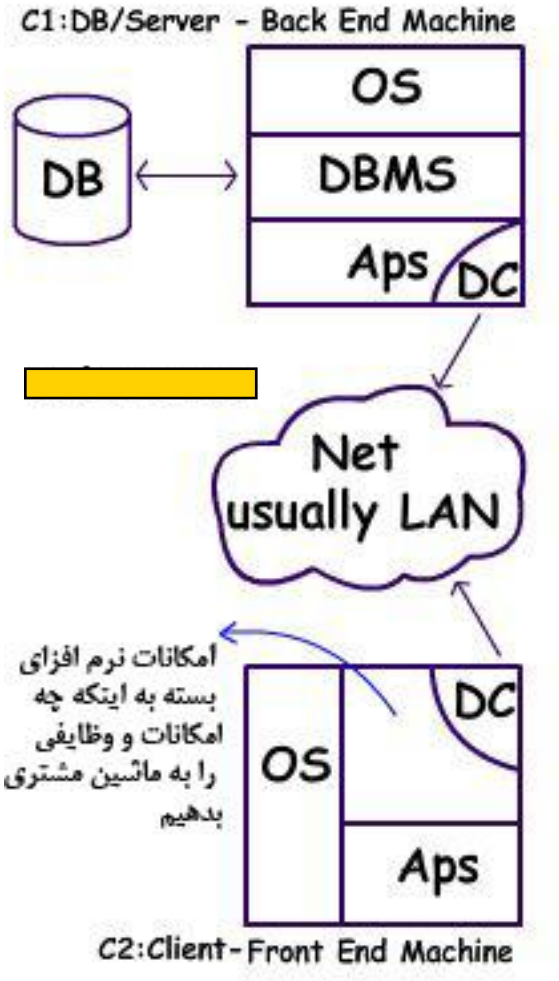
سیستم پایگاهی C/S

دلیل اصلی : تقسیم وظایف سیستم و در نتیجه افزایش همروندی عملیات و افزایش سرعت

هرکدام از سیستم‌های زیر راه حلی است برای یک محیط مشخص با نیازهای مشخص

۱. سیستم پایگاهی تک مشتری / تک خدمتگزار C/S DB Single SYS
۲. سیستم پایگاهی چند مشتری / تک خدمتگزار MC/S DB SYS
۳. سیستم پایگاهی تک مشتری / چند خدمتگزار C/MS DB SYS
٤. سیستم پایگاهی چند مشتری / چند خدمتگزار MC/MS DB SYS

نمایش سیستم پایگاهی تک مشتری / تک خدمتگزار:

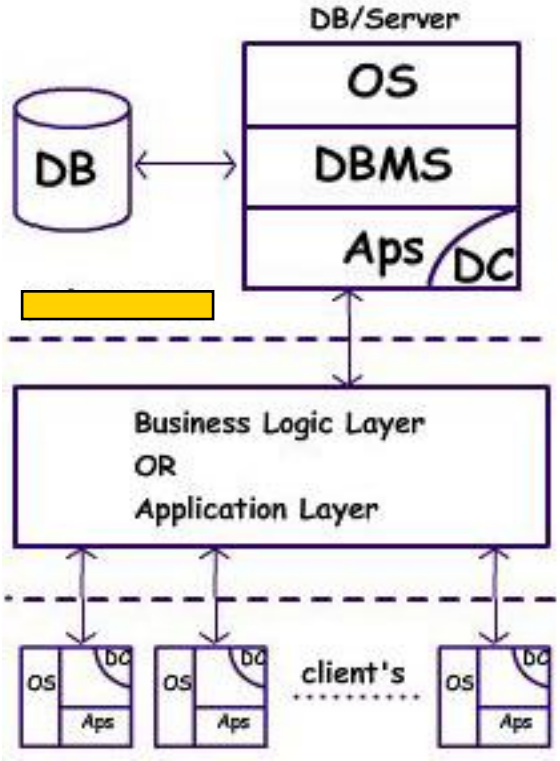


لازمه وجود یک Data Base ، وجود حداقل یک DBMS در سیستم است.

معماری نمایش داده شده در شکل قبل را معماری two-tier (دولایه یا دوردیفی) می گویند اما معماری سه ردیفی و چهار ردیفی هم داریم.

نمایش معماری three-tier

در این معماری ماشین های مشتری ماشین‌هایی هستند بسیار ساده ، ارزان ، حتی ممکن است فاقد هارد باشند ، با کمترین امکانات نرم افزاری برای data entry ، نمایش خروجی ها و ...



در شکل بالا :

لایه بالایی : ردیف یا لایه خدمتگزار پایگاهی

لایه میانی : لایه برنامه کاربردی / لایه منطق فعالیت های محیط

لایه پایینی : لایه نمایش یا کاربرد / presentation layer – User Interface

تمرین : مزایای این معماری نسبت به دو لایه چیست ؟

۱. اطمینان عملیاتی بالا از جانب کنترل بهتر روی برنامه ها
۲. اعمال تغییرات در برنامه ها آسانتر است
۳. امنیت داده ها بیشتر
٤. صرفه جویی در هزینه سخت افزار (به ویژه اگر تعداد مشتری ها زیاد باشد. معمولاً ۵۰-٦۰ مشتری به بالا مناسب است)
۵. آیا تنگنا های شبکه ای نسبت به دو لایه کاهش پیدا می کنند؟ (Network Buttleneck)

نکته : معمولا برای بیش از 50 مشتری معماری 3 لایه توصیه می شود.

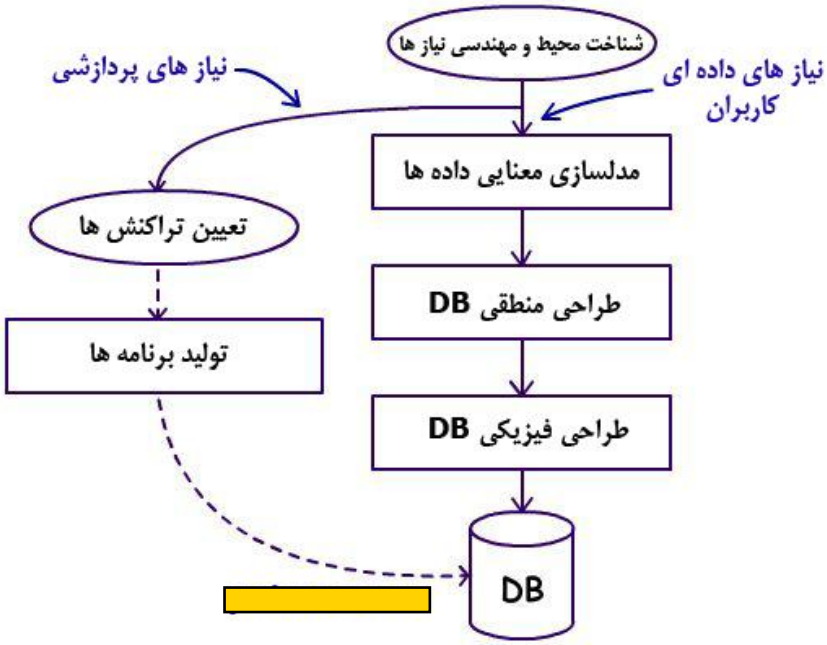
تمرین : همین معماری سه لایه را می خواهیم در web داشته باشیم browser کجا قرار می گیرد؟

روی client ها : لایه ی بین client و Application Server قرار می گیرد.

استراحت دوم ۱:۱۰:۳۰ تا ۱:۱۹:۳۰

مدلسازی معنایی داده‌ها – *Semantic Data Modeling*

مقدمه : مراحل کلی ایجاد سیستم پایگاهی:



هر AP از تعدادی transaction تشکیل شده است.

تاکید: AP از m>=1 تراکنش تشکیل شده است.

یادآوری : تراکنش چیست ؟

- قطعه برنامه ای که معمولا یک عمل تغییر دهنده در DB انجام می دهد . (در حالت خاص فقط خواندن یا readonly trans .)
- یا به تمامی اجرا می شود و یا اجرا نشده تلقی می شود.
- همچنین خواصی دارد موسوم به خواص اسید[11]

[11] ACID : atomicity consistency isolation durability

مثال: A,B در حساب بانکی:

```
BEGIN TRANS
      READ A;
      A := A-50;
      write A;
      READ B;
      B := B+50;
      WRITE B;
END TRANS
```

شرط سازگاری داده‌ها در مثال بالا این است که جمع A+B ثابت باشد.

در صورتی که به هر دلیلی در میانه کار failed شود ، شرط سازگاری برقرار نیست ، سیستم در مرحله restart اعمال زیر را انجام می دهد:

۱. اثر اجرای ناقص این تراکنش را undo می کند.

۲. اگر نوبت اجرای برنامه باشد دوباره اجرا می کند.

این را خاصیت atomicity(خاصیت اتمیک یا تجزیه ناپذیر بودن) تراکنش می‌گویند. یعنی یا همه یا هیچ

تمرین : برنامه بالا را در SQL بنویسید .

سئوال – مدلسازی معنایی داده‌های به چه معنی است؟

پاسخ : ارائه یک مدل کلی از پایگاهی که می خواهیم ایجاد کنیم با استفاده از مفاهیم انتزاعی و با توجه به معنایی که کاربر برای داده ها قائل است. (در محیط مورد نظر)

درک ← برداشت ← مدلسازی

مفاهیم انتزاعی یعنی مفاهیمی مطرح در سطحی فراتر از سطح نمایش منطقی و طبعا سطح پیاده سازی

گاهی در برخی منابع برای مدلسازی معنایی اصطلاحات زیر به کار برده می شود:

۱. مرحله پیش طراحی

۲. طراحی conceptual (ادراکی یا مفهومی) ← بهتر است به مدلسازی معنایی طراحی conceptual گفته نشود.

برای مدلسازی نیاز به یک روش داریم، روش رایج تر در بحث DB روش ER[12] است.

روش های دیگر:

۱. OMT → Object Modeling Technics
۲. UML → Unified Modeling Language

- UML خاص مدلسازی معنایی داده‌ها نیست ← می توان با UML داده ها را مدلسازی کرد.
- UML برای مدلسازی- طراحی سیستم های نرم افزاری است.
- UML نمودارهای متعدد دارد ازجمله نمودار رده[13] که شبیه نمودار ER است.

انواع ER :

۱. ER مبنایی ← فقط با همین سه مفهوم زیر باید داده‌های محیط مدل شوند:

الف – نوع موجودیت

ب – صفت / خصیصه / ویژگی

ج – نوع ارتباط

۲. ER گسترش یافته (تقویت شده)

- مدلسازی قبل از هر چیز درست دیدن محیط است

[12] Entity Relation ship
[13] Class Diagram

نوع موجودیت: مفهوم کلی آن چه می خواهیم در باره اش اطلاع داشته باشیم (از یک محیط مشخص)

مثال : محیط ← کارخانه :

بعضی نوع موجودیت های مطرح :

۱. قطعه
۲. پروژه
۳. تهیه کننده
٤. کارمند
٥. انبار
٦. ...

قدم اول مدلسازی معنایی تشخیص این مجموعه است.

محیط:

1. Micro Real World خرد جهان واقع
2. Mini World
3. Universe of Discourse جهان مطرح

نکات:

۱. هر نوع موجودیت [یک نام دارد / یک معنا دارد]
۲. هر نوع موجودیت نمونه هایی دارد .m>=1 و m تعداد نمونه ها

نوع موجودیت تک نمونه ای از نظر تئوریک متصور است اما در عمل شاید لازم نباشد آن مفهوم را به عنوان نوع موجودیت در مدل سازی دخالت دهیم.

هر نوع موجودیت صفت یا صفاتی دارد .

نوع موجودیت تک صفتی متصور است اما در عمل باید دقت کرد ، شاید لازم آن نوع موجودیت را به صورت موجودیت دخالت دهیم.

۳. هر نوع موجودیت ارتباط هایی دارد با نوع های دیگر (در مواقعی ممکن است با خودش ارتباط داشته باشد)

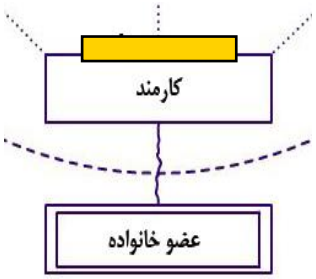
نوع موجودیت بدون ارتباط (Isolated) متصور است ولی در عمل معمولاً مطرح نیست.

٤. نوع موجودیت : (دقت کنید که مفهوم قوی و ضعیف مفهومی است نسبی و نه مطلق)

- قوی strong : مستقلا در محیط مطرح است.
- ضعیف weak : مستقلا مطرح نیست. وابستگی وجودی (Existence Dependency) دارد. به نوع های دیگر به گونه ای که اگر آن نوع دیگر مطرح نباشد نوع ضعیف هم مطرح نخواهد بود .

مثال :

Entity Type → Single



عضو خانواده، نوع موجودیت ضعیف کارمند است چرا که به اعتبار مطرح شدن کارمند مطرح می شود.

اگر قوی مطرح نباشد ضعیف دیگر مطرح نیست .

جلسه دوم دوشنبه 1388/04/22 ساعت ۷:۳۰

مدلسازی معنای داده‌ها (ادامه بحث)

یادآوری : در روش ER سه مفهوم زیر وجود دارد:

- نوع موجودیت
- صفت (خصیصه یا ویژگی)
- نوع ارتباط

هر نوع موجودیت مجموعه ای از صفات دارد که فضای اطلاعاتی ناظر به نمونه های آن نوع را تشکیل می دهد.

مثال : نوع موجودیت : کارمند

مجموعه صفات : شماره کارمندی . نام . نام خانوادگی . کد ملی . سال تولد

نکات :

- هر صفت یک نام دارد. نامی که برای کاربر در محیط مطرح است.
- هر صفت یک معنا دارد.
- هر صفت یک دامنه (domain یا میدان) دارد. از طریق دامنه نوع، طیف مقادیر و معنای صفت مشخص می شود.

به بیان دیگر صفت از دامنه اش [نوع – مقدار – معنا] می گیرد و نه لزوماً نام را.

هر صفت محدودیت (هایی) دارد : (محدودیت را قید یا constraint)

- محدودیت دامنه ای (نوع و مقدار) Domain Constraint
- محدودیت یکتایی مقدار uniqueness
- محدودیت هیچ مقدار ناپذیری (Null Value نداشته باشد – صفتی که همیشه باید مقادیرش معلوم باشند)
- محدودیت های عملیاتی (مثلا نمی‌توان آدرس را با هم جمع کرد)
- محدودیت های بین صفات (وابستگی های بین صفات)

۱. وابستگی تابعی

۲. وابستگی شمول: صفت B با صفت A وابستگی شمول دارد (محدودیت) هر گاه مجموعه مقادیر B زیرمجموعه یا خود مجموعه مقادیر A باشد. مثلاً کد ملی زنان زیر مجموعه کد ملی افراد کشور است.

مثال دیگر وابستگی Fk به Pk

- محدودیت های دیگر (محدودیت های خاص هر محیط) مثل اینکه نباید موجودی کالا در انبار از یک مقداری بیشتر باشد

رده بندی صفت:

۱. صفت ممکن است شناسه(ID) باشد یا ناشناسه

ویژگی‌های صفت شناسه:

- صفتی است که محدودیت یکتایی مقداردارد
- محدودیت « هیچ مقدار » ناپذیری دارد

شناسه ویژگیهای بالا را حتماً دارد و بهتر است ویژگی‌های زیر را نیز داشته باشد:

- تغییر ناپذیری مقدار (چون شناسه قرار است کلید شود و تغییر آن مشکل زاست)
- کوتاهی طول نمایش مقدار (ثابت شده که هر چه طول کلید کوتاهتر باشد کارایی سیستم از نظر حافظه و از نظر زمان عملیات بیشتر می شود)

نکته – اگر n تعداد شناسه های نوع موجودیت E باشد داریم n>=۱ یعنی موجودیت می‌تواند بیش از یک شناسه داشته باشد.

مثال : کارمند : کد کارگزینی شناسه است کد ملی هم می تواند شناسه باشد

۲. صفت ممکن است هیچ مقدارپذیر باشد یا هیچ مقدارناپذیر

۳. صفت ممکن است ساده باشد یا مرکب

- صفت ساده یا single صفتی است تک معنایی (قابل تجزیه به لحاظ معنایی نیست)
- صفت مرکب یا composite تشکیل شده از چند صفت ساده، اجزای تشکیل دهنده آن خود صفتی از همان محیط هستند (معنا دارد).

نکته : ساده یا مرکب بودن صفت نسبی است . مطلق نیست . بستگی دارد به آن حیطه ی معنایی . بستگی به کاربر (application) دارد.

مثالی برای صفت مرکب :

- آدرس : ترکیبی از نام خیابان ، کوچه و . . .
- تاریخ : ترکیبی از رور و ماه و سال
- شماره دانشجویی که ترکیبی از سال ورود، کد رشته و . . . است. این روش مرکب در معنا، ساده درنمایش است.

کد دادن به پدیده ها می تواند یک روش صرفه جویی درحافظه باشد.

کنجکاوی: آیا یک چنین صفاتی را بهتر نیست به صورت صفاتی جدا بگیریم که هر یک استقلال وجودی خود را داشته باشند ؟

(هر یک از قسمت های معنایی را جداگانه در یک فیلد بگذاریم)

دراین نوع سئوال ها باید trade off کنیم.

٤. صفت ممکن است تک مقداری یا چند مقداری باشد : (تک مقداری بودن نوعی محدودیت است)

- تک مقداری یا single value صفتی است که به ازای هر نمونه از نوع موجودیت حداکثر یک مقدارمی گیرد. مثل شماره ملی ، تاریخ تولد و . . .
- چند تعدادی multiple value صفتی است که برای حداقل یک نمونه از نوع موجودیت بیش از یک مقدار می گیرد. مثل شماره تلفن

٥. صفت ممکن است واقعی یا مجازی باشد.

- صفت واقعی (real یا صفت ذخیره شده) : صفتی که مقادیر ذخیره شده در db دارد .

  مثل شماره کارمند، رنگ قطعه
- صفت مجازی (مشق یا محاسبه شده) : صفتی که مقادیر ذخیره شده در db ندارد. مقدارش با پردازشی محاسبه یا تولید می شود و در اختیار کاربر قرار داده می شود . مثل میانگین قد افراد ، جمع مقادیر یک ستون

تست : کدام مفهوم مترادف به سه مفهوم دیگر نیست ؟

- صفت مشق
- صفت عمده *
- صفات مجازی
- صفت محاسبه شده

تعریف : صفت عمده prime attribute صفت جزء کلید وقتی که کلید مرکب باشد. در نرمال تر سازی از آن استفاده می شود.

مثال : صفت مجازی:

SUM (quantity) AS SQ

وقتی که صفتی ماهیتاً محاسباتی باشد به این معنا نیست که در طراحی برای آن ستون در نظر نگیریم . مثلاً اگر query با فرکانس اجرایی بالا داشته باشیم بهتر است جایی آن را store کنیم تا هر دفعه recomputation انجام ندهیم. این کار به دلیل سرعت در بازیابی صورت می گیرد.

۵۷:۵۰

نوع ارتباط:

عمل [فعل-کنش] بین n>=1 نوع موجودیت رخ می دهد.

اگر n=1 باشد ارتباط با خود یا Self Relation ship خواهد بود.

مثال – تهیه کننده و قطعه : تهیه کننده قطعه را تهیه می کند. نوع ارتباط ← تولید کردن

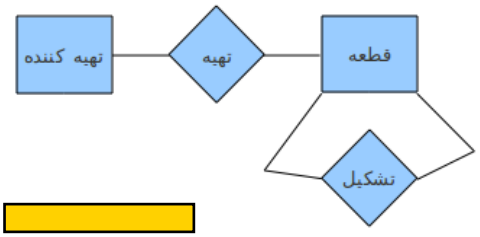
- تشخیص درست نوع ارتباط ها از مراحل اساسی مدلسازی است.

برای نمایش از نماد های رسم ERD استفاده می‌کنیم. ER Diagram

مثال – مدل قطعه مرکب است. یک قطعه از قطعاتی تشکیل می شود:

Entity ← مستطیل

ارتباط ← لوزی



مثال ارتباط با خود :

در شکل بالا قطعه از قطعاتی تشکیل می‌شود : ارتباط با خود

پیش نیازی درس‌ها نیز ارتباط با خود است.

نکات :

- هر نوع ارتباط یک نام دارد : معمولاً دروجه مصدری بیان می شود مثل تهیه کردن، مدیریت کردن
- هرنوع ارتباط یک معنا دارد.
- هرنوع ارتباط شرکت کنندگانی دارد (همان موجودیت های دخیل)

دریک ارتباط : به تعداد نوع موجودیت های شرکت کننده در یک ارتباط درجه ارتباط می گویند. درجه را با n نشان می دهیم.

- اگر n=۱ باشد : ارتباط یگانی یا Unary Relation Ship
- اگر n=۲ باشد : ارتباط باینری یا دودویی یا دوگانی
- اگر n=۳ باشد : ارتباط سه گانی
- اگر درجه ارتباط n باشد به آن ارتباط nگانی یا n-ary می گویند. به همین خاطر به درجه arity نیز می گویند.

نکته : درجه ارتباط معمولاً ۱، ۲ یا ۳ می‌باشد ، به ندرت ٤ می‌باشد و تقریباً هیچ‌گاه بیش از ٤ نمی باشد.

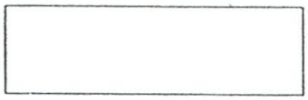
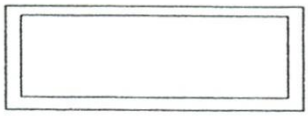
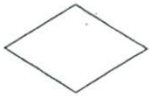
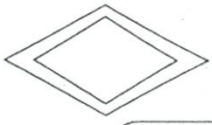
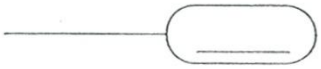
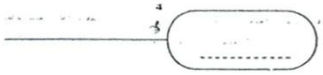
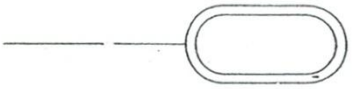
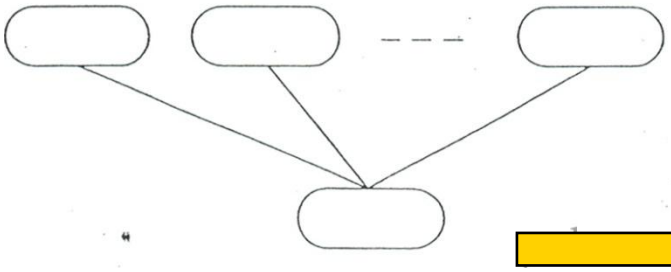
نکته : شرکت نوع موجودیت E درنوع ارتباط R : ممکن است الزامی باشد یا غیر الزامی باشد.

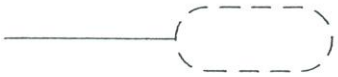
مشارکت وقتی الزامی یا کامل است که تمام نمونه‌های E در R شرکت داشته باشند. درغیر این صورت مشارکت را غیر الزامی ، ناقص یا partial گوییم.

نکته : الزامی بودن مشارکت خود نوعی محدودیت معنایی است ناظر به ارتباط و خواهیم دید در طرز طراحی منطقی تأثیر دارد .

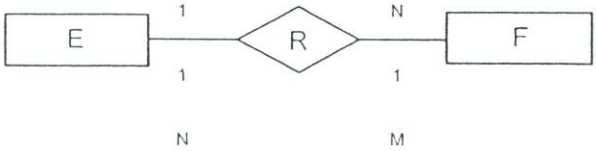
راهنمای شمایل :

نمادهای رسم نمودار در (E)ER : این نمادها رایج‌ترند؛ نمادهای دیگری هم وجود دارد.

معنا	نماد
نوع موجودیت	
نوع موجودیت ضعیف(وابسته)	
نوع ارتباط	
نوع ارتباط با موجودیت ضعیف (وابسته)	
صفت	
صفت : شناسه‌ی اول موجودیت	
صفت : شناسه‌ی دوم موجودیت(در صورت وجود)	
صفت ممیزه	
صفت : چند مقداری	
صفت : مرکب	

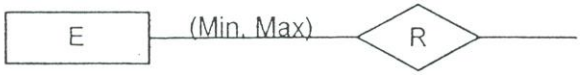


صفت : مشتق (مجازی- محاسبه شدنی)

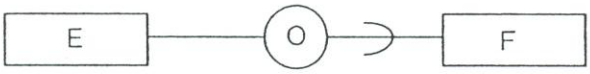


چندی نوع ارتباط R و نوع مشارکت در آن

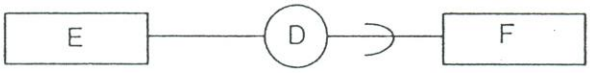
مشارکت E در R : غیر الزامی
مشارکت F در R : الزامی



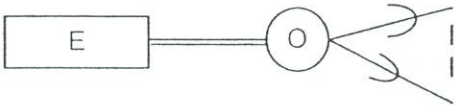
مشارکت غیر الزامی (0,...)
مشارکت الزامی (1,...)



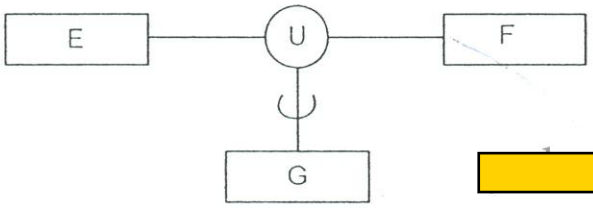
E *'IS-A'* F تخصیص هم‌پوشا



E *'IS-A'* F تخصیص مجزا

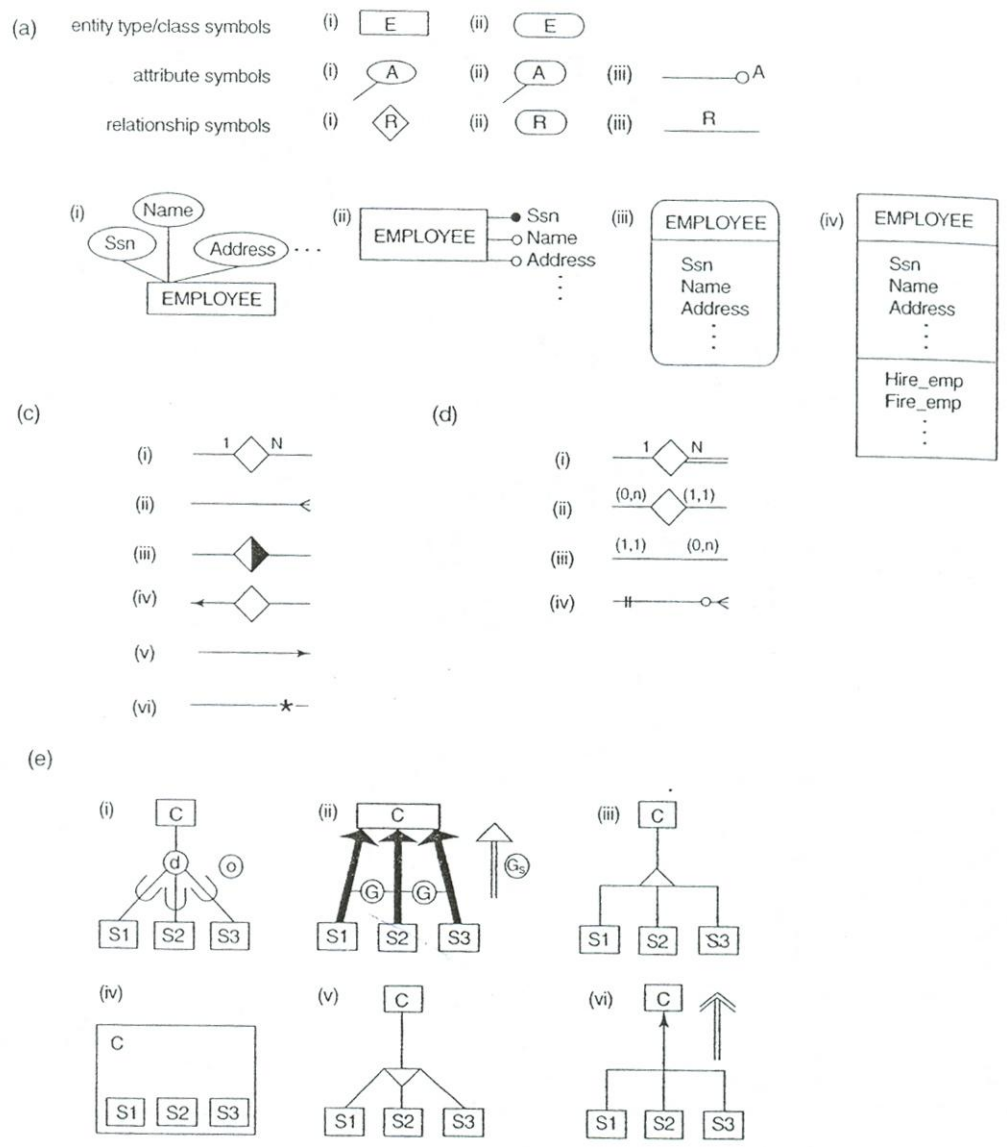


تخصیص کامل {مجزا/هم‌پوشا}



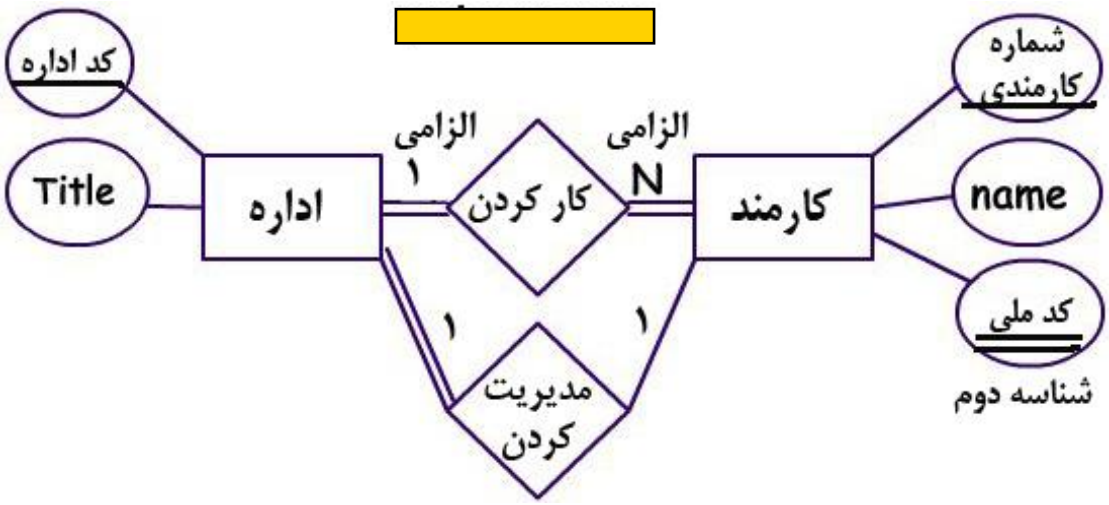
G *'IS-A'* U-Type of E, F

نماد هایی دیگر برای نمایش مدلسازی معنایی :



Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

مثال :



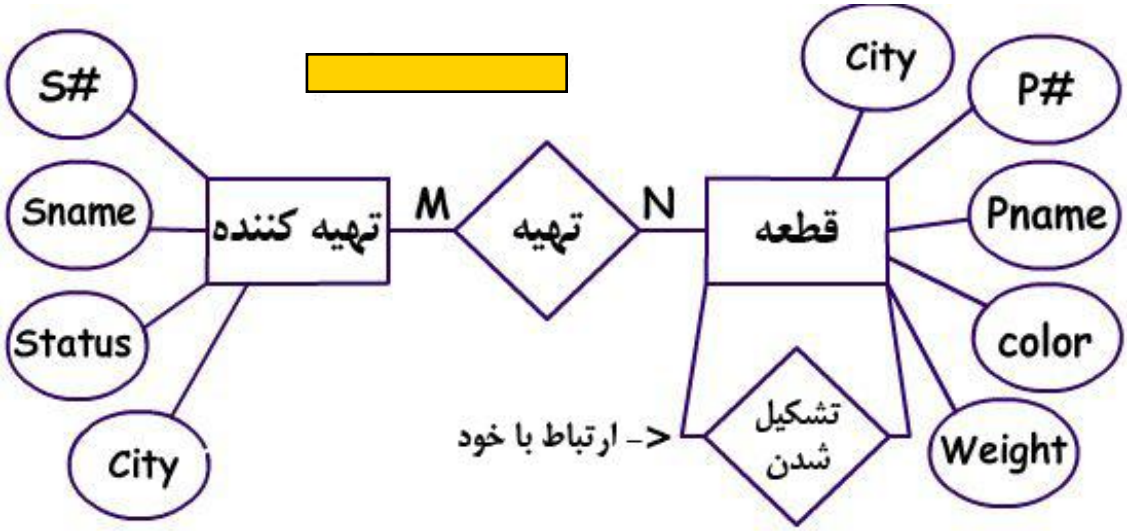
هر اداره مدیر دارد، اما هر کارمندی مدیر نیست.

نکته : هر نوع ارتباط یک چندی دارد (Multiplicity یا نسبت کاردینالیتی یا Cardinality Ratio).

تعریف : چندی نوع ارتباط R بین دو نوع موجودیت Eو F [نه لزوماً متمایز] عبارت است از چگونگی تناظر بین عناصر مجموعه نمونه های E وعناصر مجموعه نمونه های F .

این تناظر می تواند 1:1 ، 1:N یا N:N باشد.

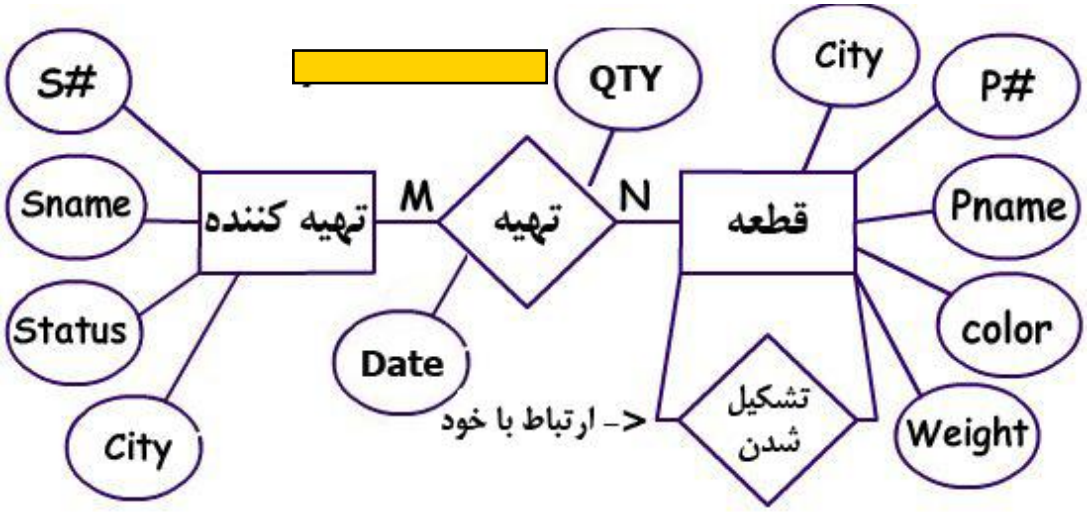
مثال:



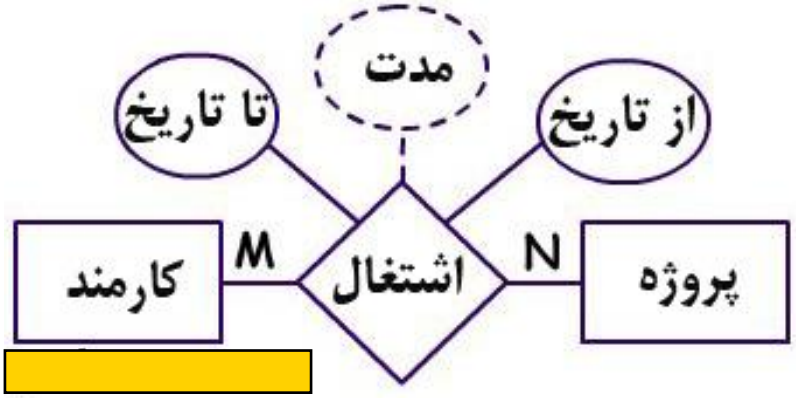
نوع ارتباط می تواند صفت یا صفاتی داشته باشد.

نکته : اگر n تعداد صفات نوع ارتباط R باشد آنگاه $n \geq 0$.

مثال : S1 از P1 چه تعداد تهیه کرده است. پاسخ به این پرسش با افزودن صفت QTY به ارتباط Supply یا "تهیه" امکان پذیر است و صفت چه تعداد p تهیه شده است توسط s در صفات p دیده نشده است.



نکته : وقتی که به این نوع ارتباط صفات زمانی می دهیم . (مثل تاریخ شروع و پایان) می گوییم مدلسازی را زمان مند کرده ایم .



صفت می تواند صفت مجازی باشد اگر بخواهیم صفت مجازی در نظر بگیریم با خط چین نمایش می دهیم و می توانیم آن را محاسبه کنیم .

همچنین می توانیم به جای اینکه مدت را صفت مجازی در نظر بگیریم به جای صفت تا تاریخ صفت مدت را به عنوان صفت اصلی ایجاد کنیم.

همین مشکلات می توانند در performance سیستم دخالت داشته باشند.

کنجکاوی – اگر ارتباط مدیریت را زمان مند کنیم ، چه مشکلی پیش می آید ؟

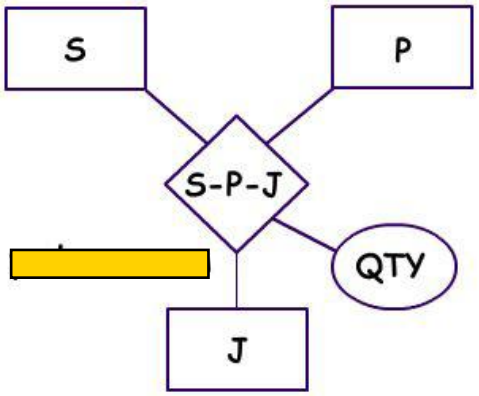
۱. چندی ارتباط تغییر می کند
۲. چندی ارتباط می تواند تغییر کند.*
۳. چندی ارتباط 1:N می شود
٤. چندی ارتباط به قواعد معنایی محیط بستگی دارد.

ابتدای صدای ۲ جلسه دوم

مثال– ارتباط ۳ موجودیتی و نکته مهم آن

تهیه کننده S قطعه P را برای استفاده در پروژه J تهیه کرده است.

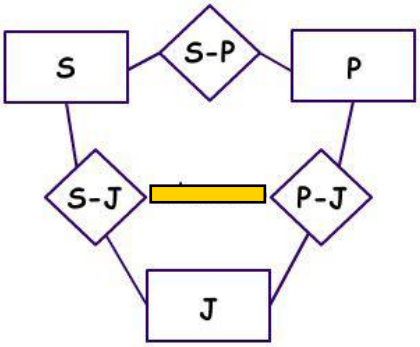
مدل ۱:



سئوال – آیا این محیط به طرز دیگری هم مدل می شود؟

بله به سه طرز دیگر مدل می شود

مدل ۲: ممکن است در مدلسازی به جای ارتباط فوق (مدل ۱)، سه ارتباط دوگانی ببیند .



این دو مدل معادل نیستند از نظر مجموعه اطلاعاتی که به ما می دهند. از مدل ۲، فقره اطلاع های دو موجودیتی به دست می آید.

مثال :

I1 : تهیه کننده S1 قطعه P1 را تهیه می کند.

I2 : قطعه P1 در پروژه J1 استفاده می شود.

I3 : تهیه کننده S1 از تهیه کنندگان [قطعه] برای J1 است.

- از I1 و I2 و I3 لزوماً همیشه نمی توان منطقا استنتاج کرد فقره اطلاعات سه موجودیتی I4 را که از مدل 1 به دست می آید.

I4 : S1 قطعه P1 را برای پروژه J1 تهیه کرده است.

اگر کاربر به اطلاعات سه موجودیتی نیاز داشته باشد با مدل 1 طراحی می کنیم.

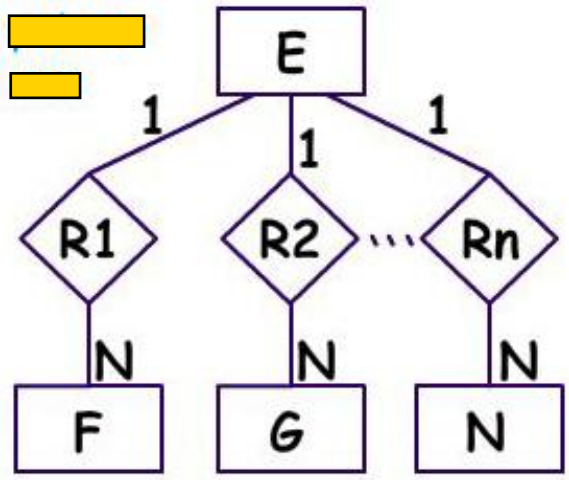
فرض کنید اگر از سه ارتباط دو موجودیتی وجود ارتباط سه موجودیتی را استنتاج کنیم و از آن جا فقره اطلاع های سه موجودیتی را ، در شرایطی که منطقاً این استنتاج درست نباشد می گوییم دچار دام پیوندی (ارتباطی) شده ایم. در اینجا چون حلقه داریم به این دام Ring Connection Trap یا دام پیوندی حلقه ای می گویند.

در مدلسازی باید دقت کنید دچار ring نشود.

نکته تکمیلی : حداقل دو نوع دام دیگر وجود دارد:

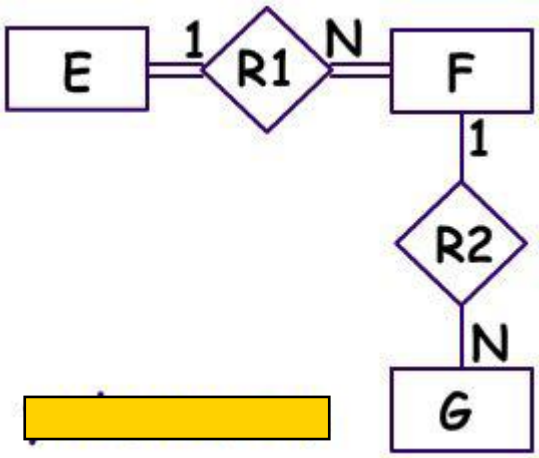
۱. Fan Trap : دام چتری یا چند شاخه
۲. Ckasm Trap : دام شکاف
۳. Connection Trap : دام حلقه ای یا ring

دام چند شاخه یا چتری در وضع زیر بروز می کند:



بر اساس این مدل نمی توان اطلاعات همیشه درست ناظر به نوع موجودیت های F , G , N استنتاج کرد و اگر این کار را انجام دهیم دچار دام چتری می شویم.

دام شکاف در وضع زیر بروز می کند:



نمونه هایی از F, G در R2 شرکت ندارند (چون مشارکت غیر الزامی است) و شما نمی توانید اطلاع های سه موجویتی همیشه درست بین Fو G وE به صرف این دو ارتباط استنتاج کنید و در صورت انجام چنین کاری دچار دام شکاف می شویم.

تمرین : درهمین وضع چه کنیم که این Gap به وجود نیاید؟

کنجکاوی: محیط S-P-J را به دو طرز مدل کردیم، و گفتیم به ٤ طرز مدل می شود، دو طرز دیگر کدامند؟

نوع موجودیت ضعیف: Weak Entity (بحث تکمیلی)

تعریف نوع موجودیت F را ضعیف نوع موجودیت E گویم هرگاه F با E وابستگی وجودی داشته باشد.

نکات :

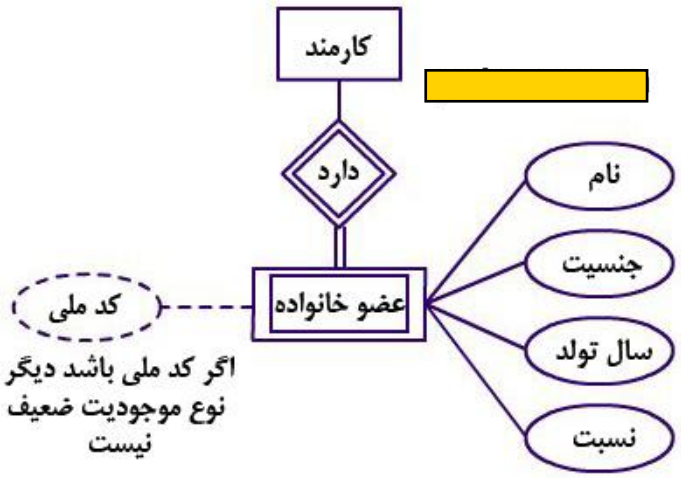
۱ – طرز نمایش (لوزی و مستطیل دو خط برای نمایش نوع موجودیت ضعیف)

۲– نوع ضعیف از خود شناسه ندارد. صرف وجود وابستگی وجودی ایجاب نمی کند که نوع موجودیت را ضعیف ببینیم باید آن موجودیت از خود شناسه نداشته باشد.

قبلا نیز گفته شد قوی یا ضعیف بودن مطلق نیست و بستگی به حیطه معنایی دارد ولی اگر قوی دیدیم حتما باید شناسه داشته باشد.

۳۶:۵۰

مثال :



اگر صفت کد ملی را که ماهیتا شناسه است را وارد کنیم، عضو دیگر نباید ضعیف دیده شود.

۳- نوع ضعیف حداقل یک صفت جدا ساز (ممیزه یا Discriminator) دارد که در مدلسازی با خط چنین نمایش می دهیم

این صفت ممیزه با شناسه قوی ترکیب می شود و برای نمونه های ضعیف کلید می سازد.

تعریف صفت ممیزه : صفتی است که یکتایی مقدار دارد اما فقط در مجموعه نمونه های وابسته به نمونه قوی اش و نه در مجموعه تمام مقادیرش

مثال – نام عضو ها در داخل هر مجموعه تکراری نیست اما در کل ممکن است تکراری باشد:

شماره کارمند	نام عضو
۱۰۰	A B C
۱۰۱	A D J

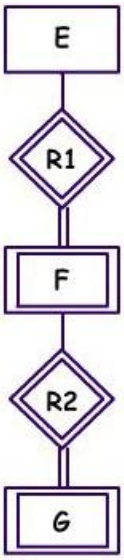
در مثال بالا A در مجموعه اول تکراری نیست ولی در کل دو مجموعه تکراری است.

٤- ارتباط قوی– ضعیف را اصطلاحاً شناسا(Identifying) گوییم. چون از طریق نمونه قوی نمونه ضعیف ها را از یکدیگر تشخیص می دهیم.

۵- مشارکت نوع ضعیف در ارتباط شناسا همیشه الزامی است . ضعیف بدون قوی معنا ندارد.

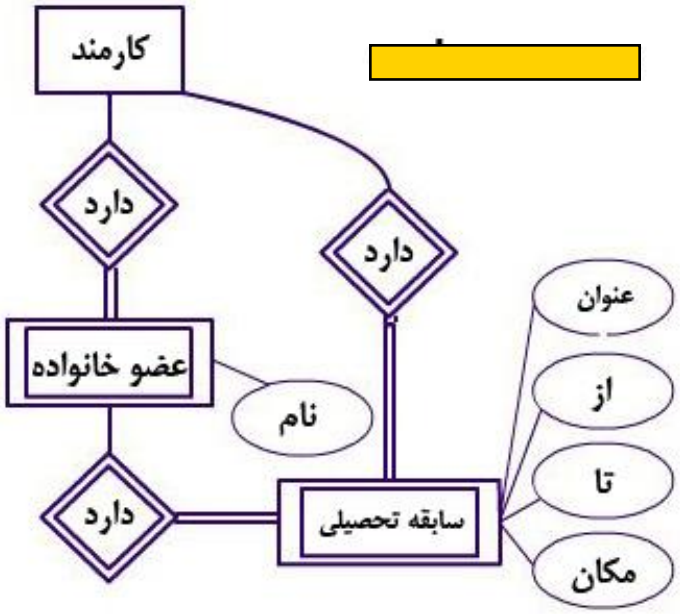
۶- چندی ارتباط شناسا . معمولاً 1:N است به حدی که اصلاً گاهی چندی آن را نمی گویند. ۱:۱ حالت خاص است و اگر M:N بود در ضعیف بدون باید شک کنیم.

۷- نوع ضعیف می تواند خود قوی برای ضعیف دیگر باشد:



در عمل ارتباط ها نباید ۱ یا حداکثر ۲ بیشتر باشد

مثال – کارمند و عضو خانواده

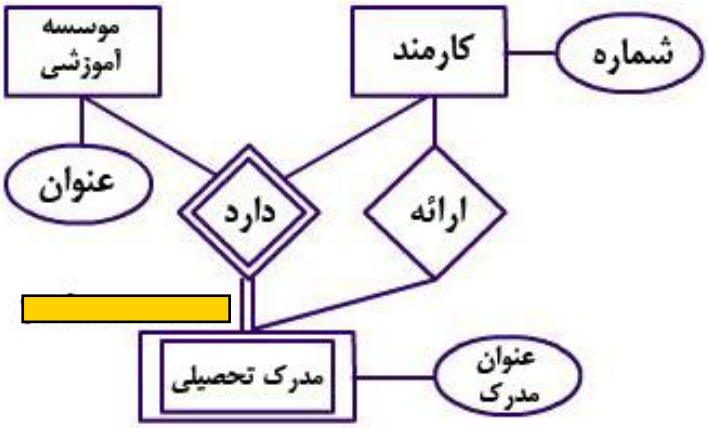


اگر بخواهیم سابقه تحصیلی کارمند را هم مدل کنیم ارتباط کارمند و سابقه تحصیلی را ایجاد می کنیم که در شکل بالا می بینید.

که این راه حل خوبی نیست.

۸- درجه ی ارتباط شناسا معمولا ۲ ولی از نظر تئوریک بیشتر نیز می تواند باشد.

مثال : فرض کنید کاربران راجع به موسسه محل اخذ مدرک اطلاعات می خواهند:



این یک طرز مدلسازی است (و نه لزوما تنها طرح مدلسازی این محیط) که به مناسبت این مثال مطرح شده است.

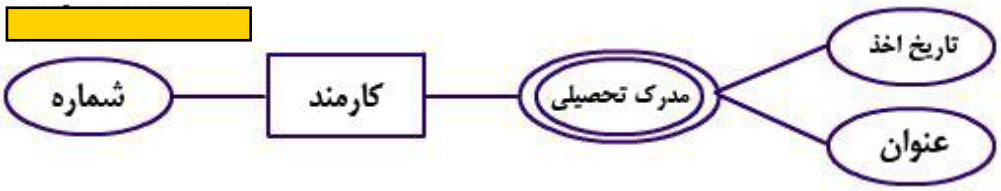
تمرین - همین محیط به چه طرز های دیگری مدل می شود؟

نکته : یک محیط مشخص با یک مجموعه نیاز مشخص می تواند به بیش از یک طرز مدل شود.

۱:۰۱:۰۰

۹- همیشه می توان صفت چند مقداری به ویژه مرکب را با مفهوم نوع ضعیف مدل کرد. عکس این تکنیک توصیه نمی شود.

مثال:



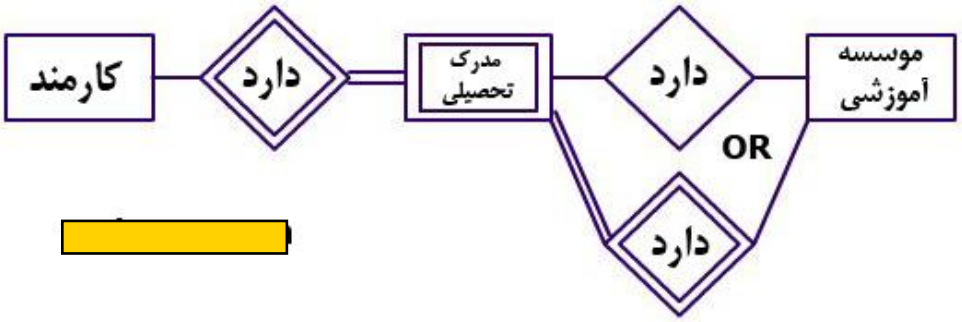
می توانید صفت چند تعدادی را با موجودیت ضعیف (weak Entity) مدل کنید.

در طراحی performance مهم است.

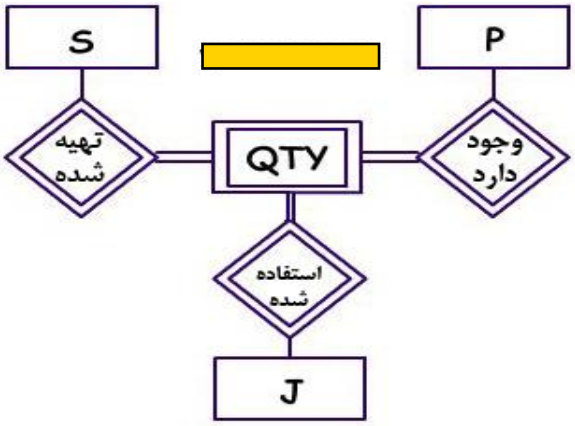
چرا عکس این تکنیک توصیه نمی شود؟

زیرا انعطاف پذیری مدلسازی را از نظر گسترش های بعدی مدل کاهش می دهد.

یک weak می تواند در ارتباط مجزا ضعیف بیش از یک نوع باشد.



محیط s-p-j مدل سوم:



سئوال : در مدل سازی مدل ۳ چه مفهومی را می خواهیم با نوع ضعیف نمایش دهیم؟

مفهوم تعداد به تنهایی نقش Weak Entity را بازی می کند و سه ارتباط دوگانی به ما می دهد.

کنجکاوی: در مدل ۲ نیز سه ارتباط دوگانی داشتیم تفاوت مدل دوم و سوم در چیست؟

فرق اساسی مدل ۳ , ۲ در طرز طراحی DB است. طراحی های یکسان نداریم . در نتیجه پیاده سازی هم می تواند متفاوت باشد و نهایتا میزان کارایی در پاسخگویی به پرسش ها متفاوت خواهد بود.

۱:۱۸:۰۰ break ۱:۳۴۰۰:

EER (بحث تکمیلی) : ER گسترش یافته (Extended)

در روش ER مبنایی بعضی از ارتباط های مهم بین نوع موجودیت ها مطرح نشده اند از آنجا لزوم گسترش ER مطرح شده است.

۱. ارتباط "IS_A" ("گونه ای از" یا "هست یک")
۲. ارتباط "IS_A_PART_OF"
۳. ارتباط با ارتباط

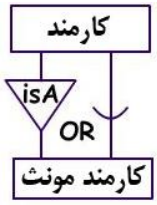
تعریف "IS_A" : ارتباط بین نوع موجودیت عام است و نوع موجودیت های خاص آن

به نوع موجودیت عام اصطلاحاً زبرنوع می گوییم. Super Type

نوع موجودیت خاص اصطلاحاً زیرنوع می گوییم. Sun Type

مثال : نوع کارمند مونث گونه ای است از نوع کارمند.

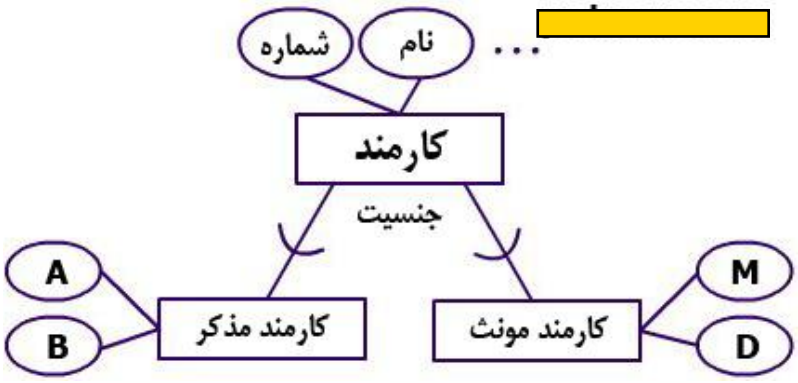
نمایش :



نکته: وقتی که نوع های خاص یک نوع عام را بر اساس یک ضابطه مشخص بازشناسی می کنیم می گوییم از تکنیک تخصیص (ویژه نمایی Specialization) استفاده کرده ایم.

ضابطه تخصیص را Defining Attribute یا صفت معرف می گویند.

مثال: ضابطه تخصیص : جنسیت



نکته : به عکس تخصیص تعمیم(Generalization) می گوییم.

تعمیم عبارت است از بازشناسی یک نوع موجودیت جدید از روی موجودیت های از پیش دیده شده در مدلسازی ۲- زیرنوع مجموعه صفاتی دارد مشترک در تمام زیر نوع ها

مثال:

۳- زیرنوع تمام صفات زیرنوع را به ارث می برد. (وراثت ساختاری)

یادآوری : انواع وراثت :

- وراثت ساختاری ← وراثت صفات Structural
- وراثت رفتاری ← وراثت متد ها – function ها ، Behavioral

تست : با ارتباط IS_A کدام نوع وراثت مدل می شود؟

۱. ساختاری*

۲. رفتاری

۳. هردو

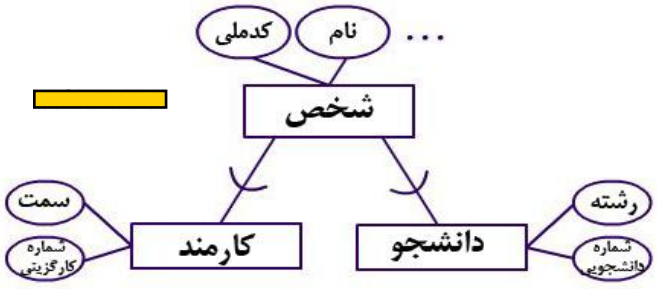
٤. هیچکدام

۴- زیرنوع در عین حال مجموعه صفات خاص خود را هم دارد (حد اقل یک صفت)

مثال ۱: در مثال زیر چند نکته می بینیم:

۱. زیر نوع در عین حال صفات خود را دارد.(غیر از مشترکات)

۲. زیر نوع ممکن است شناسه ای هم داشته باشد که در اینجا شماره دانشجویی و شماره کارگزینی که در واقع شناسه دوم می باشند.

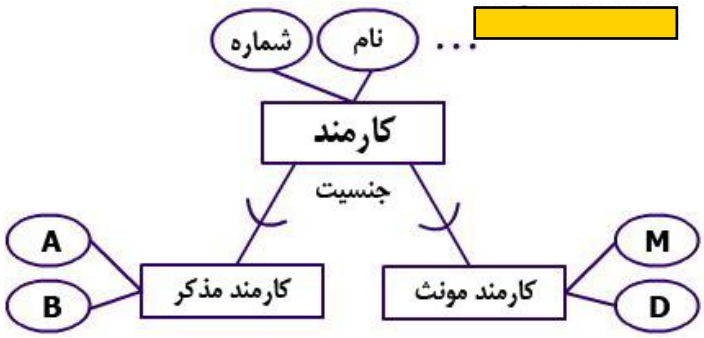


۱:۵۳:۰۰

۵- تخصیص ممکن است کامل باشد ، ممکن است ناقص باشد. Total or Partial

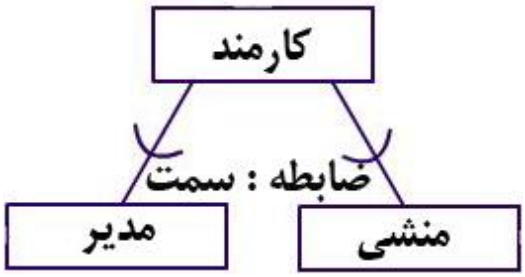
تخصیص وقتی کامل است که تمام زیر نوع ها با توجه به ضابطه تخصیص در مدلسازی داده شده باشند.

مثال ۲ : ضابطه جنسیت تخصیص کامل است



تخصیص وقتی ناقص است که با توجه به ضابطه بازشناسی همه زیر نوع ها دیده نشده باشند

مثال ۳ : تخصیص ناقص

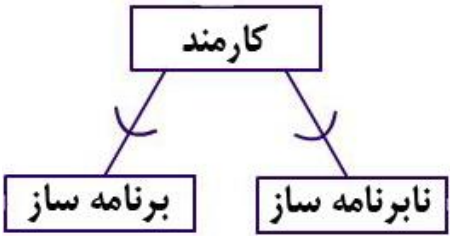


۶- تخصیص ممکن است مجزا (Distinct) یا هم پوشا (Overlapping) باشد.

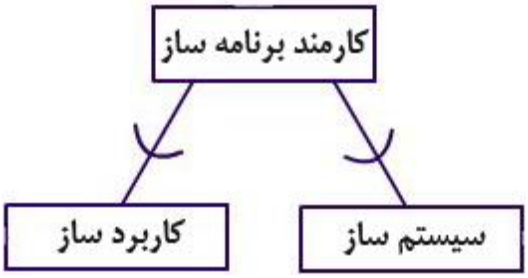
- در تخصیص مجزا یک نمونه از زیرنوع جزء مجموعه نمونه های حداکثر یک زیر نوع است . به بیان دیگر در تخصیص مجزا مجموعه نمونه های زیرنوع ها عنصر مشترک ندارند .

- اما در تخصیص هم پوشا عکس این حالت است یعنی یک نمونه زیرنوع جزء مجموعه نمونه های بیش از یک زیر نوع است.

مثال ٤: مجزا: (مثال دیگر هم مثال جنسیت است)



مثال ٥: هم پوشا : چون ممکن است یک کارمند هم کاربرد ساز باشد هم سیستم ساز



بنابر این با توجه به دو مثال قبل چهار گونه تخصیص داریم:

١. کامل – مجزا

٢. کامل – هم پوشا

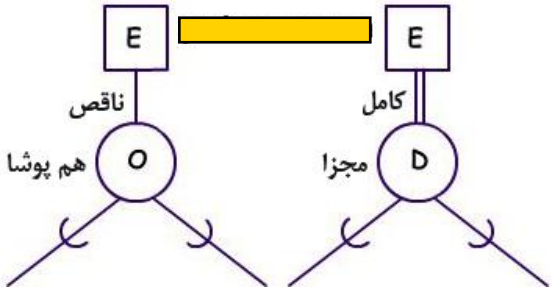
٣. ناقص – مجزا

٤. ناقص – هم پوشا

نمادها :

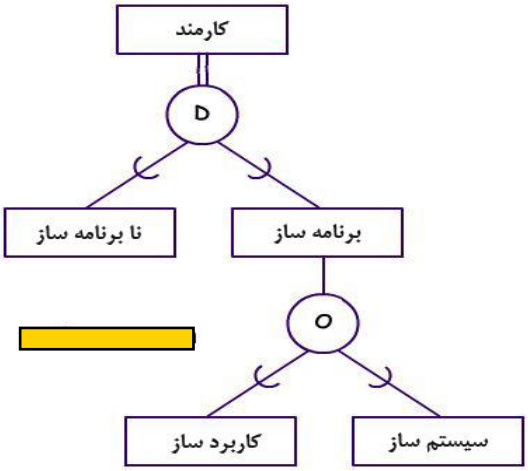
در تخصیص مجزا حرف D (Distinct) و در تخصیص هم پوشا O (Overlapping) در داخل دایره قرار داده می شود.

برای تخصیص ناقص از یک خط و برای تخصیص کامل از دوخط استفاده می شود.



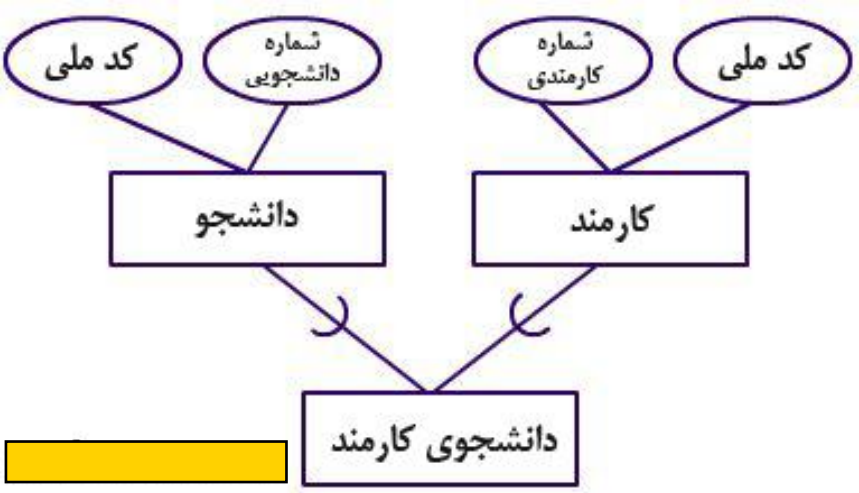
۷- زیرنوع می تواند خود زیرنوع (هایی) داشته باشد.

مثال ٦:



دقت کنید : عمق یک شاخه درخت تخصیص معمولا ۲ ، به ندرت ۳ و هرگز بیش از ۳ نیست.

۸- زیر نوع می تواند بیش از یک زیرنوع داشته باشد با این امکان وراثت چند گانه مدل می شود.



کارمند دانشجو هم صفات دانشجو را به ارث می برد و هم صفات کارمند.

در مثال بالا دانشجوی کارمند حداقل دو شناسه دارد. یعنی هم شماره دانشجویی دارد و هم شماره کارگزینی و چون ممکن است زیرنوع شناسه های دیگری نیز داشته باشد تعداد شناسه ها بیش از دو می باشد.

شناسه های مشترک در دانشجوی کارمند یک بار می آید چون Union است.

دانشجوی کارمند = دانشجو ∪ کارمند

جلسه سوم سه شنبه ۱۳۸۸/۰٤/۲۳ساعت ۷:۳۰

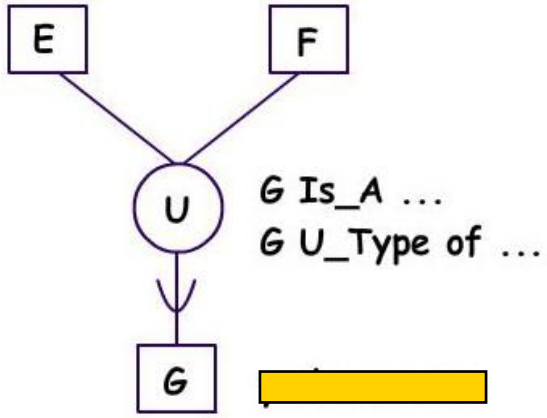
مدلسازی معنایی داده‌ها (ادامه بحث)

۹- زیر نوع U_Type

U : Union Or Category Or دسته

تعریف : نوع موجودیت G را زیر نوع U_Type نوع موجودیت های E و F و ... گوییم هرگاه در مجموعه نمونه های G، نمونه هایی از E و نمونه هایی F و... وجود داشته باشد.

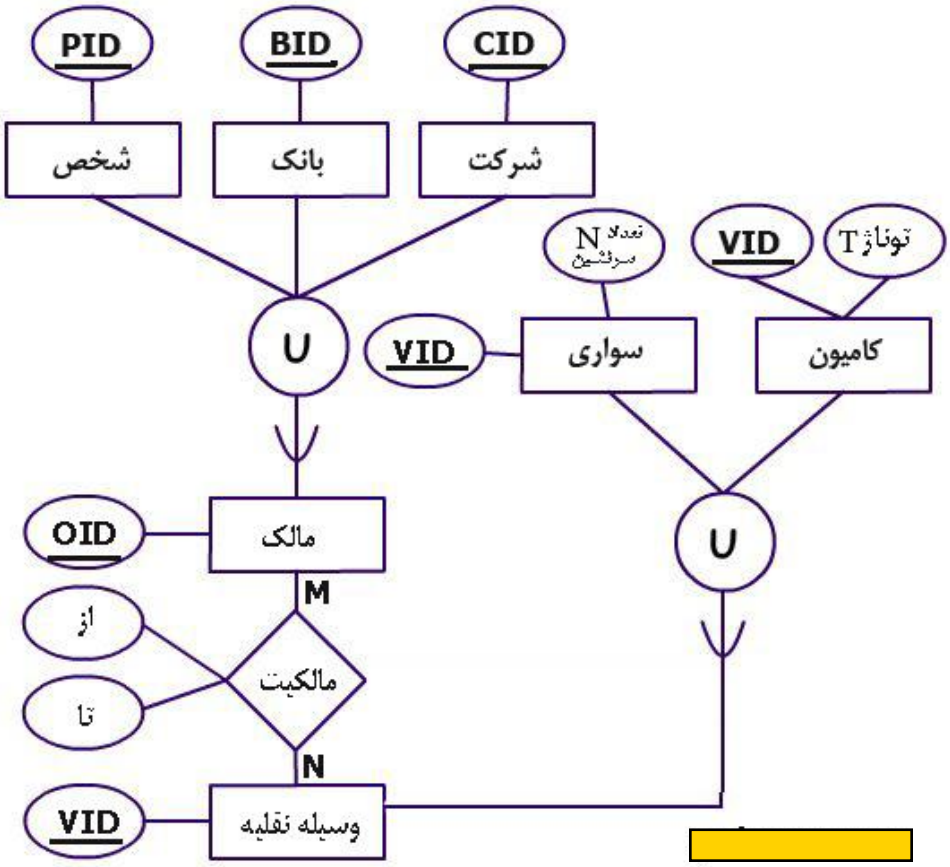
مدل کلی:



مثال : U_Type وقتی مطرح می شود که زیرنوع ها اساساً تایپشان متفاوت باشد.

در این مثال شناسه زیرنوع ها (شرکت ، شخص و بانک) متفاوت است.

Type کامیون و سواری یکی است و آن دسته را نمی توان با تخصیص معمولی انجام داد.



نکات :

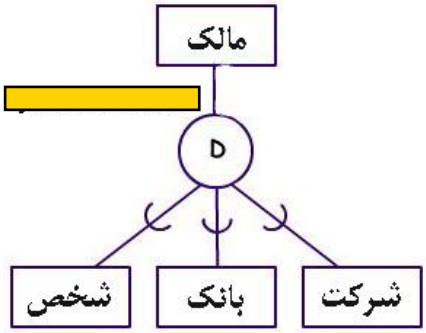
۱. شناسه زیرنوع ها ممکن است از دامنه های متفاوت یا یکسان باشند.

۲. اگر N تعداد زیرنوع ها باشد: $N \geq 2$

۳. دسته ممکن است کامل یا ناقص باشد. وقتی که تمام نمونه های زیر نوع ها در category باشند کامل و گرنه ناقص است یعنی مثلاً در مثال بالا همه بانک ها ، همه شرکت ها و همه اشخاص مالک باشند.

٤. وقتی که category کامل باشد می توان مدلسازی را با تخصیص معمولی انجام داد.

اگر تخصیص کامل بود می توان این چنین هم کشید به شرطی که category کامل باشد.

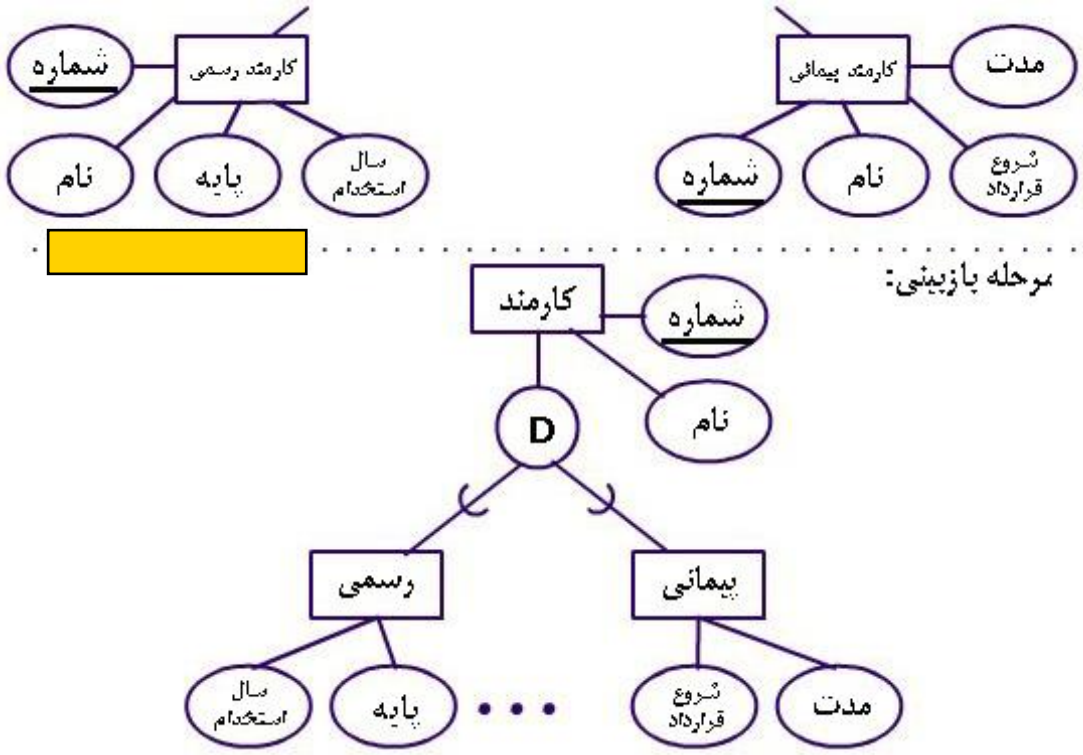


۵. وقتی که شناسه زیرنوع ها از دامنه های متفاوت باشد برای U_Type خود باید شناسه در نظر بگیریم (مالک OID) اما اگر شناسه زیرنوع ها از دامنه های یکسان باشد همان شناسه به U_Type داده می شود.

تکنیک تعمیم **عکس تکنیک تخصیص**(Generalization) است.

تعریف: بازشناسی یک نوع موجودیت جدید به عنوان زیرنوع با داشتن $n \geq 2$ نوع موجودیت از پیش دیده شده (که می شوند زیر نوع های آن زیرنوع)

مثال : فرض می کنیم مدلساز در یک مدلسازی چنین دیده باشد :



درمرحله بازبینی یا در مرحله ادغام دو مدلسازی (اگر جدا انجام شده باشند) متوجه می شویم که این دو مفهوم گونه های خاصی از یک مفهوم عام تر هستند بنابراین قابل تعمیم هستند. به این مفهوم جدید (کارمند) صفات مشترک را می دهیم و این صفات مشترک را از گونه های خاص حذف می کنیم.

شرط لازم برای تعمیم چیست؟

- حداقل درصفت شناسه مشترک باشند.
- حداقل دو زیر مجموعه داشته باشد.

۳۱:۰۰

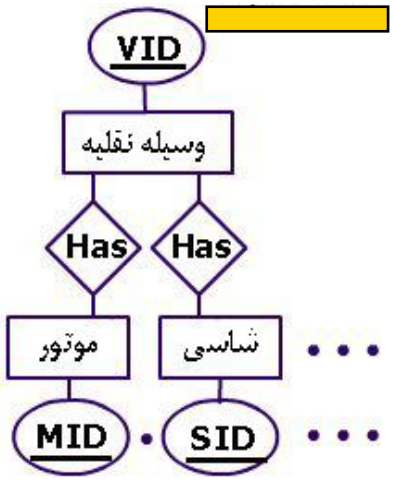
ارتباط "IS_A_PART_OF"

با کمی تسامح ارتباطات Contains و HAS معادل یکدیگر دیده می شوند.

تعریف : ارتباط بین نوع موجودیت کل است ونوع موجودیت های جزء (تشکیل دهنده) آن

توجه داشته باشید تخصیص بین عام و خاص است اما این نوع ارتباط بین کل و جز یا ارتباط بین شامل و مشمول است.

مثال:



نکات :

۱. وقتی که نوع موجودیت های جزء یک نوع موجودیت کل را بازشناسی می کنیم می گوییم از تکنیک تجزیه استفاده کرده ایم . Decomposition

۲. به عکس تکنیک تجزیه ترکیب (Composition) می گوییم. یعنی تشخیص یک نوع موجودیت کل از روی موجودیت هایی که جزءآن هستند.

۳. نوع کل مجموعه صفات خود را دارد.

٤. نوع جزء هم مجموعه صفات خود را دارد.

از ۳ و ٤ نتیجه می گیریم نوع جزء هیچ صفتی از نوع کل را به ارث نمی برد.

تفاوت نوع جزء و نوع موجودیت های ضعیف

۱. نوع جزء از خود شناسه دارد، ضعیف ندارد.

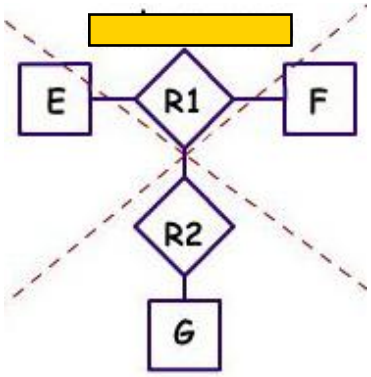
۲. با حذف نوع قوی از مدلسازی نوع ضعیف هم حذف می شود. اما در حالت کل- جزء لزوما چنین نیست . (بحث وابستگی وجودی به معنای دیده شده در بحث قوی- ضعیف دراینجا مطرح نیست)

۴۱:۰۰

ارتباط با ارتباط تکنیک تجمیع Aggregation

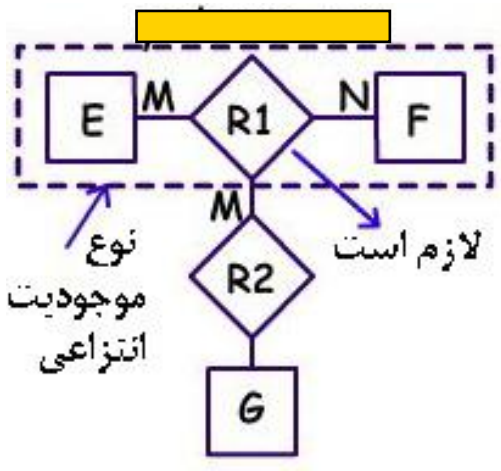
اصطلاح Aggregation در بحث UML برای ارتباط Contains استفاده می شود (برای نمایش شئی مرکب)

در مدلسازی با [E]ER مدلسازی به شکل زیر رایج نیست:



در عمل گاه ممکن است نیاز به ارتباط با ارتباط داشته باشیم برای این منظور از تکنیک تجمیع استفاده می کنیم.

تعریف : تکنیک تجمیع عبارت است از دیدن n>=1 (۱ حالت خاص است) نوع موجودیت شرکت کننده در ارتباط R به صورت یک نوع موجودیت انتزاعی، به منظور مدلسازی ارتباط با ارتباط.



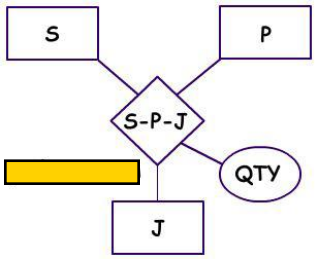
در شکل بالا R2 ارتباط است بین G و موجودیت جدید

دلیل استفاده از تکنیک تجمیع:

دلیل نا مهم : چون بعضی ابزار های مدلسازی ، ارتباط دوگانی بیشتر ، مدل نمی کنند [نمی توانند مدل کنند] این را به دو ارتباط دو گانی تبدیل می کنیم. این دلیل می گوید اگر بخواهیم ارتباط سه گانی یا بیشتر را بشکنیم به تعدادی ارتباط با درجه کمتر از این تکنیک استفاده می کنیم

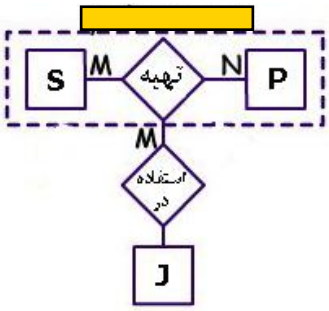
دلیل اصلی افزایش کارایی سیستم است زیرا در همین دو مدل قبل طراحی های متفاوت خواهیم داشت. مثلا در طراحی جدولی مدل اول به ما ٤ جدول می دهد و مدل ۲ پنج جدول از جمله یک جدول نمایشگر ارتباط دو گانی R1 و چنین جدولی را با مدل اول نخواهیم داشت. حال اگر مرتبا به ارتباط R1 (به جدولش) در application نیاز داشته باشیم مدل ۲ بهتر جواب می دهد.(کارا تر است)

مدل ۲:



مثال : همان مدل S-P-J و گفتیم که به چهار طرز مدل می شود:

طرز چهارم : استفاده از تکنیک تجمیع است.



یادآوری:

- طرز ۱: سگانی
- طرز ۲ : سه تا دوگانی
- طرز ۳: weak Entity و..
- طرز ٤: استفاده از تکنیک تجمیع

کنجکاوی: روش مدلسازی چه ویژگی های باید داشته باشد؟

نکته : ادغام ER های جزیی کاربران : User's ER's Integration

درعمل وقتی محیط بزرگ باشد آن را به تعدادی زیر محیط تقسیم می کنیم. برای هر زیر محیط یک [E]ERD تهیه می کنیم. سپس این نمودارها را در هم ادغام می کنیم.

توجه : تکنیک هایی که در EER دیدیم مثل Is-A ، U-Type ، تخصیص ، تعمیم Category ، تجزیه ، ترکیب و تجمیع به ویژه در این مرحله (ادغام) استفاده می شوند.

ابتدای صدای ۲ جلسه سوم ساعت ۸:۳۰

مقدمات طراحی DB

- ✓ ساختار داده جدولی TDS
- ✓ پایگاه داده جدولی TDB
- ✓ زبان جدولی TDBL ← SQL

مفهوم جدول صرفا امکانی است برای مفهوم ریاضی رابطه (relation)

مفاهیم جدولی : (عناصر ساختاری TDS)

۱. مفهوم نوع جدول : Table Type

۲. ستون

۳. سطر

با این سه مفهوم ما باید بتوانیم طراحی منطقی را انجام بدهیم ، داده ها و ارتباطات بین آنها را هم نمایش بدهیم.

نکته : هر DS حداقل یک عنصر ساختاری اساسی دارد.

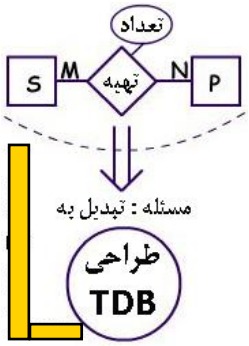
- عنصر ساختاری اساسی عنصری است که به کمک آن نوع موجودیت یا نوع ارتباط یا هر دو نمایش داده می شود (در مرحله طراحی منطقی)
- در TDS فقط یک عنصر اساسی داریم، همان نوع جدول
- خود Table Type هم یک نام دارد و یک سرآیند (نام ستون ها)
- TDB : تعدادی table type است .(نوع جدول) [که متناسباً طراحی می کنیم]
- ستون : برای نمایش صفت است.
- سطر : برای نمایش نمونه موجودیت و یا نمونه ارتباط
- سطر تشکیل شده است از تک مقدارها (همان رابطه نرمال) : در یک جدول در محل تقاطع هر ستون و هر سطر Single Value داریم.

TDB [از نظر محتوای داده ای] : مجموعه ای است از نمونه های متمایز یک یا چند نوع سطر Row Type (نوع سطر را نوع جدول مشخص می کند)

مثالی از طراحی منطقی با TDS:

مسئله : تبدیل به طراحی TDB

در تبدیل مدلسازی به طراحی حالات مختلفی داریم.



دراین حالت سه نوع جدول لازم است یک جدول برای هر entity و یک جدول هم برای خود ارتباط.

در جدولS :

- در هر سطر جدول S یک نمونه نوع موجودیت S را می بینیم (یک Tapple)
- S# کلید اصلی می باشد.

S: | **P:**

CITY	STATUS	SName	S#	P#	PName	Color	Weight	City
C1	10	SN1	S1	P1	PN1	Blue	5	C1
C2	17	SN2	S2	P2	PN2	RED	…	C2
…	…	…	…	…	…	…	…	…

کنجکاوی – چرا برای نمایش ارتباط M به N جدول جداگانه باید طراح شود.

در جدول نمایشگر نوع ارتباط، شناسه نوع موجودیت های طرفین، همچنین صفت یا صفات ارتباط در صورت وجود باید وارد بشود.

هر سطر جدول SP یک نمونه ارتباط می باشد.

SP

S#	P#	QTY
S1	P1	100
S2	P2	90
…	…	…

کلید اصلی جدول نمایشگر ارتباط M:N از ترکیب شناسه شرکت کنندگان در ارتباط تشکیل می شود و احیاناً صفاتی از ارتباط.

کلید خارجی- FK (تعریف مقدماتی- در عمل): ستون C در جدول T2 کلید خارجی(FK) است هرگاه در جدول T1 کلید اصلی(PK) باشد.

کنجکاوی : سطح فایلینگ این DB چگونه است ؟ (چند فایل است و چگونه ذخیره می شود)

نکته : تناظر بین این جدول ها (موسوم به جدول های مبنا) و فایل های سطح فایلینگ پایگاه ممکن است 1:1 یا 1:N یا N:M باشد. (در عمل معمولا 1:1 یعنی یک table یک فایل)

از نظر ساختار هر یک از این فایل ها می تواند ساختار خاص خود را داشته باشد. مثلا فایل های ترتیبی

در حاشیه: ساختار ترتیبی(فایل ترتیبی):

کلیدی KS یا Key Sequence

زمانی ES یا Entry Sequence

ساده ترین ساختار فایل برای DB ساختار ترتیبی زمانی ES است.

KS نیاز به سورت دارد.

پیاده سازی

برای پیاده سازی ابتدا باید جدول را به سیستم معرفی کنیم.

کنجکاوی : چند نوع table داریم ؟ (در عمل و در تئوری)

- ✓ جدول مبنا: جدولی که برای محیط طراحی می کنیم و از سیستم می خواهیم آن را ذخیره کند.
- ✓ مجازی view
- ✓ موقت temporary table
- ✓ لحظه ای snapshot

شمای پایگاه داده ها (Database Schema):

```
CREATE TABLE S
( S# CHAR(6) NOT NULL,
SNAME CHAR(20),
STATUS SMALLINT,
CITY CHAR(15),
PRIMARY KEY ( S# )
CHECK STATUS IN(5 , 25) ) ;
```

// چک ها را می توان موقع تعریف ستون هم نوشت

```
CREATE TABLE P
( P#  CHAR(6) ,
PNAME CHAR(20),
COLOR CHAR(6),
WEIGHT NUMERIC(5,1),
CITY CHAR(15),
PRIMARY KEY ( P# ) ) ;
```

```
CREATE TABLE   SP
( S#  …   ,
P#   …    ,
QTY …. )
PRIMARY  KEY ( S#, P# ),
CHECK QTY IN(0,500)
FOREIGN KEY S# , REFRENCES  S,
…
FOREIGN KEY P# , REFRENCES ,
…
) ;
```

تعریف شمای پایگاه داده ها : نوعی برنامه شامل دستورهای تعریف داده ها(Data Definition) و کنترل داده ها (DataControl) و نه دستورهای پردازش یا عملیات در داده ها (Data Manipulation)

به تعریف هر کدام از جداول Table Scheme گفته می شود.

عملیات در پایگاه جدولی (با استفاده از SQL ***)*** Structured Query Language

- ✓ زبان استانده سیستم های جدولی (رابطه ای)

٤٥:٤٥

۱- بازیابی :

```
SELECT items list
FROM table(s)
[WHERE conditions]
[ORDER BY collumns]
[GROUP BY collumns]
[HAVING conditions]
```

نکته : حاصل اجرای هر دستور SELECT معتبر یک جدول می باشد.

HAVING امکانی است برای اعلام شرط یا شرایط ناظر به گروه سطر ها یعنی همان نقشی را دارد که WHERE نسبت به سطر دارد.

```
SELECT S.S# AS SN , S.SNAME AS SNAM
FROM S
WHERE STATUS >=20 AND
(CITY =’C2’ OR CITY=’C7’)
ORDER BY SNAME
```

- QUERY بالا شماره و نام تهیه کنندگان با وضعیت .... و ساکن شهر ... را به ما می دهد.
- در S.S# حرف S نام جدول است که می تواند بسته به SELECT اجباری یا اختیاری باشد که به آن Qualifier یا قید ستون می گویند.
- AS برای دگرنامی می باشد.
- شرط یا شرایط بازیابی در حالت کلی یک عبارت بولی است که به آن مسند گزینش می گویند.
- ORDER BY برای مرتب سازی است(ASSENDING/DESENDING) و پیش فرض آن ASSENDING است. ستونی که می خواهیم بر اساس آن مرتب کنیم می توانیم بر اساس نام یا شماره مکانی(Position Number) مشخص نماییم.

کنجکاوی : چگونه جدول جواب را نام دار کنیم ؟

- ساده ترین روش با AS
- استفاده از جدول کمکی

حال می خواهیم جواب همین query را در جدول مورد نظر خودمان وارد کنیم. فرض می کنیم جدول را چنین تعریف کرده باشیم:

```
CREATE TEMPORARY TABLE TEMP
(SN CHAR(16) NOT NULL,
SNAME CHAR(20))
PRIMARY KEY SN;
```

جدولی موقتی ایجاد کرده ایم حال می خواهیم نتیجه SELECT را در این جدول بریزیم.

تکنیک درج گروهی:

```
INSERT INTO TEMP

      SELECT ….. همان پرس و جوی بالا...
```

حال اگر QUERY زیر را بریزیم داده های بازیابی شده را نشان می دهد:

```
SELECT * FROM TEMP;
```

کنجکاوی : آیا راه دیگری هم وجود دارد ؟ آن کدام است ؟

دستور تعریف جدول همانند اساساً نسخه سازی است.

```
CREATE TABLE

      T2 AS T1 WITH DATA;
```

نکته : فرق بین base table و temporary table چیست؟

جدول مبنا جدولی است پایا (persistent) حال آن که جدول موقت یک جدول گذرا (transient) است یعنی پس از اتمام session کاربر، توسط سیستم حذف می شود.

ابتدای صدای ۳ جلسه سوم

QUERY : شماره تهیه کنندگانی را بدهید که بیش از یک قطعه تهیه می کنند.

SP:

S#	P#	QTY
S1	P1	100
S1	P2	90
S1	P3	70
S2	P1	60
S2	P2	80
S3	P1	90

```
SELECT S#

FROM SP

GROUP BY S#

HAVING COUNT(*)>1
```

COUNT : تابع سطرشمار است.

COUNT(*) : همه سطر ها حتی اگر NULL باشد.

```
COUNT [DISTINCT] (col)
```

سطر های تکراری و NULL را نمی شمارد.(تکراری ها را یک بار می شمارد و سطر های بدون دخالت NULL را نمی شمارد.)

جدول جواب:

S#
S1
S2

مثال در رابطه با COUNT :

QUERY زیر تعداد تهیه کنندگان را به ما می دهد:

```
SELECT COUNT(*) FROM S;
```

فرض کنید جواب N1 است.

تعداد تهیه کنندگانی که قطعه تهیه کرده اند:

```
SELECT COUNT DISTINCT (S#) FROM SP;
```

جواب S1 و S2 و S3 : سه جواب دارد. فرض کنید جواب N2 است.

با فرض اینکه قاعده جامعیت ارجاعی در این DB رعایت شده باشد در این صورت پاسخ QUERY دوم نمی تواند از پاسخ QUERY اول بیشتر باشد. چون S# در SP کلید خارجی است و نمی تواند کلیدی در SP باشد که در S نباشد.

تست: با فرض این که قاعده جامعیت ارجاعی دراین DS رعایت شده باشد دراین صورت:

۱. N2>N1
۲. N2<N1
۳. * N2≤N1
۴. N2=N1

- Fk زیر مجموعه Pk است.

GROUP BY جدول داده شده در FROM Clause را منطقاً بازآرایی می کند به نحوی که مقدار ستون داده شده در GROUP BY در هر گروه یکسان باشد. حال نوبت HAVING است. HAVING گروه به گروه تست می کند .

در اینجا تست شمارش سطر ها می باشد گروه به گروه تست می کند هر کدام که COUNT(*)>1 باشد را بر می گرداند:

```
SELECT S#
      FROM SP
      GROUP BY S#
HAVING COUNT(*)>1;
```

SP:

	S#	P#	QTY
گروه S1	S1	P1	100
	S1	P2	p
	S1	P3	
گروه S2	S2	P1	
	S2	P2	
گروه S3	S3	P1	

تمرین : همین پرسش بدون استفاده از group by , having نوشته شود.

مثال : بزرگترین مقدار ستون را باز می گراند

```
SELECT MAX(status) AS x  FROM s;
```

مفهوم پرسش فرعی (*Nested Query*) یا پرسش تو در تو)

نام تهیه کنندگان قطعه p2 را بدهید:

این QUERY حداقل به ۸ روش نوشته می شود.

راه اول : به روش زیر که به آن join شبیه سازی شده گویند:

در این روش داخل S را می گردد اگر در SP نظیر داشت و P# ='P2' جواب را بر می گرداند:

```
SELECT S.SNAME

FROM S , SP

WHERE S.S# = SP.S#

AND SP.P# ='P2';
```

راه دوم : همچنین می توان به روش زیر که روش Natural Join است نوشت:

```
SELECT SNAME

      FROM S JOIN SP

WHERE P# = 'P2';
```

راه سوم :

- پرسش فرعی یک ساختار SELECT درون پرسش دیگر است.

تعریف : پرسش فرعی (Nested Query) پرسشی است درون پرسش دیگر و در واقع SELECT های تو در تو داریم به پرسش بیرونی Outer Query و به پرسش درونی Inner Query گویند.

پرسش درونی ممکن است Correlated و یا Non Correlated باشد.(به هم بسته و یا نا به هم بسته)

پرسش درونی را وابسته یا به هم بسته به پرسش بیرونی می گوییم هرگاه در پرسش درونی به ستونی از جدولی از پرسش بیرونی ارجاع داده باشیم، در غیر اینصورت نا بهم بسته است. این گونه پرسش ها به صورت موازی اجرامی شوند parallel execution و نه سریال.

```
SELECT SNAME FROM S
WHERE S# IN (SELECT S# FROM SP
WHERE P# =’P2’)
```

در پرسش بالا به جای IN می توان ANY گذاشت. ANY به معنی some one می باشد.

QUERY بالا نا به هم بسته است.

در پرسش های non Correlated سیستم ابتدا پرسش درونی را به طور کامل اجرا می کند و آن گاه پرسش بیرونی را.

فرض کنید جواب پرسش داخلی {S1,S2} باشد پرسش به شکل زیر می شود:

```
SELECT SNAME FROM S
WHERE S# IN (S1,S2);
```

به IN عملگر تعلق می گوییم.

Q : شماره تهیه کنندگانی را بدهید که مقدار وضعیت آن ها ماکزیمم نباشد.(کوچکتر از حداقل یکی باشد. در واقع فقط MAX جواب نیست.)

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

S:

S#	Status
S1	10
S2	25
S3	17
S4	25
S5	15

با توجه به SQL و جدول S جدول جواب به شکل زیر خواهد بود:

S#
S1
S2
S3

مثال بالا مثالی است از پرسش های به هم پیوسته.

توجه داشته باشید که ANY یا SOME با مفهومی استفاده می شوند که در بحث های ریاضی به آن theta می گویند. Theta هر کدام از عملگر های زیر می تواند باشد:

= != < <= > >=

مثل ANY> که مفهوم > همراه با ANY آمده است.

مثال:

```
SELECT S# FROM S
      WHERE status < (SELECT MAX (status) FROM s)
```

ابتدا مقدار MAX را به دست می آورد و سپس SELECT بیرونی اجرا می شود.(نا بهم پیوسته)

۴۷:۳۰

نکته: وقتی که جواب پرسش درونی تک مقدار باشد می توان پشت select درونی مستقیما از خود theta استفاده کرد. (ANY یا SOME لازم نیست)

مثالی دیگر: شماره همشهری های S1 را می دهد:

```
SELECT S# FROM S
      WHERE city = (SELECT city FROM s WHERE S# = 'S1')
```

طرز اجرای روش هفتم از دید کاربر:

```
SELECT S.SNAME FROM S
      WHERE 'P2' IN (SELECT SP.P# FROM SP
WHERE SP.S# = S.S#)
```

در Query بالا در Where Clause ارجاع دادیم به ستونی از جدول S، پس به هم پیوسته است.

همچنین به جای IN می توانستیم ANY= یا SOME= بگذاریم که خود دو حالت می شود که تا اینجا ۹ شکل شد.

در بهم پیوسته بازای هر سطر از جدول S یک بار SELECT درونی به تمامی اجرا می شود:

مثلا در اولین بار:

برای سطر S1 : نتیجه اجرا ی پرسش درونی {S1,S2,S3} ← نتیجه نهایی : نام S1 جواب است.

در دومین بار:

برای سطر S2 : نتیجه اجرا ی پرسش درونی {....} ← نتیجه نهایی : ....

تست: فرض تعداد سطر های دو جدول t1 و t2 به ترتیب N1 و N2 است. دریک پرسش تودرتو چنین داریم:

```
SELECT ……..   FROM T1 …

               SELECT ……..   FROM T2
```

پرسش بیرونی چند بار اجرا می شود؟

جواب بستگی دارد که این دو Correlated باشند یا نباشند اگر Correlated باشند جواب N1 (یک بار به ازای هر سطر جدول بیرونی) اگر نباشند جواب ۱ است.

همچنین در SQL فیچر هایی مثل LIKE ، NOT LIKE ، BETWEEN و … نیز داریم.

مثال:

```
SELECT S# FROM S

      WHERE SNAME LIKE '%A'
```

۱:۰۳:۰۰

دستورات ذخیره سازی:

- INSERT
- DELETE
- UPDATE

عمل delete حالت خاصی از update است از دیدگاه فایلینگ.

مثال UPDATE:

به هنگام سازی یک سطر:

```
UPDATE S
      SET CITY='C3' , STATUS=STATUS+5
      WHERE S# = 'S3';
```

به هنگام سازی چند سطر:

```
UPDATE S
      STATUS=STATUS+10
      WHERE CITY = 'C2' OR CITY='C10';
```

حذف تک سطر : چون کلید داده ایم.

```
DELETE FROM SP
    WHERE S# =’S1’ AND P# =’P3’;
```

حذف مجموعه ای : هر سطری که S1 را دارد حذف می کند.

```
DELETE FROM SP
    WHERE S# =’S1’;
```

حذف کل دیتا : تمام سطر ها حذف می شود:

```
DELETE FROM SP;
```

سئوال : DELETE FROM SP چه فرقی دارد با DROP TABLE SP ؟

کنجکاوی: با اجرای دستور DROP TABLE درسیستم چه پیش می آید؟

۱. داده ها حذف می شوند.

۲. تعریف جدول از کاتالوگ سیستم حذف می شود.

۳. اگر جدول هایی به این جدول ارجاع داشته باشند باید به نحوی اصلاح شوند.

٤. فضای اشغال شده توسط جدول و ساختارهای مربوطه اش آزاد می شوند.

٥. اگر روی این جدول view هایی تعریف شده باشند نا معتبرمی شوند.

نکته: در عمل سیستم ها به کاربر امکاناتی می دهند . مثلا روی یک جدول مبنا دید تعریف کرده باشیم درخواست drop آن جدول می تواند اجرا نشود.

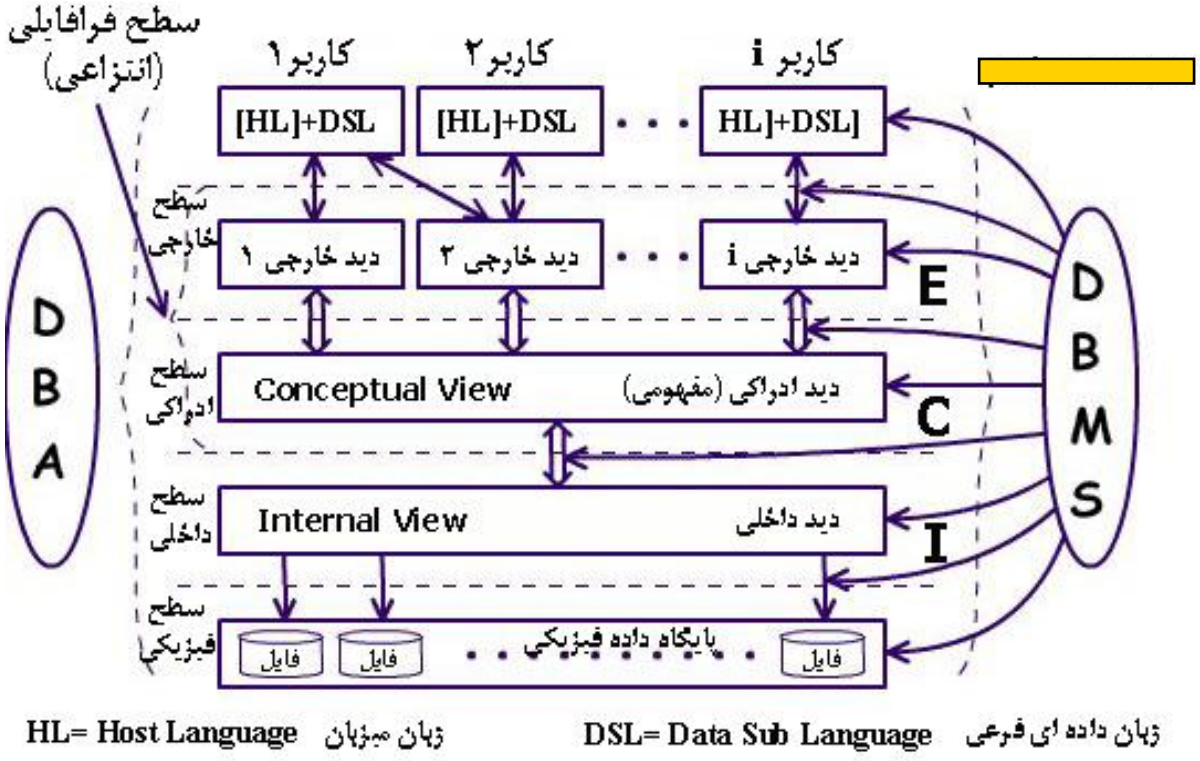
جلسه چهارم چهارشنبه ۲۴/۴/۱۳۸۸ ساعت ۷:۳۰

معماری پایگاه داده ها : (پیشنهادی ANSI)

نکات مهم تر: معماری ANSI یک ضابطه مهم برای تشخیص واقعی بودن DBMS هاست.

معماری سه سطحی

نمایش:



اجزای معماری:

سطح ۱:

- کاربر
- زبان میزبان [HL]
- زبان داده های فرعی DSL

سطح ۲:

- دید (نما) خارجی
- دید ادراکی
- دید داخلی

سطح ۳:

- تبدیلات بین سطوح
- پایگاه داده ها ی فیزیکی (فایل)
- DBMS (نرم افزار کنترل متمرکز)
- DBA (تیم تخصصی مدیریت)

نکته : معماری ANSI در واقع ضابطه ای مهم برای تشخیص واقعی بودن DBMS هاست. اگر DBMS ی از آن پشتیبانی نکند در بهترین حالت آن را شبه DBMS گوییم.

شرح سطوح سه گانه : این سه سطح برای تعریف و کنترل داده ها هستد ← معماری شمایی

دید ادراکی Conceptual View *مهم:

نکات:

- دید طراح است (در مرحله طراحی منطقی) نسبت به داده های ذخیره شده.
- این دید جامع است شامل نیازهای داده ای همه کاربران محیط.
- این دید مبتنی است بر یک DS [از یکDM] و در محیط فرا فایلی مطرح است.
- این دید [پس از طراحی] باید تعریف شود.
- به تعریف دید ادراکی شمای ادراکی گوییم. (سطح ادراکی) : نوعی برنامه شامل دستورهای DD و DC و نه DM است. شمای ادراکی به سیستم داده هامی شود سیستم اطلاعات موجود در آن را به نحوی در جایی نگهداری می کند تا بعداً از آن استفاده کند.

مثال : با فرض TDS(RDS) جدول های مبنای S و P و SP → دید ادراکی

معماری سه سطحی مد نظر ANSI شامل سه سطح تعریف و کنترل داده ها هستند. در واقع سه شما داریم: (خارجی- ادراکی - داخلی) به همین خاطر به آن معماری سه شمایی نیز می گویند.

مثال شمای ادراکی : CREATE TABLE ها

نکته : سیستم اطلاعات لازم را در کاتالوگ یا دیکشنری داده ها یا متاداده ها ذخیره می کند.

کنجکاوی : در کاتالوگ چه اطلاعاتی ذخیره می شود؟

- اطلاعات مربوطه به دید ادراکی.
- اطلاعات مربوطه به دید خارجی
- اطلاعات مربوطه به دید داخلی
- اطلاعات مربوط به خود کاربران
- اطلاعات مربوط به امنیت داده ها
- اطلاعاتی مربوطه به جامعیت داده ها

نکته : درسیستم های جدولی : کاتالوگ خود تعدادی جدول است که سیستم ایجاد می کند.

مثال : یکی از جدول های کاتالوگ چنین است

جدول کاربر (طراح پیاده ساز)

SYSTABLE:

نام جدول	ایجاد کننده	تاریخ ایجاد	تعداد ستون ها	کلید اصلی	....
S	x	D1	4	S#	
P	x	D2	5	P#	

وقتی دستور ایجاد جدولی را کاربر طراح صادر می کند:

```
CREATE TABLE S
```

سیستم عملیات زیر را برای این منظور انجام می دهد:

```
INSERT INTO SYSTABLE VALUES(S,x,d1,4,S#)
```

سایر table ها:

```
SYSINDEXES , SYSUSERS , SYSVIEWS
```

تست: آیا کاربر می تواند محتوای کاتالوگ را تغییر دهد؟

بله . به طور غیر مستقیم یعنی کاربر برنامه ساز نمی تواند با دستورات insert ، update و delete روی جدول های کاتالوگ عملیات انجام دهد بلکه با دستور های DD و DC به طور غیرمستقیم به عنوان مثال CREATE TABLE از دستور های DD است.

مثال دیگر : کاربر می نویسد

```
DROP TABLE S;
```

سیستم عملیات زیر را انجام می دهد:

```
DELETE FROM SYSTABLES WHERE TABLENAME='S'
```

تغییرات در کاتالوگ توسط کاربر برنامه ساز انجام شدنی نیست.

شرح دید داخلی (نکات مهم تر)

دید خود DBMS است نسبت به داده های ذخیره شده (و نیز تا حدی دید طراح در مرحله طراحی فیزیکی)

- این دید مبتنی است بر یک یا چند ساختار فایل (منطقی و گاه مجازی)
- تناظر بین مجموعه ساخت های سطح ادراکی (مثلا جدول های مبنا) و مجموعه ساختار های سطح داخلی (فایل های منطقی یا مجازی). این تناظر می تواند 1:1 یا 1:N یا N:M می تواند باشد. بیان ساده مثلا یک جدول-یک فایل
- این دید هم باید تعریف شود (توسط خود سیستم و با اطلاعاتی که کاربر [طراح-پیاده ساز] به سیستم می دهد)
- به تعریف دید داخلی شِمای داخلی Internal Schema می گوییم نوعی برنامه که خود سیستم ایجاد می کند و در کاتالوگ نگهداری می شود.. مثلا با فرض تناظر یک به یک با هر CREATE TABLE سیستم یک فایل تعریف و ایجاد می کند.

مثال برای اطلاعاتی که طراح پیاده ساز می دهد:

کاربر مرحله ۱:

```
CREATE TABLE S;
```

سیستم در مرحله ۱: با فرض تناظر یک به یک، یک Declare انجام می دهد:

```
DEFINE SFILE ….
      Struct : SN
      ….
      Ccity;
```

Struct و ... Ccity رکورد های داخلی می باشند (Internal Record)

ساختار فایل Sfile فعلا ES[14] یا IS[15] است.

کاربر مرحله ۲:

```
CREATE INDEX CX ON S (CITY) [CLUSTERED[16]]
```

سیستم در مرحله ۲:

شاخص درختی روی فیلد Ccity از فایل Sfile ایجاد می کند.

آگاهی: اکثر package ها روی کلید اصلی automatic index می زنند. ولی برای ستون های دیگر باید درخواست بدهیم.

[14] Entry Sequenced
[15] Indexed Sequential
[16] شاخص خوشه ساز

نکته : وقتی می گوییم دید داخلی مبتنی است بر یک یا چند ساختار فایل منطقی یعنی DBMS (در مورد فایلینگ پایگاه) همان چیز هایی را می داند که برنامه ساز فایل پرداز در محیط زبان های برنامه سازی می داند اما در مورد فایلینگ فیزیکی چیزی نمی داند در نتیجه :

DBMS می داند :

- چه فایل هایی وجود دارد.
- فرمت رکورد هر فایل
- ساختار هر فایل (منطقی)
- استراتژی دست یابی به رکورد ها (Access Strategy)
- فیلد (های) کلید
- فیلد نظم
- اندازه جاری هر فایل
- ظرفیت هر فایل (میزان گسترش)
- اطلاعات شناسنامه ای فایل

DBMS نمی داند : (اطلاعات مربوطه به فایلینگ فیزیکی)

- طرز نشست فایل ها روی رسانه (پیوسته / گسسته)
- لوکالیتی رکورد ها (میزان نزدیکی فیزیکی رکورد های منطقا هم جوار (۹ درجه دارد) همچنین لوکالیتی بر روی بعضی عملیات روی فایل تاثیر زمانی دارد)
- بافرینگ (سطح os)

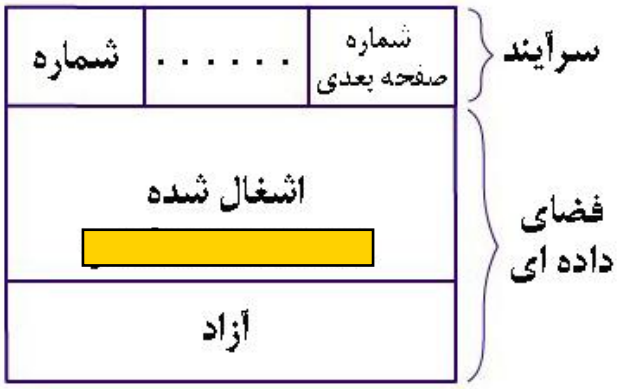
نکته : فیلد کلید ، نظم و جستجو لزوما یکی نیستند.

نکته : لوکالیتی ممکن است

- درون فایلی intera file (رکورد های یک فایل)
- بین فایلی inter file (لوکالیتی بین رکورد های حداقل دو فایل که به نحوی با هم ارتباط دارند.)

نکته : فایلینگ مجازی سطحی از فایلینگ است که در آن یک فایل به صورت تعدادی page دیده می شود.

یادآوری : page معادل مجازی مفهوم block است روی دیسک.



بنابراین در سیستم هایی که سطح فایلینگ مجازی را دارند DBMS پایگاه داده ها را به صورت مجموعه ای از مجموعه صفحات می بیند .

مثال :

نام فایل	تعداد صفحات
sfile	20
Pfile	8
spfile	50

بین صفحات یک ساختار منطقی وجود دارد و درون هر صفحه رکورد ها ساختار خود را دارند (مرتب/نامرتب)

ابتدای صدای ۲ جلسه چهارم

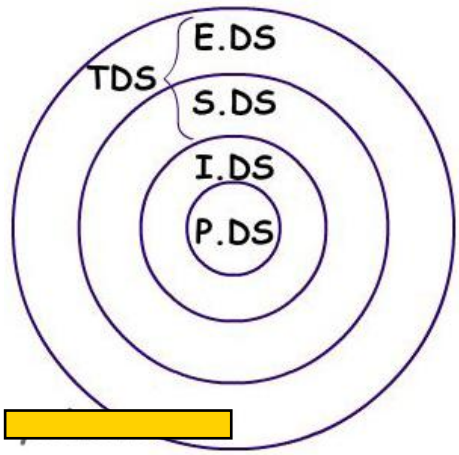
دید خارجی

- دید یک کاربر خاص است نسبت به داده های ذخیره شده (هر کاربر دیدهای خاص خود را دارد)
- این دید جزئی است (و نه جامع). :نشان دهنده محدوده داده ای مورد نیاز یک کاربر (بخشی از داده ها که یک کاربر مجاز است ببنید)
- این دید هم در یک سطح فرافایلی مطرح است و بر یک DS [از یک DM] مبتنی است. معمولاً همان DS که دید ادراکی بر آن مبتنی است

مثال : اگر DS برای سطح ادراکی جدولی باشد، در سطح خارجی هم DS معمولا جدولی است اما از نظر تئوریک لزومی ندارد DS در این دو سطح یکی باشد.

کنجکاوی : مفهوم DS در معنای عام (از نظر DM) در چند سطح مطرح است؟

پاسخ: با توجه به معماری ANSI در ٤ سطح مطرح است:



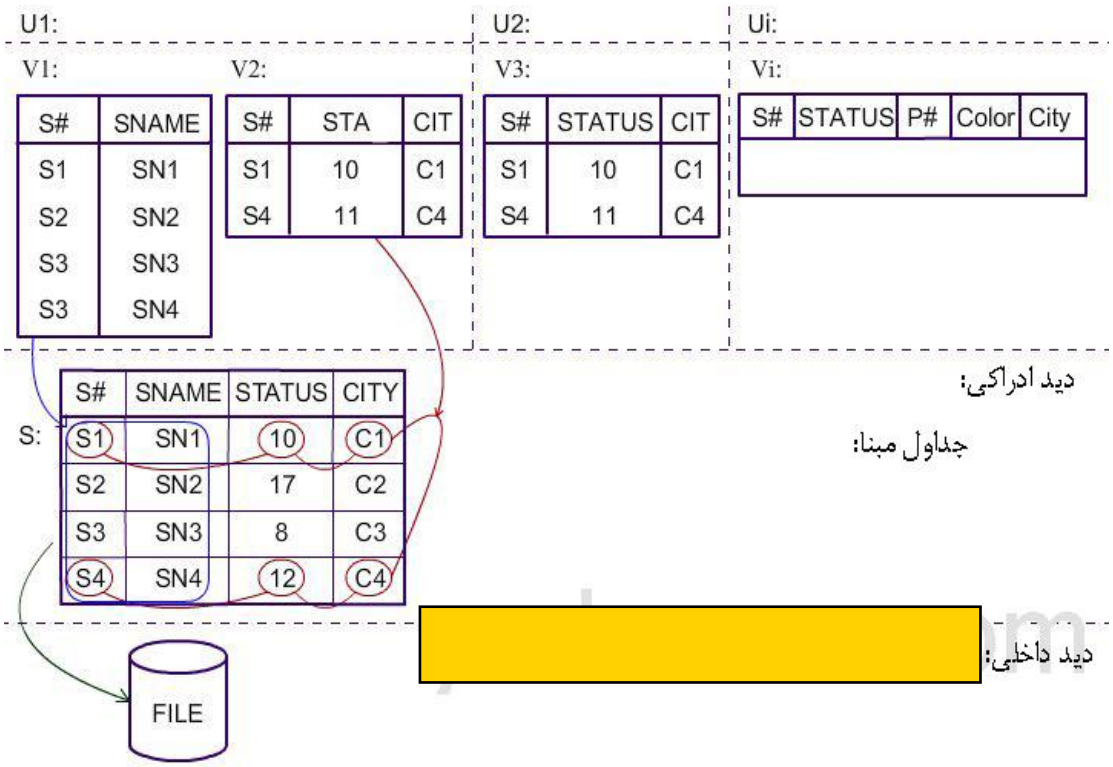
نکته: سطح physical و internal مربوط به فایلینگ است.(فایلینگ یک DS خارجی است.)

TDS شاملی Conceptual و External می باشد.

- این دید روی دید ادراکی تعریف و طراحی می شود.
- این دید هم باید تعریف شود.
- به تعریف دید خارجی می گوییم شمای خارجی که همان سطح خارجی است. (یک نوع برنامه است شامل دستورات DD و معدود دستور های DC)
- اطلاعات شمای خارجی هم در کاتالوگ نگهداری می شود.
- یک کاربر ، می تواند چند دید داشته باشد.
- چند کاربر می توانند اشتراک در یک دید داشته باشند.

نکته : دید خارجی در سیستم های جدولی خود نوعی جدول است اما مجازی، به این معنا که داده ذخیره شده خاص خود ندارد و سیستم برای آن فایل خاص ایجاد نمی کند.

مثال: با فرض TDS جدول های مبنای S ، P و SP مفروضند.



آنچه در شکل بالا می بینیم:

- کاربر ۱ : کاربر می تواند چند دید داشته باشد
- کاربر ۱: هیچ چیز از DB نمی بیند به جز دو ستون از جدول S
- دید خارجی یک نوع پنجره است.
- کاربر سطح خارجی می تواند با رعایت Syntax ستون ها را به نام دیگر بنامد(City به CIT)
- دید ۲ و دید ۳ مثالی از چند کاربر در یک دید می باشد(آخرین نکته)
- در سیستم های جدولی مفهوم دید خارجی مفهومی است بسیار پویا، به این معنا که انواع دید ها از نظر ساختار می توان تعریف کرد.

کنجکاوی : مزایا و معایب دید خارجی چیست؟

عملیات از دید خارجی در DB (عملیات در دید)

- این بحث را در تئوری گاهی View Operation و گاهی View Updating می گویند.(Updating در اینجا یعنی عملیات ذخیره سازی)

۱- بازیابی :

ازدستور select استفاده می کنیم.

فرض : کاربر دیدش را با دستور CREATE VIEW تعریف کرده است

مثال : دستورات ایجاد دید های مثال قبل را در زیر می بینیم:

```
CREATE VIEW V1 [(S# , SNAME)]

      AS SELECT S.S# , S.SNAME

      FROM S;
```

نکته : وقتی که نام ستون های دید همان نام ستون های Base Table باشد نیازی به قید آنها نیست.

```
CREATE VIEW V2 (SN , STA , CITY)

      AS SELECT S.S# , S.STATUS , S.CITY

      FROM S
      WHERE CITY != C2

      [WITH CHECK OPTION];
```

نکته : Select Clause در دستور CREATE VIEW اجرایی نیست ، اعلانی است. (یعنی هیچ داده ای بازیابی نمی شود یعنی امکانی است برای مشخص کردن محدوده داده ای کاربر(Data Scope))

With Check Option : برای کنترل عملیات ذخیره سازی از دید در DB از نظر رعایت محدودیت دید(View Consternate) یا شرط داده شده در تعریف دید می باشد.

مثال : اگر کاربر بخواهد از دید V2 سطری در DB درج کند، که مقدار ستون CITY در آن C2 باشد، سیستم این درخواست را رد می کند.

اصل: دستور (حکم) عمل کننده [روی شِمای خارجی] در دید خارجی حتی الامکان باید تبدیل شود به دستور(هایی) عمل کننده روی شمای ادراکی و سپس به یک قطعه برنامه ای عمل کننده روی شمای داخلی و نهایتا در DB فیزیکی. با توجه به این اصل ما (حداقل) در دو سطح تبدیل داریم بنابر این تبدیلات بین سطوح (تبدیلE/C یا External To Conceptual و تبدیل C/I یا Conceptual to Internal) می باشند.

- گاه به این تبدیلات (E/C,C/I) محاسبه دید هم می گویند View Computation هم می گویند که البته Overhead خاص خود را هم دارد.

نکته : یک نوع تبدیل دیگر هم متصور است :

E/E : External to External

این تبدیل وقتی لازم می شود که روی دید، دید تعریف کرده باشیم. View definition on view که کاربرد خاص خود را دارد.

۴۸:۰۰

کنجکاوی : کدامیک از این سه تبدیل در چه وضعی لازم نمی شود:

E/E لازم نمی شود، هرگاه از تکنیک تعریف "دید روی دید" استفاده نکنیم.

E/C لازم نمی شود، هرگاه از مفهوم "دید خارجی" استفاده نکنیم. از دید خارجی وقتی استفاده نمی کنیم که:۱- سیستم تک کاربری باشد. ۲- در مواردی برای افزایش سرعت برنامه(مستقیما برنامه را روی Base Table می نویسیم)

کنجکاوی در کنجکاوی : C/I چه زمانی انجام نمی شود؟

مثال از تبدیلات بین سطوح در بازیابی:

```
SELECT SN
      FROM V2
      WHERE STA <=10;
```

جدول جواب:

S#
S1

کاربر چه چیزی را خواسته بوده؟

تبدیل E/C :

```
SELECT S#
FROM S
WHERE STATUS <=10 AND CITY !='C2'
```

اگر " AND CITY≠'C2' " را ننویسیم S3 نیز به جدول جواب اضافه می شود در حالی که اجازه دیدن آن را ندارد.

سیستم در تبدیل E/C عمل محدودیت دید را اعمال کند (View Computation انجام می دهد). یعنی شرط یا شرایط داده شده در تعریف دید را AND می کند با شرط داده شده در دستور بازیابی از دید.

حال باید تبدیل C/I انجام گیرد.

شبه کد تبدیل C/I : برای سادگی فرض می کنیم سیستم رکورد به رکورد به ما می دهد . (هر بار یک رکورد در ibuf)

در کد زیر STAX Value نام شاخص روی Status می باشد.

```
Open Sfile(R,ibuf,…)
Loop : logical read sfile ON STAX Value <=10;
If ibuf.city != 'C2' then
Move S# to Ubuf;     ← بافر کاربر
Loop control ….
END
```

این شبه کد تبدیل می شود به :

```
PSEEK = physical seek
PREAD
```

ابتدای صدای ۳ جلسه چهارم

۲- **عملیات ذخیره سازی** (از دید در DB)

ایده ی اصلی ← لزوما از همه انواع دید ها نمی توان عملیات ذخیره سازی در DB انجام داد. (همه دیدها updatable نیستند) به بیان دیگر همیشه تبدیل E/C همیشه امکان پذیر نیست.

۱- درعمل [SQL استاندارد] ← محدودیت های شدید وجود دارد (view ها در عمل اکثرا برای بازیابی هستند)

محدودیت های دید در برگه های پیوست می باشد.

بطور خلاصه برای اینکه دید updatable باشد:

- دید باید روی یک جدول مبنا تعریف شده باشد.
- هرسطر دید باید متناظر باشد با یک سطر مشخص از جدول مبنا. این شرط وقتی برآورده است که view دارای کلید باشد.(کلید جدول مبنا در دید باشد.)
- هر ستون دید باید متناظر باشد با یک ستون مشخص از جدول مبنا از نظر type name

مثال برای دید updatable : دید V2 قابل عملیات به هنگام سازی است.

نکته: در این بحث مجاز بودن کاربر به انجام عمل فرض است شدنی بودن را بررسی می کنیم.

```
UPDATE V2
SET STA = STA + 5
WHERE S#='S1'
```

تبدیل E/C :

```
UPDATE S
      SET STATUS = STATUS + 5
WHERE S#='S1'
```

مثال ۲:

```
UPDATE V2
SET CIT = 'C2'
WHERE S# = 'S4'
```

از نظر تئوریک این درخواست باید طرد شود چون با محدودیت دید (با شرط داده شده در تعریف دید) تعارض دارد. در عمل(SQL) اگر از with check option استفاده کنیم این درخواست طرد می شود.

کنجکاوی : اگر از with check option استفاده نکنیم چه پیش می آید؟

Update انجام می شود اما دیگر در محدوده ای V2 قابل دیدن نیست.

نکته : اما همین دید V2 که قابل عملیات به هنگام سازی است در عمل درج مشکل دارد.

مثال:

```
INSERT INTO V2
      VALUES (S5,15,'C5')
```

تبدیل :

```
INSERT INTO S
      VALUES (S5,?,15,'C5')
```

اگر در تعریف جدول S برای Sname گزینه NOT NULL انتخاب کرده باشیم. این درخواست طرد می شود زیرا محدودیت هیچ مقدار ناپذیری رعایت نشده است.

حال فرض می کنیم به جای C5 نوشته باشیم C2

```
INSERT INTO V2
      VALUES (S5,15,'C2')
```

همان نکته ای که در مورد update گفتیم اینجا پیش می آید یعنی اگر از with check option استفاده کنیم طرد می شود وگر نه درج انجام می شود اما از محدوده دید کاربر خارج می شود. یعنی از دید کاربر درج تبدیل به delete شده.

حال فرض می کنیم به جای S5 نوشته باشیم S3

```
INSERT INTO V2
      VALUES (S3,15,'C2')
```

به دلیل عدم رعایت محدودیت یکتایی مقادیر کلید این درخواست درج طرد می شود. حال آنکه کاربر بی اطلاع بوده است.(کاربر S3 را نمی دیده)

خلاصه : درخواست درج طرد می شود به دلیل عدم رعایت محدودیت های :

- یکتایی مقادیر کلید
- دید
- هیچ مقدار ناپذیری

نکته : (ادامه بحث به طور تئوریک)

- دسته اول دید ها: پس دید های تعریف شده روی یک جدول و دارای کلید (Key Freserving حافظ کلید) قابل به هنگام سازی هستند اما مشکل هایی هم دارند.
- دسته دوم دیدها : دیدهایی هستند که روی یک جدول مبنا تعریف می شود اما فاقد کلید هستند. این دیدها در عمل غیر قابل عملیات ذخیره سازی هستند اما در تئوری اگر اصل زیر را رعایت نکنیم، تبدیل E/C درتئوری انجام می شود.

اصل : عمل تاپلی (سطری) از دید، باید کماکان تاپلی (سطری) در سطح پایین تر انجام شود.

مثال : اگر بخواهیم یک سطر را از یک دید حذف کنیم، این عمل در تبدیل E/C باید سطری انجام شود. یعنی منجر به حذف یک سطر از جدول شود و نه بیشتر.

مثال : فرض می کنیم سطر های S چنین باشند:

S:

S#	SName	STATUS	QTY
S1	a	10	C1
S2	b	15	C2
S3	a	7	…
S4	c	18	

و دید کاربر چنین باشد : (دید فاقد کلید می باشد)

V4:

SName	STATUS
a	10
b	15
a	7
c	18

عمل درج در عمل ناممکن است.(درتئوری هم باید کلید داشته باشد)

اعمال حذف و update اگر اصل تاپلی رعایت نشود، انجام می شوند.

مثال :

```
DELETE FROM V4
      WHERE Sname='a'
```

چون دو سطر با Sname='a' داریم با اجرای این دستور هر دو سطر حذف می شوند. یعنی تناظر یک به یک سطری به دلیل عدم وجود کلید تضمین نیست.

کنجکاوی : حال اگر در همین جدول S به جای 7 داشته باشیم 10 چه پیش می آید؟

همان مشکل کماکان وجود دارد .

- در چنین وضعی اگر در تعریف V4 نوشته باشیم:

```
CREATE VIEW V4
      AS SELECT DISTINCT Sname , Status
      FROM S;
```

چه پیش آید ؟

USER سطر ۳ در V4 را نمی بیند و مشکل کماکان وجود دارد.

یادآوری : برای اعمال محدویت یکتایی مقدار در عمل clause های زیر را دایم.

۱- Primary key
۲- Unique Key

هر دو Clause بالا باعث یکتایی مقدار می شوند اما با Primary key محدودیت هیچ مقدارناپذیری نیز اعمال می شود.

- دسته سوم دیدها:

دیدهای دارای صفت مجازی (ستون Calculated)

جلسه پنجم شنبه ۲۷/٤/۱۳۸۸ ساعت ۷:۳۰

- **دسته سوم دیدها:**

دید های دارای حداقل یک ستون یا صفات مجازی (مشتق، محاسبه شده)

این گونه دید ها غیر قابل ذخیره سازی هستند تبدیل E/C ناممکن است.

مثال :

SP:

S#	P#	QTY
S1	P1	100
S1	P2	80
S2	P1	90
S2	P2	80
…	…	…
S3	P1	90

```
CREATE VIEW V3 (PN,SQ)

AS SELECT SP.P# , SUM(QTY) AS SQ

FROM SP

GROUP BY P#;
```

در این مثال SQ ستون مجازی است:

V3:

PN	QTY
P1	270
P2	180
P3	90
…	…

نکات:

- عملیات ذخیره سازی مثل update روی SQ قابل انجام نیست. (زیرا SQ ستون متناظر زیرین ندارد- stored data ندارد)
- به این دیدها گاه دید های آماری گویند چون توابع آماری مثل SUM و... در آنها به کار می رود.
- دیدی که در آن GROUP BY استفاده شده باشد updatable نیست.
- درتئوری: دید V3 قابل عملیات ذخیره سازی نیست. دلیل آن تعارض معنایی(semantic conflict) بین دید V3 و جدول (رابطه) SP.

جدول SP می گوید تهیه کننده قطعه را تهیه کرده است. اما دید V3 کل تعداد تهیه شده است و این تعارض معنایی دارد و به همین دلیل mapping نباید انجام گیرد.

مثال: فرض می خواهیم بدون توجه به تعارض معنایی و محدودیت SQL استاندارد عمل زیر را انجام دهیم:

```
DELETE FROM V3
      WHERE  PN='P1'
```

تبدیل E/C :

```
DELETE FROM SP
      WHERE  P# ='P1'
```

تمام P1 ها حذف می شود.

پس بدون توجه به تعارض معنایی و محدودیت SQL استاندارد این عمل انجام شدنی است.

عمل درج نیز ناممکن است:

```
INSERT INTO V3  VALUES(P4,400)
```

یعنی از P4 در مجموع ۴۰۰ تا تولید شده . حال این که کجا تولید شده معلوم نیست.

- **دسته چهارم دیدها:**

دیدهای تعریف شده روی بیش از یک جدول (رابطه ی مبنا):

معروف ترین این دید ها دید های پیوندی هستند. join [natural]

توضیح مقدماتی join طبیعی:

T1:

A	B
a1	b1
a2	b2
a3	b3

T2:

C	A
c1	a1
c2	a1
c3	a2
c4	a4

T3= T1JOIN T2:

A	B	C
a1	b1	c1
a1	b1	c2
a2	b2	c3

حال اگر V حاصل پیوند باشد دید V در عمل (SQL استاندارد) نمی پذیرد. (غیر قابل به هنگام سازی)

در تئوری :

- دید پیوندی Pk-Pk (در واقع Sk-Sk) دیدی است که در آن ستون پیوند در هر دو جدول PK است. این گونه دید ها قابل به هنگام سازی هستند (زیرا تناظر یک به یک بین سطر های دید و سطر های جدول مبنای زیرین وجود دارد)
- اما جز این اگر باشد (PK-FK یا FK-FK یا NK-NK) اگر محدودیت تناظر یک به یک (عمل سطری در دید کماکان (پس از تبدیل E/C) سطری انجام شود) را دخالت ندهیم این دید ها هم قابل عملیات به هنگام سازی هستند.

یادآوری: NK-NK یعنی هیچ نوع کلیدی در پیوند نباشد.

مثال : فرض کنیم جدول های مبنای SX و SY را داشته باشیم:

SX:

S#	SName	City
S1		
S2		
S3		
S4		

SY:

S#	Status
S1	
S2	
S3	
S4	

کاربر دید V4 را اینچنین تعریف می کند:

V4:

S#	SName	Status	City
S1			
S2			
S3			
S4			

```
CREATE VIEW V4
      AS SELECT SX.S# , SY.Sname , SY.Status , SX.City
      FROM SX JOIN SY;
```

در اینجا در واقع جدول S را به شکل یک view می سازیم.

تناظر یک به یک سطری و ستونی در دید V4 با جداول مبنا وجود دارد.

بررسی عملیات بر روی V4 :

ابتدا delete :

```
DELETE FROM V4 WHERE S# ='S2';
```

تبدیل E/C : یک دستور تبدیل به دو دستور

```
DELETE  FROM  SX  WHERE S# = 'S2'
```

و

```
DELETE  FROM  SY  WHERE S# = 'S2'
```

نکته : در تبدیل E / C یک دستور ممکن است به یک یا چند دستور تبدیل شود و ممکن است اصلا تبدیل انجام نشود

تست: اگر N تعداد دستور های حاصل از تبدیلE/C باشد:

۱. N≥1
* ۲. N≥0
۳. N=1
۴. N>4

نکته : درعمل UPDATE دید V4 تنها اگر #S را UPDATE کنیم این عمل هم در SX باید انجام شود هم در SY . در غیر این صورت یا در SX یا در SY

این به صورت اتوماتیک انجام می شود .

```
INSERT INTO V4
      VALUES (S5,SN5,15,C5)
```

تبدیل E/C :

```
INSERT      INTO SX     VALUES (S5,SN5, C5)     و
INSERT      INTO SY  VALUES (S5,15)
```

در عمل Package ها چنین عملی را انجام نمی دهند بلکه خود برنامه ساز باید مثلا با نوشتن trigger به سیستم بفهماند که وقتی می گوییم درج در V4 قسمتی از آن باید در SX و قسمتی در SY درج شود.

۴۱:۰۰

تمرین:

```
V5 = S join SP (PK-FK)
V6 = S join SP (NK-NK)
```

آیا V5 و V6 قابل عملیات به هنگام سازی هستند؟

اگر تناظر یک به یک را رعایت کنیم نه جز آن بله.

تمرین :

V7 = R1 op R2

R1 و R2 رابطه یا جدول هستند. و op می تواند یکی از عملیات UNION و Intersect و Minus و ضرب کارتزین باشد. آیا قابل عملیات ذخیره سازی(درج- حذف – به هنگاه سازی) هست ؟

مزایا و رعایت مفهوم دید خارجی:

معایب :

۱. فزون کاری(Overhead) به خاطر لزوم انجام تبدیلE/C (View Computation)

۲- دشواری درعملیات ذخیره سازی (عدم امکان انجام تبدیل E/C در مواردی)

راه حل مشکل ۱ در عمل و تئوری:

۱- عدم استفاده از مفهوم VIEW (اگر امکان پذیر باشد)

۲- تکنیک : استفاده از تکنیک Materialized View (دید ساخته شده یا ذخیره شده Stored view) یعنی دید را چنان تعریف می کنیم که سیستم نتیجه Clause SELECT آن را ذخیره کند. یعنی تبدیل را یک بار انجام دهد تا هر بار که کاربر نیاز به آن دید را داشته باشد ، جدولش آماده باشد. این یکی از تکنیک هایی است که Performance را بالا می برد.

نکته: همه گونه های دید قابل عملیات به هنگاه سازی نیستند.

کنجکاوی: در SQL این نوع دید را با کدام دستور باید تعریف کنیم؟

شرایط استفاده از Materialized View :

- نرخ عملیات ذخیره سازی در جدول های مبنای زیرین پایین باشد یعنی داده های لایه زیرین یا اصلا تغییر نکند یا تغییرات کم باشد، یعنی یا باید جداول زیرین جدول آرشیو باشند یعنی اصلا تغییر نکنند یا خیلی ACIVE نباشد زیرا در غیر این صورت دید ساخته شده هم باید مرتبا update بشود.
- فرکانس اجرای برنامه کاربر بالا و سرعت مد نظر باشد.

مزیت Materialized View افزایش سرعت دربازیابی.

عیب:

- مصرف حافظه (از مصادیق افزونگی) (چون stored است.)
- خطر بالقوه ناسازگاری داده ها (لزوم حفظ سازگاری داده ها) (چون افزونگی داریم)

مزایای مفهوم دید :

۱. تأمین کننده محیط فرافایلی (انتزاعی) برای کاربران
۲. تأمین کننده اشتراک داده (داد ها یک جا ذخیره می شوند، کاربران مختلف هر کدام بر اساس نیازشان از آن استفاده می کنند)
۳. مکانیزم اتوماتیک امنیت داده ها است.

زیرا کاربر خارج از محدوده داده ای خود هیچ نمی بیند.

داده نهان (hidden data) برای کاربر خود به خود ایمن است.

٤. تسهیل برنامه سازی:
   - کاربر با ساختار داده کوچکتر در حد نیاز خود کار می کند.(درگیرهمه جدول ها نیست)
   - نوعی امکان "ماکرونویسی" است. قطعه برنامه ای که یک بار نوشته می شود. Stored شده. هر وقت آن را CALL کنی در بدنه برنامه COPY می شود.
٥. تأمین کننده استقلال داده ای (DI = Data Independence)
٦. کنجکاوی؟؟؟

ابتدای صدای ۲ جلسه پنجم

تعریف DI و نکات مربوطه: Data Independence

- اصطلاح استقلال داده ای یک اصطلاح تئوریک است که در عمل به آن جدایی برنامه ها از داده ها می گویند.

تعریف : مصونیت برنامه های کاربران (سطح خارجی) در قبال تغییرات در سطوح زیرین معماری DB .

سطوح زیرین :

- سطح ادراکی (شمای ادراکی)
- سطح داخلی (شمای داخلی)

DI می گوید اگر لایه های زیرین (شمای داخلی و ادراکی) تغییر کند ، شمای خارجی و برنامه ها تغییر نکنند.

نتیجه مهم تامین DI صرفه جویی در هزینه سازمان (هزینه تولید – نگهداری – سیستم های کاربردی)

انوع DI :

۱. logical : LDI
۲. physical : PDI

تعریف PDI : مصونیت برنامه های کاربران در قبال تغییرات در شمای داخلی (سطح داخلی–در فایلینگ پایگاه)

نکته PDI در سیستم های رابطه ای به بعد به ویژه سیستم های جدیدتر ٪۱۰۰ تأمین است. به کمک مفهوم دید خارجی زیرا کاربر دارای دید اساسا در محیط فرافایلی کار می کند و برنامه هایش در گیر جنبه های فایلینگ پایگاه نیستند. بنابراین درقبال تغییر در فایلینگ دید کاربر تغییر نمی کند و برنامه هایش کماکان اجرا می شوند.

مثال هایی از تغییر در فایلینگ: تقریبا اکثر جنبه های فایلینگ پایگاه می توانند تغییر کنند توسط خود DBMS یا به درخواست Admin . به عنوان مثال طول رکورد داخلی – تعداد فیلد ها – ساختار فایل – رسانه ذخیره سازی می تواند عوض شود – یک فایل به یک یا دو فایل تقسیم شود.

تعریف LDI : مصونیت برنامه کاربران در قبال تغییرات در شمای ادراکی (سطح ادراکی). یعنی اگر شمای ادراکی تغییر کند برنامه ها تغییر نکنند.

– تغییرات در سطح ادراکی کدامند؟

۱. رشد DB درسطح ادراکی (Database Growth)

۲. سازمان دهی مجدد سطح ادراکی[تغییرات طراحی منطقی]

توجه : معمولا تغییرات در سطح ادراکی در سطوح زیرین (داخلی – فیزیکی) منعکس می شود.

نکته: با این تغییرات با محتوایی داده ای DB کم نمی شود.

دلیل رشد DB :

- مطرح شدن نیاز های جدید برای کاربران.

اگر در سطح مدلسازی دیده شوند اضافه شدن منجر می شود به اضافه شدن :

۱. صفت (صفات) جدید

۲. نوع موجودیت های جدید

۳. نوع ارتباط های جدید

ولی اگر در سطح طراحی منطقی دیده شوند با فرض TDS ، منجر می شود به اضافه شدن:

۱. ستون های جدید به جدول(هایی)

۲. ایجاد جدول(های) جدید

نکته: در قبال این نوع تغییر LDI صددرصد تامین است. با همان مفهوم دید خارجی(view). چون کاربر دارای دید، خارج از محدوده دید خود هیچ نمی بیند . بنابراین اطلاعی از ستون ها و جدول های اضافه شده ندارد. البته کاربر ممکن است میزانی افت کارآیی در برنامه اش حس کند(سرعت اجرا کم می شود).

مثال: فرض می کنیم TABLE مبنای S را داریم و کاربرانی روی این جدول دیدهایی دارند

نیاز جدید در حد اضافه شدن یک صفت (شماره حساب).

برای پاسخ دهی به نیاز جدید، جدول S باید گسترش داده شود.

این گسترش هیچ تاثیری در دید های موجود ندارد. برنامه هم اجرا می شود.

در SQL :

```
ALTER TABLE S

      ADD SUPNUM CHAR (15)
```

توجه داشته باشید که در ALTER TABLE نمی توان NOT NULL زد چون دیفالت آن از نظر سیستم NULL است.

با اجرای این دستور کاتالوگ متناسباً update می شود:

در جدول مشخصات جدول ها تعداد ستون جدول S یکی اضافه می شود.

در جدول مشخصات ستون ها یک سطر جدید اضافه می شود. (مشخصات ستون جدید)

کنجکاوی : درسطح فایلینگ این گسترش چگونه انجام می شود؟

کمترین ALTER را باید انجام دهیم زیرا کارهای سطح فایلینگ خیلی سخت است.

جواب:

۱. گسترش ایستا: با هر ALTER فایل جدید تعریف شود داده ها منتقل شوند. فایل قدیم حذف شود.
۲. گسترش پویا: در این روش سیستم به تدریج این گسترش را پیاده سازی می کند. به این معنا که هر بار کاربر در برنامه اش به هر دلیلی به سطری دست یابی پیدا کند و عملی انجام بدهد که باز نویسی(rewrite) لازم داشته باشد، (مثل عمل های delete, update, insert) سیستم در مرحله بازنویسی، رکورد جدید را گسترش می دهد. در واقع رکورد را با طول متغییر ذخیره کرده و با آن کار می کند.

۴۳:۳۰

تمرین: فرض می کنیم در SQL دستور ALTER TABLE وجود نداشته باشد. آن را شبیه سازی کنید.

نکته : LDI درقبال رشد پایگاه صد در صد تأمین است .

جنبه ی دیگر تغییر:

سازمان دهی مجدد سطح ادراکی

LDI در قبال ساز ماندهی مجدد صد در صد تأمین نیست . تا حد زیادی به کمک مفهوم دید تأمین است. در این نوع تغییر طراحی منطقی DB عوض می شود و درنتیجه شمای ادراکی هم تغییر می کند.

مثال : فرض می کنیم طراح جدول S را به دو جدول مبنای جدید SX و SY تجزیه عمودی کرد باشد. (در طراحی جدید دو جدول مبنا داریم). کاربرانی روی جدول S دید های دارند.

S:

S#	….	City

SX:

S#	Sname	City
S1		
S2		

SY:

S#	…	Status
S1		
S2		

نکته: کلید باید روی هر دو جدول باشد.

کنجکاوی : دلایل این تجزیه چیست ؟

۱- تأمین فایلینگ کاراتر ← سرعت بیشتر

۲- نرمال تر سازی رابطه ها

۳- توزیع داده ها در سایت ها Vertical Fregmentation

٤- اعمال محدودیت DBMS (از نظر تعداد ستون ها). Package تعداد محدودی ستون در جداول می پذیرد.

٥- کاهش حجم NULL VALUE

٦- کنجکاوی

معایب :

۱. عیب اصلی در بازیابی است. نیاز به انجام عمل پیوند برای بازیابی اطلاعات دارد.

۲. کلید باید تکرار شود پس نوعی افزونگی ایجاد می شود.

درشمای جدید: (مراحل تجزیه)

```
1) CREATE table SX
…
2) CREATE table SX
…
3) INSERT INTO SX SELECT S# , Sname , City from S;
4) INSERT INTO SY SELECT S# , Status from S;
```

نکته : به مراحل ۳ و ٤ در تئوری مهاجرت داده ها گفته می شود.(Data Migration)

6) `DROP TABLE S`

با حذف جدول مبنای دیدهای S دید های قبلا تعریف شده نا معتبر می شوند و در نتیجه برنامه های عمل کننده در آن دیدها دیگر اجرا نمی شوند LDI تأمین نیست. مگر این که طراح .پیاده ساز تدبیری بیندیشد.

آن تدبیر عبارت است از استفاده از مفهوم دید و امکان تعریف دیده روی دید، به این ترتیب که جدولی با نام و ساختار S این بار به صورت یک دید، روی دو جدول SX و SY تعریف می کنیم با مکانیسم پیوند.

از نظر تئوری دید S دید پیوندی Pk-Pk است به شرح دیده شده و مشکلی درعملیات ذخیره سازی ندارد.

```
5) CREATE VIEW S
AS SELECT S# , …
FROM SX JOIN SY
```

تأکید : با این تدبیر چون می توان روی دید، دید تعریف کرد، دید های قبلا تعریف شده روی S معتبر می مانند. برنامه های کاربران کماکان اجرا می شود. اما به قیمت کاهش سرعت زیرا تبدیل E/E هم لازم می شود.

تاکید : از تکنیک دید روی دید وقتی استفاده می کنیم که بخواهیم LDI را در قبال سازماندهی مجدد تأمین کنیم.

سئوال: چرا LDI در قبال سازماندهی مجدد صد در صد تأمین نیست؟

چون بعضی دیدها در عملیات ذخیره سازی مشکل دارند. دراین مثال جدول جدید یک دید پیوندی PK-PK است و در تئوری مشکل ندارد ولی همیشه که چنین نیست

کاربرد دیگر تعریف دید روی دید:

تقویت بیشتر امنیت داده ها دراین حالت هیچ اطلاعی از وجود Stored Data ندارند. در وضعیت قبلی کاربرای دارای دید V1 تا Vi به هر حال از وجود جدول S آگاهند. در حالت جدید هیچ اطلاعی از وجود جدول های ذخیره شده ندارند.

ابتدای صدای ۳ جلسه پنجم

مفاهیم اساسی مدل رابطه ای Relational Data Model (RDM:RM)

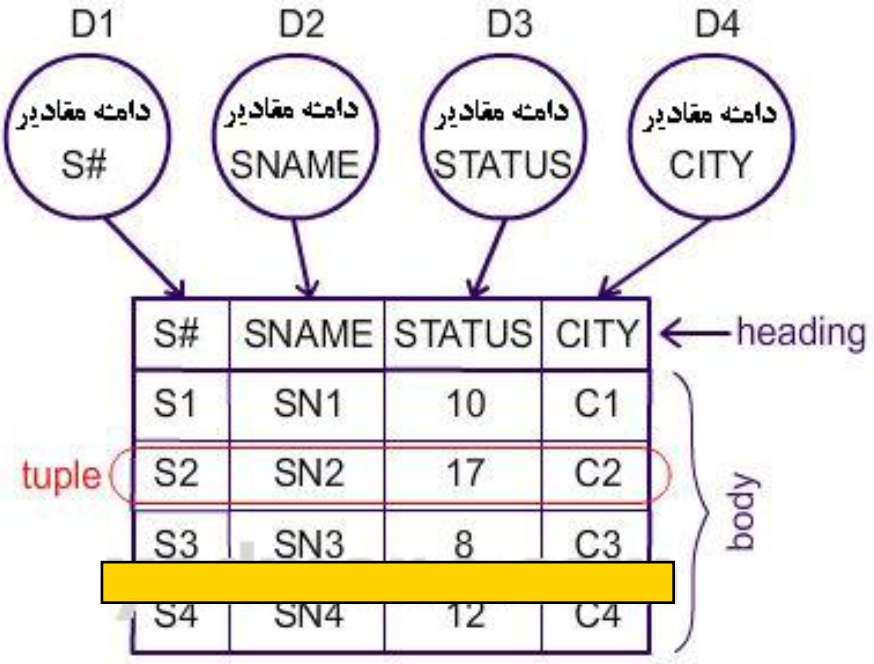
مبنای سیستم ها ی رابطه ای (جدولی)

تعریف رابطه: Relation

تعریف ۱: رابطه یک زیر مجموعه است از ضرب کارتزین n مجموعه (تعریف عام در ریاضیات)

تعریف ۲: با فرض وجود n مجموعه از مقادیر Dn , ..., D3 , D2 , D1 موسوم به دامنه، رابطه R با صفات R(A1 ,A2 , ...,An) تعریف شده روی این n دامنه مجموعه ای است از عناصری به صورت: <di1,di2,...,din> و i=1,... موسوم به n-taple [تاپل] به نحوی که di1 عضو D1 و di2 عضو D2 و ....

مثال:



تعریف ۳: (مبنای بحث ها ی بعدی)

با فرض وجود n مجموعه از مقادیر [هم نوع] Dn , ..., D3 , D2 , D1 موسوم به دامنه نه لزوماتمایز، رابطه R تعریف شده روی این دامنه، دو مجموعه است:

۱- heading (عنوان رابطه – چکیده رابطه)

۲- body (بدنه - بسط - گسترده)

Heading : مجموعه ای است نام دار از اسامی صفات.

R(A1 , A2 , …,An) : HR یا R(H)

S(S#,Sname,,,,)
P(P#,…)
SP(S#,P#,QTY)

نکات:

۱- **درجه رابطه** : کاردینالیتی مجموعه H یا کاردینالیتی عنوان (تعداد صفات در H).

اگر n درجه رابطه R باشد داریم $n \geq 0$ (درجه می تواند تهی هم باشد)

۲- عنوان رابطه intension (چکیده-خلاصه-فشرده) رابطه هم گفته می شود.

۳- همین عنوان رابطه است که باید به سیستم رابطه ای معرفی کنیم. مثلا با دستور Create Relation . در عمل Create Table می دهیم.

٤- هر رابطه یک معنا دارد. این معنای رابطه در خود عنوان رابطه است. بیانگر یک واقعیت کلی از محیط

مثال: وقتی می نویسیم :

S(S#,…..,City)

معنایش این است : وجود دارد در خرد جهان واقع (محیط) نوع موجودیتی با نام S و با صفات (S#,…..,City)

Body : مجموعه ای است از تاپل ها (همان مجموعه دیده شده در تعریف دوم)

نکات:

۱- بدنه رابطه در ابتدای تعریف تهی است یعنی تاپلی در آن نیست

۲- کاردینالیتی رابطه همان کاردینالیتی بدنه است. به بیان دیگر تعداد تاپل ها که متغیر در زمان است

۳- به بدنه رابطه Extension (بسط-گسترده-رابطه) نیز می گویند.

تناظر بین مفاهیم رابطه ای و اصطلاحات جدولی:

مفهوم رابطه ای	اصطلاح جدولی
رابطه	جدول
دامنه	مجموعه مقادیر مجاز ستون
تاپل	سطر
صفت	ستون
درجه	تعداد ستون ها
کاردینالیتی	تعداد سطر ها
کلید	؟

تفاوت رابطه و جدول : جدول تنها یک امکان است برای نمایش مفهوم ریاضی رابطه و حداقل ۸ تفاوت با رابطه دارد : رجوع شود به برگ رابطه

+ اصطلاح جدولی کلید را گاهی شناسه سطر (Row ID)می گویند. (کلید در مدل رابطه ای یک مفهوم ریاضی است.)

۲۹:۳۰

ویژگی های رابطه: (٤ ویژگی مهم دارد)

۱- رابطه تاپل تکراری ندارد. دلیل : بدنه مجموعه است و مجموعه عنصر تکراری ندارد.

۲- صفات در عنوان نظم مکانی ندارد دلیل : عنوان مجموعه است:

$$R(A,B) \sim R(B,A)$$

حال آن که در عمل می توانیم برای ستون ها ی جدول نظم مکانی قائل شویم . بنابراین در عمل دو امکان برای ارجاع به ستون ها داریم:

By Name
By Position Number

اما درتئوری تنها امکان ارجاع به صفت در رابطه نام صفت است. (By Name)

۳- تاپل ها نظم ندارند. (حسب مقادیر یک یا چند صفت) زیرا بدنه مجموعه است.

نکته مهم : از ویژگی ۲و۳ نتیجه می گیریم که در مدل رابطه ای یک مفهوم اساسی (محوری) فایلینگ، نه مطرح است

نه لازم. آن مفهوم نظم است.

نکته : اساسا در مدل رابطه ای مفاهیم ریاضی (انتزاعی) وجود دارد بنابراین اصطلاحات و مفاهیم فایلینگ، نباید در سطح مدل مطرح باشد. بنابراین در مدل رابطه ای این مفاهیم مطرح نیستند:{ ۱- نظم ۲ -کلید ۳- آدرس ٤- اشاره گر ٥- بلاک ٦- رکورد ۷- فیلد ....}

٤- تمام صفات رابطه تک مقداری هستند (رجوع کنید به مفهوم رابطه نرمال) یعنی به ازای نام هر صفت در هر تاپل فقط یک مقدار داریم و نه بیشتر.

سه ویژگی اول دلیل ریاضی دارد اما ویژگی چهارم دلیل ریاضی ندارد. بلکه دلیل تکنیکی دارد (در بحث رابطه نرمال خواهیم دید.)

نکته: این چهار ویژگی چهار فرق اساسی بین مفهوم ریاضی رابطه و اصطلاح جدول هستند. اما در مدل رابطه ای باید باشد . البته از دیدگاه ریاضی مفهوم چهارم الزامی مدل رابطه ای یا مفهوم ریاضی رابطه نیست اما در مدل رابطه ای باید باشد.

تفاوت های دیگر:

۱. رابطه می تواند از درجه صفر باشد اما جدول نمی تواند.
۲. رابطه نمی تواند NUIL VALUE داشته باشد ولی در جدول اجتناب ناپذیر است.
۳. تفاوت در طرز نمایش رابطه با جدول
٤. تفاوت در طرز نمایش تاپل باسطر
٥. رابطه می تواند بیش از دو بعد داشته باشد در حالی که جدول دو بعدی است.

جلسه ششم دوشنبه ۲۹/٤/۱۳۸۸ ساعت ۷:۳۰

مفهوم دامنه و نکات آن

تعریف : مجموعه ای است نامدار از مقادیر هم نوع که حداقل یک صفت از رابطه از آن نوع، معنا، مقدار، محدودیت می گیرد.

معادل است بامفهوم نوع داده(Data Tape) درتئوری انواع

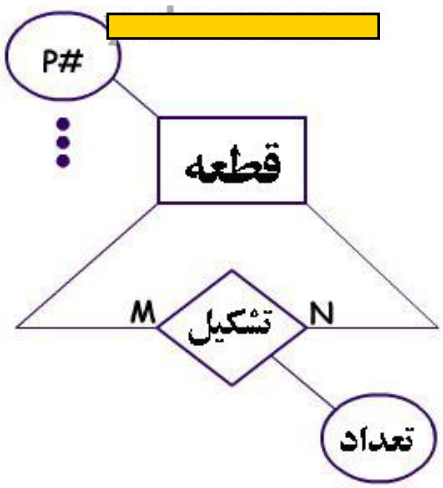
- برای تعریف یک رابطه ابتدا باید دامنه هایش را تعریف کرد.
- دامنه هایی که یک رابطه روی آن ها تعریف می شود لزوما متمایز نیستند.

بیان ریاضی: اگر Ai عضو HR و Aj عضو HR دو صفت متمایز از R باشند ، لزوما نداریم:

Di != DJ

مثال: قطعه از قطعاتی تشکیل شده است:

P(P#,……)
P-P(MagP#,MinP#,Q)



- MagP# شماره قطعه اصلی و MinP# شماره قطعه فرعی است . دامنه مقادیر هر دو یکی است.
- توجه داریم که رابطه P-P (P-P نام رابطه است) درجه اش از 3 است اما تعداد دامنه ها 2است.
- اگر n درجه رابطه R و m تعداد دامنه های R باشد داریم:

$m \leq n$

- در اینجا ما دو شماره قطعه داریم . حداقل یکی از آنها باید دگر نامی شود. چون heading مجموعه است و در مجموعه نمی تواند نام تکراریداشته باشیم.

مثال از شمای پایگاه رابطه ای (مدل تئوریک) [در SQL استاندارد] :

تأکید : برای تعریف رابطه ابتدا باید دامنه هایش را تعریف کرد.

دامنه های لازم برای تعریف پایگاه رابطه ای SP :

```
CREATE DOMAIN SN char(6) DEFAULT  '..??...'

CREATE DOMAIN SNam ……

CREATE DOMAIN STA …….

CREATE DOMAIN CITY      ……..

CREATE DOMAIN PN  ……..

……

CREATE DOMAIN Q small int ….
```

حال تعریف رابطه ها:

```
CREATE RELATION S
      (S#         DOMAIN      SN,
      Sname DOMAIN  SNam ,
      Status      Domain      STA,
      City        Domain      CITY)
      PRIMARY KEY S#;


CREATE RELATION P

….

CREATE RELATION SP

……..
```

در حاشیه : در SQL همچنین داریم:

```
CREATE DOMAIN

ALTER DOMAIN

DROP DOMAIN
```

مزایا مفهوم دامنه:

۱. کاهش اندازه شما

۲. امکانی است برای کنترل مقداری پرسش ها (یعنی اگر برنامه ساز بخواهد مقداری برای صفتی در پایگاه ایجاد کند، خارج از طیف مقادیر دامنه اش، سیستم آن درخواست را رد می کند.

۳. دگر نامی صفات را تسهیل می کند .(صفات هم دامنه هر چه باشد نام)

٤. اعمال برخی محدودیت های جامیعتی را تسهیل می کند (مشخصا صفت می گیرد از دامنه اش، محدودیت را)

٥. امکانی است برای کنترل معنایی پرسش (QUERY)ها:

اگر کاربر به هر دلیلی پرسشی به سیستم بدهدکه فاقد معنا باشد (مثلا درآن پرسش دو صفت را با هم مقایسه کند که از نظر معنا قابل مقایسه نباشند) اگر سیستم دامنه آن صفات را بشناسد می تواند بی معنا بودن آن پرسش را تشخیص دهد، چون صفات هم دامنه قابل مقایسه اند یا صفاتی که دامنه های آن ها را به صورت دامنه های مقایسه شدنی تعریف کرده باشیم.

مثال:

```
S(S#,…)

P(P#,…..,W ,…..)

SP(S#,P#,QTY)
```

فرض می کنیم w (وزن) و QTY (وزن) هر دو عدد صحیح int باشند. شماره قطعاتی را بدهید که وزن آنها مساوی تعداد تهیه شده از آنها باشد. هر چند هر دو int هستند اما در معنا قابل مقایسه نیستند.

حداقل انتظار از سیستم این است که به کاربر اخطار بدهد که پرسش بی معنا است.

کنجکاوی : دامنه چه مزایای دیگر دارد ؟

۳۶:۰۰

تعریف رابطه نرمال:

رابطه نرمال (NR) و رابطه غیر نرمال (NNR یا N2R)

تعریف رابطه نرمال: رابطه است که تمام صفات آن تک مقداری باشند.

توجه : این تعریف غلط است : رابطه نرمال رابطه ای است که تمام صفات آن اتمیک (ساده و تجزیه نشدنی) باشند.

خاصیت تک مقداری با خاصیت atomicity فرق دارد.

صفت ساده یا اتمیک صفتی است تک معنایی به این معنا که اگر مقدارش را به اجزا تجزیه کنیم، اجزا در همان کاربرد و در همان حیطه معنایی مقادیر صفتی دیگر نباشند به بیان ساده تر اجزای آن معنا نداشته باشند.

رابطه غیر نرمال: رابطه ی است که حداقل یک صفت آن چند مقداری باشد. به چنین رابطه ای رابطه تو در تو هم گفته می شود.(Nested Relation) زیرا حداقل یکی از صفات آن خود از نوع رابطه است . در بیان عملی یعنی جدول در جدول.

مثال:

NNSP:

	PQTY	
S#	P#	QTY
S1	P1	10
	P2	90
	P3	80
S2	P1	80
	P2	90
S3	P1	70

tuple

رابطه شکل بالا غیر نرمال است زیرا صفت PQTY چند مقداری است. ولی غیر نرمال بودن ربطی به مرکب بودن ندارد بلکه چه ساده باشد چه مرکب بایستی چند مقداری نباشد.

مثال دیگر: یک شخص چند تلفن دارد. تلفن صفت چند مقداری است: (مرکب هم نیست)

PERTEL:

NC	PHONE
n1	N1
	N2
	N3
n2	N1
	N2

تبدیل رابطه غیر نرمال SP به رابطه نرمال یا رابطه هنجار:

برای تبدیل به نرمال هر صفت را به عنوان یک تاپل تک مقداری در نظر گرفته و کلید ها را تکرار می کنیم:

NSP:

S#	P#	QTY
S1	P1	10
S1	P2	90
S1	P3	80
S2	P1	80
S2	P2	90
S3	P1	70

سئوال: چرا در مدل رابطه ای ، رابطه باید نرمال باشد؟

دلیل نرمال بودن رابطه در مدل رابطه ای سادگی (Simplicity) است.

سادگی در :

۱. نمایش جدولی رابطه

۲. در زبان پایگاهی [DSL] ← دستور های DD و DC و DM

۳. اجرای دستور ها

مثال : پرسش های زیر را در نظر می گیریم:

۱- درج کن I1 : (اطلاع) : <S4,P1,40> را

۲- درج کن I2 : (اطلاع) : <S2,P3,30> را

درج اطلاع I1 در هر دو رابطه NNSP و SP با همان دستور ساده "درج کن تاپل را در رابطه" انجام می شود. یعنی می توان نوشت:

```
INSERT INTO NNSP
      TUPLE <S4,P1,40>
```

و

```
INSERT INTO SP
      TUPLE <S4,P1,40>
```

زیرا I1 برای هر دو رابطه ساختار تاپلی دارد:

- از تک مقدار ها تشکیل شده.
- مقدار کلید را دارد. (وقتی مقدار کلید را داشته باشد یعنی تاپل)

درج اطلاع I2 در رابطه نرمال SP کماکان با همان دستور ساده "درج کن تاپل را در رابطه" انجام می شود. یعنی می توان نوشت:

```
INSERT INTO SP
      TUPLE <S2,P3,30>
```

اما نمی توان نوشت :

```
INSERT INTO NNSP
      TUPLE <S2,P3,30>
```

زیرا I2 برای رابطه NNSP ساختار تاپلی ندارد، زیرا از قبل تاپلی با کلید S2 در NNSP داریم.

توجه : برای درج I2 در NNSP دستور دیگری لازم است که سادگی دستور "درج کن تاپل را" ندارد.

ما یک دستوری نیاز داریم که عنصر <P3,30> را بگیرد و آن را اضافه کند به یک مجموعه که خود مقداری است از PQTY متناظر با S2 .

نکته : با این همه رابطه غیر نرمال هم مزایایی دارد. در جدول زیر مزایا و معایب این دو نوع رابطه آمده است.

نوع رابطه	مزایا	معایب
نرمال	۱- سادگی در: ۱. نمایش جدولی رابطه ۲. در زبان پایگاهی ۳. اجرای دستور ها ۲- خاصیت تقارن صفات	۱- افزونگی ۲- طولانی شدن ۳- دشواری در نمایش نوع داده پیچیده Complex Data Type ٤- دشواری در نمایش مفهوم وراثت ٥- دشواری در نمایش ارتباط سلسله مراتبی
غیر نرمال	عکس معایب رابطه نرمال	۱- عدم سادگی در: ۱. نمایش جدولی رابطه ۲. در زبان پایگاهی ۳. اجرای دستور ها ۲- عدم خاصیت تقارن صفات

توضیح در رابطه با خاصیت تقارن صفات:

خاصیت تقارن صفات یعنی می توان از هر صفتی مستقلا در where clause استفاده کرد در مقایسه های تک مقداری :

Where Attribute theta 'rule' → theta : = , > , <

تقارن صفات یعنی هیچ صفتی به صفت دیگر از نظر مقایسه های تک مقداری اولویت و ارجحیت ندارد.

به این معنا نیست که جواب تک تاپل باشد مگر اینکه کلید باشد.

توضیح در رابطه با افزونگی : در اینجا منظور افزونگی فیزیکی نیست بلکه منظور افزونگی ادراکی یا منطقی است یعنی پدید آمده در سطح طراحی منطقی، ممکن است منجر به افزونگی فیزیکی بشود و ممکن است نشود. بستگی دارد به تناظر بین رابطه ها و فایل ها اگر این تناظر یک به یک باشد یعنی یک رابطه را در یک فایل ذخیره کنیم که در این صورت معمولا هر تاپل به صورت یک رکورد ذخیره می شود ، افزونگی فیزیکی خواهیم داشت.

تمرین : ساختار فایلی برای رابطه SP پیشنهاد کنید که افزونگی فیزیکی نداشته باشد.

تمرین : بررسی کنید در یک DBMS تصویر Image Type چگونه نمایش داده می شود؟

نکته در مورد طولانی شدن کلید: وقتی تبدیل به نرمال می کنیم حداقل یک صفت دیگر جزو کلید می شود که این باعث کاهش کارآیی مصرف حافظه و پایین آمدن سرعت می شود.

ابتدای صدای ۲ جلسه ٦

کلید در مدل رابطه ای (RM):

یک مفهوم عام است . گونه هایی دارد :

۱. سوپر کلید (ابر کلید) SK(Super Key)
۲. کلید کاندید (داوطلب نامزد) (نامزد کلیدی) CK(Candidate Key)
۳. کلید اصلی PK(Primary Key)
٤. کلید بدیل (دیگر) AK(Alternate Key)
٥. کلید خارجی FK(Foreign Key)

تعریف کلید ها

مفروض است رابطه :

R(A1,A2,…..,Am)

۱- **تعریف SK** : هر زیر مجموعه S از HR که خاصیت یکتایی مقدار داشته باشد.

نکات:

نقش SK :

- تضمین می کند عملیات تاپلی را (اما نه لزوما با کمترین تعداد صفات)
- در عمل با unique clause به سیستم معرفی می شود.
- SK فقط یک محدودیت دارد ← یکتایی مقدار.
- در تعریف بعضی فرم های نرمال کاربر دارد.

۲- **تعریف CK** : هر زیر مجموعه K از HR که دو خاصیت زیر را داشته باشد:

۱- یکتایی مقدار : در هیچ دو تاپل مقدارش یکی نیست

۲- کاهش ناپذیری(Irreducibility) یا کِهینگی(minimality): یعنی هیچ زیر مجموعه از K خود یکتایی مقدار نداشته باشد به بیان دیگر اگر یک صفت از آن زیر مجموعه خارج کنیم آن چه باقی می ماند یکتایی مقدار نداشته باشد.

مثال:

نام رابطه	CK
S	S#
P	P#
SP	(S#,P#)

در رابطه SP هر کدام از S# یا P# را که از CK خارج کنیم دیگر یکتایی مقدار ندارد.

نکات :

- هر CK ، SK هم هست عکس مطلب درست نیست.
- هر رابطه حداقل یک CK دارد. زیرا در بدترین حالت، ترکیب تمام صفات (خود H) CK می شود. زیرا بدنه مجموعه است و تاپل تکراری ندارد.
- در رابطه ای که در آن خود HR ، CK باشد رابطه تمام کلید (All Key) می گویند.
- اگر N تعداد CK های R باشد داریم : N>=1
- CK های R می توانند هم پوشا باشند. یعنی صفت یا صفات مشترک داشته باشند.

مثال:

```
EMP (E#,J#,NC,name,duration)
```

P# : شماره کارگزینی

J# : شماره پروژه

NC : کد ملی

Duration : مدت

یک کارمند در بیش از یک پروژه کارکرده است. سایر ویژگی های این رابطه فعلا مد نظر نیست با این تعریف کدام صفت ها می توانند کلید باشند :

```
(E#,J#,NC,name,duration)
```

```
(E#,J#,NC,name,duration)
```

E# به تنهایی unique نیست.

NC به تنهایی unique نیست.

کلید یک مقدار معنایی است و بنا به کاربرد و قواعد معنایی محیط مشخص می شود.

نقش CK : تضمین کننده عملیات تاپلی با کمترین تعداد صفات ممکن (یعنی با کاهش ناپذیری) به بیان دیگر امکان ارجاع است به تک تاپل با کمترین صفات

مثال : این رابطه چند SK دارد؟

```
EMP (E#,J#,NC,name,duration)
```

توضیح : هر CK خود SK هم هست. هر سوپر ست CK هم SK است. چون SK کاهش ناپذیری ندارد.

بنابر این در این مثال 2 تا CK داریم:

1- E# , J# 2- J#, NC

حال با افزودن صفت به هر کدام از CK ها SK های دیگری بدست می آید:

3- E# , J# , NC 4- E#,J#, duration
5- E# , J# , NC , duration 6- J#,NC, duration

مثال : رابطه زیر چند SK دارد؟

R(A,B,C)
SK: A A,B A,C A,B,C

کنجکاوی: یک رابطه حداکثر با درجه N چند CK و چند SK دارد؟

حداکثر تعداد SK ها : 2^n-1

حداکثر تعداد CK ها:

CK های تک صفتی ← n تا

CK های دو صفتی ← C^n2

.....

حداکثر در چه حالتی ...

۱- **تعریف PK** : یکی از CK ها به انتخاب طراح

.ضابطه انتخاب

a. کمترین تعداد صفات را داشته باشد. برای اینکه PK در فایلینگ کلید می شود و در سطح فایلینگ هرچه کوتاهتر باشد بهتر است.
b. مقدارش حتی الامکان تغییر نکند.
c. شناسه «رایج» نوع موجودیت درمحیط باشد.

مثال : رابطه مشخصات کارمند:

EMPH (EN,Ename,NC,…)

EN: شماره کارمندی

NC : کدملی

در اینجا شناسه رایج شماره کارمندی است و هنوز کد ملی در سازمان ها جا نیفتاده است.

دلایل لزوم PK :

نکته : در مدل رابطه ای اصالت با مفهوم CK است بنابر این این سئوال مطرح می شود PK چه لزومی دارد. از نظر تئوریک PK لزومی ندارد و همان مفهوم CK کفایت می کند.

دلایل:

۱- به دلیل تاریخی ← برای طراحان، پیاده سازان ، برنامه نویسان مفهوم کلید اصلی که از فایلینگ می آید مفهوم آشناتری است.

۲- سیستم ها با معرفی PK ، index اتوماتیک ایجاد می کنند.

۳- در اعمال قاعده C1 لازم می شود. (قاعده C1 :کلید اصلی نمی تواند NULL باشد).

سئوال : اگر PK تعریف نکنیم محدودیت هیچ مقدار ناپذیری روی کدام CK اعمال بشود؟

اگر بگوییم روی همه بسیار بازدارنده می شود مثلا ممکن است کد ملی کاربر در دسترس نباشد. بنابر این نمی توانیم دیتا وارد کنیم.

در عمل با کلاز Primary Key به سیستم داده می شود.

٤- تعریف AK : هر CK غیر از PK

اگر N تعداد AK های رابطه R باشد:

$N>=0$

در عمل: متناظر ندارد (پیاده سازی نشده است)

مثال:

CREATE TABLE
(A …
B…)
UNIQUE A:

کدام یک از کلید ها با این دستور معرفی نشده است ؟

AK معرفی نشده است

کلید های SK و CK و PK هر سه معرفی شده است اما محدودیت هیچ مقدار ناپذیری PK اعمال نمی شود و چنانچه آن محدودیت را در نظر بگیریم فقط SK و CK تعریف شده اند.

ابتدای صدای ۳ جلسه ٦

۵- تعریف FK

۱ - در عمل : ستون C از جدول T2 ، در جدول T2 کلید خارجی است هرگاه در جدول T1 کلید اصلی باشد.

۲- در تئوری : صفت Ai از R2 در R2 ، FK است هرگاه در R1 نه لزوماً متمایز از R2 ، CK باشد.

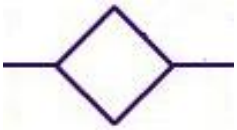
مثال:

نام رابطه	FK	دلیل CK در …
S	ندارد	-
P	ندارد	-
SP	P#	P
SP	S#	S

نکات :

- اگر N تعداد FK های رابطه ی R باشد: N>=0 (لزومی ندارد رابطه FK داشته باشد)

  یادآوری: در عمل با کلاز Foreign Key به سیستم معرفی می شود.
- نقش FK : امکان نمایش ارتباط (صریح) بین نوع موجودیت ها است درمدل رابطه ای.



مثال : وجود S# و P# در رابطه SP(S#,P#,QTY) به عنوان FK نمایشگر ارتباط با معنای "تهیه می کند-تهیه می شود"

است.

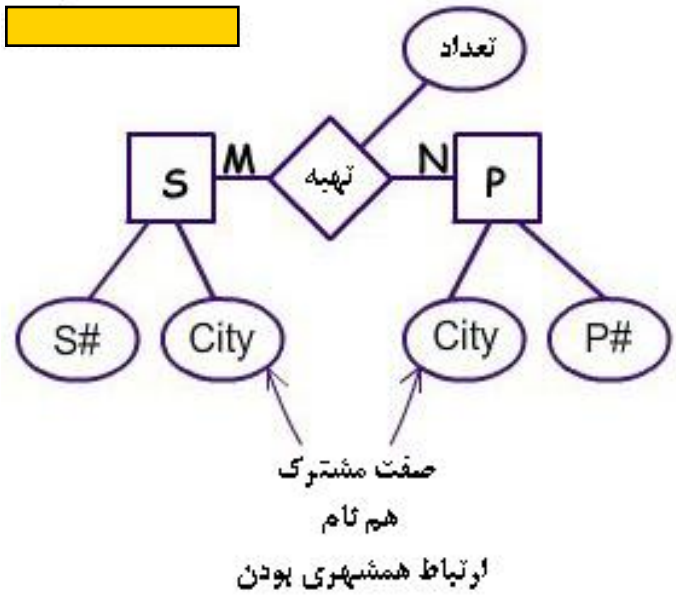
نکته : خواهیم دید FK های یک رابطه لزوماً همیشه جزء سازنده CK نیستند. (PK حالت خاص)

سئوال: آیا FK تنها امکان نمایش ارتباط است؟

ارتباط اگر صریح باشد : بله

ارتباط اگر ضمنی باشد : خیر

تعریف ارتباط ضمنی : ارتباطی است که در مدلسازی با صفت مشترک نمایش می دهیم (مشترک یعنی هم دامنه، اما در عمل هم نام)



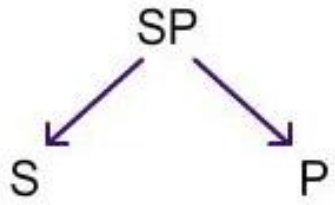
نکات FK بحث تمکیلی :

مفهوم ارجاع و گراف ارجاع :

- FK امکانی است برای ارجاع از رابطه ای به رابطه «دیگر»
- یک مقدار FK امکانی است برای ارجاع مقداری(Value Base Referencing) از تاپل(هایی)، از رابطه(هایی) به تاپلی از رابطه (هایی)
- ارجاع بین رابطه ها با گراف ارجاع نمایش داده می شود. در واقع با توجه به تعریف FK داریم:

رابطه مرجع R2 → R1 رابطه رجوع کننده

مثال: گراف ارجاع:

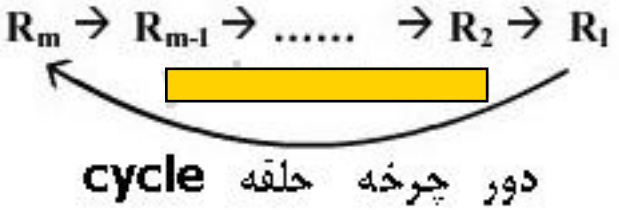


شکل کلی مسیر ارجاع:

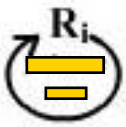
Relation Path:

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

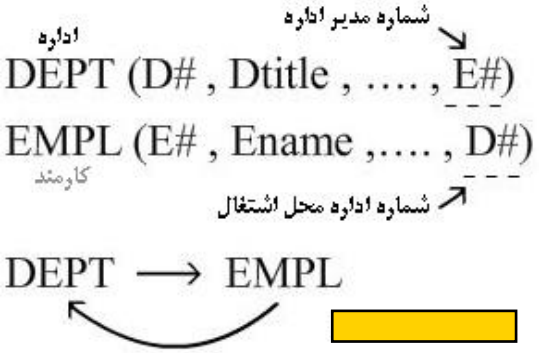
مسیر ارجاع می تواند چرخه ای یا حلقه ای باشد:



چرخه تکرار می تواند تک رابطه ای باشد: (رابطه به خود رجوع کننده یا Self Referencing)



مثال – چرخه ارجاع با دو رابطه:



تمرین عملی: همین DB را با یک package پیاده سازی کرده و داده وارد کنید. با چه مشکلی برخورد می کنید؟

مثال – چرخه ارجاع با یک رابطه:

کنجکاوی : این رابطه چه نکاتی دارد ؟

۱- مثالی است از حالتی که در آن R2 , R1 درتعریف FK لزوما متمایز نیستند.

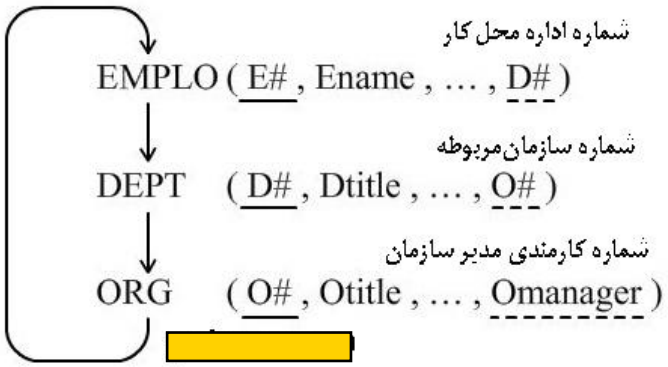
۲- چرخه ارجاع تک رابطه ای داریم. Self (به خود رجوع کننده)

۳- لزوم دگرنامی F_K (حتما باید نام دیگر داشته باشد)

٤- اگر N درجه این رابطه باشد تعداد دامنه هایش حداکثر N-1 است زیرا ممکن است صفات دیگری هم ، هم دامنه باشند.

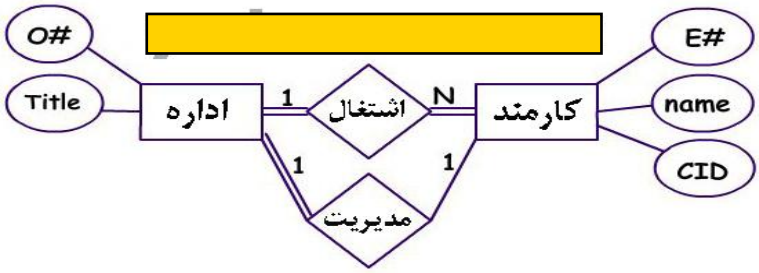
کنجکاوی : این رابطه در کدام فرم نرمال است؟

مثال – چرخه ارجاع با سه رابطه:



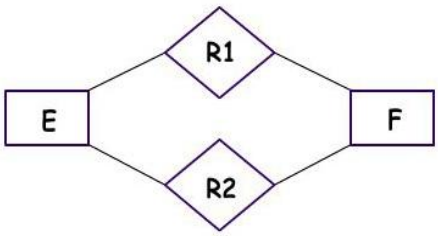
تمرین: نمودار ER هریک از این سه مثال را رسم کنید:

مثال ۱:



کنجکاوی: در چه وضعی چرخه ارجاع پدید می آید؟

باید به چندی ارتباط ها دقت کنیم .



فرض می کنیم دو رابطه داریم : حالات زیادی معروض است اما اگر هر دو M:N باشند چرخه پدید نمی آید. درحالات دیگر باید بررسی شود.

قواعد جامعیت در مدل رابطه ای: Integrity Rules (Integrity Constraint)

نکات مقدمه:

IR ها یا IC ها در RM

تعریف جامعیت : عبارت است از صحت، دقت، سازگاری و اعتبار داده های ذخیره شده در DB (در تمام لحظات)

- جامعیت از جنبه های کیفی داده است: Data Quality Features
- هر DBMS باید جامعیت DB را تضمین کند.(کنترل کند) با امکاناتی که دارد و بر اساس اطلاعاتی که کاربر به سیستم می دهد.

۱- منظور از کاربر در اینجا : طراح – پیاده ساز – تیم DBA

۲- منظور از اطلاعات همان قواعد جامعیت یا محدودیت هاست.

- چگونه اطلاعات داده می شود؟

معمولا با امکانات زبان پایگاهی

۱- بطور اعلانی Declaration ← با دستور های DD و DC

۲- بطور اجرایی ← با دستور های DD و DM (رویه ذخیره شده Stored Procedure در این بحث به نام Trigger (رهانا یا راه انداز))

نکته : هر Stored Procedure، Trigger نیست.

در RM دو دسته قاعده داریم :

۱- قواعد عام

۲- قواعد خاص

جلسه هفتم سه شنبه ۳۰/٤/۱۳۸۸ ساعت ۷:۳۰

قواعد جامعیت در RM :

- محدودیت ها ی خاص: وابسته به داده های محیط هستند: IC ها یا IR های تعریف شده توسط کاربر
- محدودیت ها ی عام : meta rules یا meta constraint . ناوابسته به داده های محیط هستند.

محدودیت ها ی خاص چهار دسته اند:

۱. محدودیت دامنه ای
۲. محدودیت صفتی یا ستونی
۳. محدودیت رابطه ای
٤. محدودیت پایگاهی

۱- **محدودیت های دامنه ای** : نوع و طیف مقادیر دامنه را مشخص می کند. درهمان دستور GREATE DOMAIN اعلان می شود.

۲- **محدودیت های صفتی:**

محدودیت دامنه ای روی صفت اعمال می شود.

صفت می تواند خود محدودیت هایی داشته باشد که ناقض محدودیت دامنه اش نباشد. مثل وابستگی تابعی بین صفات- وابستگی شمول و ...

سئوال: محدودیت صفتی چگونه اعمال می شود؟

- با دستور تعریف دامنه
- در همان دستور تعریف جدول
- مکانیزم اظهار (Assertion)
- مکانیزم رهانا (Trigger)

۳- **محدودیت رابطه ای** : ناظر است به تاپل های فقط یک رابطه (محدودیت درون رابطه ای نیز می گویند)

مثال : تهیه کنندگان ساکن 'C2' نمی توانند STATUS>15 داشته باشند(در رابطه S اعمال می شود).

محدودیت رابطه ای با مکانیسم اظهار و رهانا اعمال می شود.

٤- **محدودیت پایگاهی** : ناظر است به فایلهای بیش از یک رابطه که به نحوی با هم ارتباط معنایی دارند.

مثل رابطه های S , SP یا P , SP

نکته : محدودیت پایگاهی با مکانیسم اظهار اعلان می شود و با مکانیسم رهانا اجرایی می شود.

مثال : تهیه کنندگان ساکن 'C2' یا 'C7' با وضعیت کمتر از ۲۰ نمی توانند قطعه آبی رنگ با وزن بیش از ۱۰gr را به تعداد بیش از ۵۰ تهیه کنند.

فرض کنید می خواهیم عمل زیر را انجام دهیم:

```
INSERT INTO SP VALUES(S#,P7,70)
```

دستور رد می شود.

اظهار Assertion: امکانی است اعلانی برای اعلان قواعد جامعیت (صفتی، رابطه ای ، پایگاهی) .

(برای قواعد جامعیت دامنه ای نیست)

رهانا Trigger : امکانی است اجرایی برای اعمال قاعده یا محدودیت جامعیتی (صفتی، رابطه ای، پایگاهی) ماهیتا نوعی رویه ذخیره شده(Procedure) است که باید نوشته شود.

مبنای تئوریک رهانا مفهومی است به نام قاعده فعال (Active Rule) که در ADBMS ها مطرح است.

(Active DBMS). ساختاری به نام ECA دارد.

مثال:

```
If  EVENT on CONDITION then ACTION
```

یعنی اگر رویدادی بروز کرد، شرط برقرار بود ، عملی را انجام بده.

Event : رویداد

Condition : شرط

Action : اقدام – عمل

رویداد ها در SQL استاندارد:

۱. INSERT
۲. UPDATE
۳. DELETE

توجه : در عمل هر پکیجی از نظر ساختار Trigger ویژگی های خودش را دارد.

```
CREATE TRIGGER name
      {BEFORE | AFTER | INSTEAD OF}
      {INSERT | DELETE | UODATE OF column}
      ON tbl_name [ORDER value]
      [REFERENCING {NEW | OLD | NEW-TABLE | OLD-TABLE} AS
name]
      {WHEN Condition(n)}
      [SQL 3 Procedure                    → ACTION
      [FOR EACH {ROW | STATEMENT}]]
```

توضیح :

- با REFERENCING به نسخه قدیم و یا جدید جدول نام می دهیم. در عمل Update هم قدیم و هم جدید لازم است. در Insert فقط جدید و در Delete فقط قدیم، تا بتوانیم به سطر(ها) ارجاع بدهیم.
- SQL3 PROCEDURE همان ACTION است.
- FOR EACH ROW یعنی به ازای هر سطر از جدول که رویداد برای آن بروز کند اقدام انجام شود.
- FOR EACH STATEMENT یعنی به ازای هر دستور اقدام انجام شود. (سیستم دستور را به طور کامل اجرا می کند ، هرچه باشد تعداد سطر های دخیل و بازای هر سطر Action را انجام دهد.)

مثال ۱ : رابطه حاوی مشخصات کارمندان و میزان حقوق او را در نظر می گیریم. (مثالی از محدودیت رابطه ای) :

```
EMPREL(EMPID , EMPNAME , …. , EMPSAL)
```

محدودیت مورد نظر چنین است : "حقوق کارمند هیچگاه کاهش نمی یابد". رهانای زیر این محدودیت را اعمال می کند:

```
CREATE TRIGGER EMPREL-UPDATE-TRIG
      BEFORE UPDATE OF EMPSAL ON EMPREL
      REFERENCING OLD AS OEMPREL,NEW AS NEMPREL,
      (WHEN OEMPREL.EMPSAL > NEMPREL.EMPSAL
      SIGNAL.SQL State '7005'('salary can not be
decreased')
      FOR EACH ROW);
```

توضیح TRIGGR بالا: قبل از Update ستون حقوق از جدول داده شده نسخه قدیم OEMPREL را در نسخه جدید NEMPREL نامیدیم تا بتوانیم به ستون هایش ارجاع دهیم.

اگر مقدار حقوق در نسخه قدیم بیشتر از حقوق در نسخه جدید باشد یعنی حقوق کاهش یافته و باید Action اجرا شود. در اینجا ACTION فقط یک پیام است.

کنجکاوی: آیا به جای BEFORE در اینجا می توان AFTER نوشت.

مثال ۲: در این مثال فرض می کنیم علاوه بر EMPREL ، رابطه زیر را هم داریم: (مثالی است از محدودیت پایگاهی)

```
EMPSALAUG (EMPID , AUGM)
```

صفت AUGM مقدار افزایش حقوق کارمند است.

رهانای زیر سازگاری داده ای پایگاه داده را پس از عمل بهنگام سازی تضمین می کند.

```
CREATE TRIGGER DBUPDATETRIG
      AFTER UPDATE OF EMPSAL ON EMPREL
      REFERENCING OLD AS OEMPREL,NEW AS NEMPREL,
      (UPDATE EMPSALAUG
      SET AUGM = AUGM + NEMPREL.EMPSAL – OEMPREL.EMPSAL
      WHERE EMPSALAUG.EMPID = EMPREL.EMPID
      FOR EACH ROW);
```

تمرین در ارتباط با مثال ۲: می خواهیم سابقه افزایش حقوق را در DB داشته باشیم چه باید کرد.

توجه : مثال ۲ فقط آخرین افزایش را نگه می دارد و سابقه را نگه نمی دارد. برای حفظ سابقه باید ستون تاریخ به جدول اضافه شود(با ALTER TABLE) زیرا جدول EMPSALAUG کلیدش شماره کارمندی است و نمی توان برای یک کارمند بیش از یک سطر اضافه کرد. پس کلید ترکیب تاریخ و شماره کارمندی می شود.

مثال ٤ :

نکاتی که در مثال ٤ می بینیم :

۱. کاربرد INSTEAD OF را نشان می دهد.
۲. کاربرد رهانا در عملیات از دید در DB.

جدول دانشجو به شرح زیر مفروض است:

```
STT(STID,STname,STDEG,STMjr,STDEID)
```

دید V نیز به شکل زیر مفروض است:

```
CREATE VIEW V
      AS SELECT STID , STname,STDEG,STMjr,STDEID
      FROM STT
      WHERE STMJR = 'computer'
```

فرض کنید کاربر دستور زیر را می نویسد:

```
INSERT INTO V
 VALUES(999,ST9,'b9',D19)
```

در تبدیل E/C می شود:

```
INSERT INTO STT
      VALUES(999,ST9,'b9','NULL or Default Value',D19)
```

با فرمان زیر چه پیش می آید:

```
SELECT * FROM V;
```

کاربر تاپل جدیدی را که خودش ایجاد کرده را نمی بیند. چون شرط دید این بوده که رشته computer باشد. در صورتی که چون رشته جزو ستون های دید نیست هیچگاه رشته درج نمی شود.

برای رفع این مشکل از trigger استفاده می کنیم:

```
CREATE TRIGGER ITOOL
      INSTEAD OF INSERT ON V
      REFERENCING NEW ROW AS R,
      FOR EACH ROW
      INSERT INTO STT
      VALUES (R.STID,R.Stname,R.STDEG,'computer',R.STDEID)
```

در واقع به سیستم می گوییم به جای تبدیل E/C به طور معمول Action بالا بازای هر سطر انجام شود.

۱:۰۲:۰۰

قواعد جامعیت عام:

- قواعد جامعیت عام موسومند به محدودیت های C1 و C2 به این ها meta rules یا meta constraint نیز می گویند. ناوابسته به داده های محیط هستند. خاص داده های یک محیط نیست.

C_1 : قاعده جامعیت موجودیتی (Entity Integrity Rule): ناظر است به PK است. هیچ جزء تشکیل دهنده PK نمی تواند "هیچ مقدار" داشته باشد. مکانیسم اعمال C_1 ← معرفی PK به سیستم .

C_2 : قاعده جامعیت ارجاعی (Referential Integrity rule): ناظر است به FK است. اگر $R_2.A_i$ در R_2 ، FK باشد در R_1 ، CK است. در اینصورت A_i در R_2 باید مقدار قابل انطباق داشته باشد. یعنی آن مقدار در R_1 باشد. (Matchable Value). و یا در عمل NULL داشته باشد.

همانطوری که می بینید حداقل از دید آقای Date در تئوری رابطه نمی تواند NULL داشته باشد.

مثال:

S(S#,….) :
S1
S2
S3
SP(S#,P#,QTY)
S1,p1
…
S3,P2

تاپل زیر را در SP اضافه می کنیم:

<S5,P1,50>

این درج باید REJECT شود چون S5 مقدار قابل انطباق در S ندارد.

مکانیسم اعمال C2 :

۱. معرفی Fk (های) رابطه
۲. دادن گراف ارجاع
۳. مشخص کردن روش اعمال

مثال :

```
CREATE TABLE SP
(S# …
P#...
QTY…)
PRIMARY KEY (S# ,P#)
CHECK (0<=QTY<=500)
FOREIGN KEY S# REFERENCES S[.S#]
: ON DELETE OPTION
: ON UPDATE OPTION [cascade مثلا]
FOREIGN KEY P# REFERENCES P[.P#]
: ON DELETE OPTION
: ON UPDATE OPTION [cascade مثلا]
```

در مثال بالا آنجا که OPTION گذاشته ایم برای عمل حذف و بهنگام سازی است. عمل درج نیازی به مشخص کردن روش نیست. سیستم خود باید کنترل کند.

تست : روش اعمال C2 را در کدام عمل مشخص نمی کنیم.

۱- درج ۲- حذف و ...

مشخص کردن روش اعمال:

۱. روش cascade یا انتشاری : با حذف تاپل مرجع از رابطه مرجع تاپل هاب رجوع کننده در رابطه های رجوع کننده نیز حذف می شود.

مثال : اگر یک سطر از جدول S را حذف کنیم تمام سطر هایی که همان کلید را دارند باید از SP حذف شوند.

۲. روش Restrict : (روش منوط به یا مشروط یا تعویقی) در این روش درخواست حذف تاپل مرجع تا زمانی که تاپل های رجوع کننده به آن وجود دارد به تعویق می افتد. بنابراین اگر کاربر بخواهد عمل حذف مرجع انجام بشود ابتدا باید تاپل های رجوع کننده حذف بشوند. در صفحه I4 مثال ٥ همین نکته را نشان می دهد.

۳. روش SET TO NULL : با حذف فایل مرجع مقدار FK در فایلهای رجوع کننده NULL می شود به شرطی که جزء کلید نباشد.

٤. SET TO DEFAULT شبیه روش ۳ است به جای NULL سیستم مقدار Default Value را می گزارد

٥. روش NO ACTION (روش عدم اقدام) : دوپیشنهاد برایش شده:

۱- حذف تاپل مرجع و نه اقدام دیگر

۲- عمل عدم اقدام مطلق : عمل حذف مرجع اصلا انجام نشود.

- مطلوب این است که به طراح پیاده ساز OPTION (حق انتخاب) دهند .

کنجکاوی : درحالت وجود چرخه ارجاع کدام روش قابل انجام نیست ؟ کدام روش انجام می شود اما ممکن است مشکل ایجاد کند ؟

پاسخ: restrict قابل انجام نیست چون نمی توان نوشت cascade مشکل دارد، چون ممکن است تعداد زیادی از تاپل ها حذف شوند که مورد نظر نباشد.

ابتدای صدای ۲ جلسه هفتم

عملیات در RDB : (تا ۲ تست)

۱- جبر رابطه ای Relational Algebra

۲- حساب رابطه ای Relational Calculate

جبر رابطه ای:

عملگر ها:

- عملگر های معمولی یا متعارف : همان عملگر های جبر مجموعه ها (اشتراک – اجتماع – تفاضل – ضرب کارتزین)

نکته : برای سه عملگر اول عملوند ها باید نوع سازگار (Type Compatible) باشند.

تعریف نوع سازگار: R1 و R2 نوع سازگارند اگر $H_{R1}=H_{R2}$ یعنی Heading آنها باید برابر باشد.

مثال: $S \cup SP$ نمی توان نوشت.

نکته : حاصل ارزیابی هر عملگر جبری و نیز هر عبارت جبری باز هم یک رابطه است. (خاصیت بسته بودن جبر رابطه ای)

کنجکاوی: شرط $R_1 \times R_2$ (R_1 کارتزین R_2) چیست؟

- عملگر های خاص:

۱- گزینش (انتخاب) Restrict

۲- پرتو Project

۳- پیوند (گونه هایی دارد)

٤- نیم تفاضل

٥- تقسیم

٦- گسترش

۷- تلخیص (گروه بندی)

۸- دگر نامی

۹- مقایسه

۱۰- انتساب

شرح عملگر ها:

۱- عملگر گزینش : Restrict .گاهی به آن Select جبری هم می گویند.

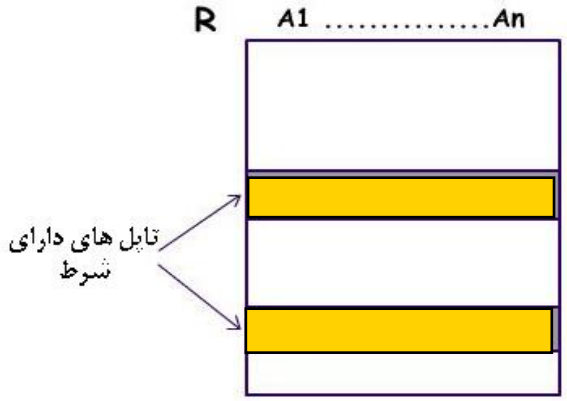
نماد : ς_c یا ς_p

c: condition شرط گزینش
p: predicate مسند گزینش

شکل کلی :

$\varsigma_c(R)$ یا R WHERE C

عملکرد: در نمایش جدولی رابطه : زیرمجموعه افقی می دهد ← عملگر تاپل(ها) یاب!



مثال : مشخصات کامل تهیه کنندگان ساکن C2 با Status ≤15

```
ςc (S)
      City ='c2' AND STATUS ≤15
```

SQL:

```
SELECT S.*
      FROM S
      WHERE …. ;
```

کنجکاوی:

$R' = \varsigma_c(R)$
$C^k R' = ?$

جواب: ممکن است $C^k R$ باشد ممکن هم هست Subset آن باشد.

تمرین :

$R' = OP\ R$
$R = R1\ op\ R2$
$C^k R' = ?$
$C^k R = ?$
$OP \in$ {عملگر های جبر رابطه ای}

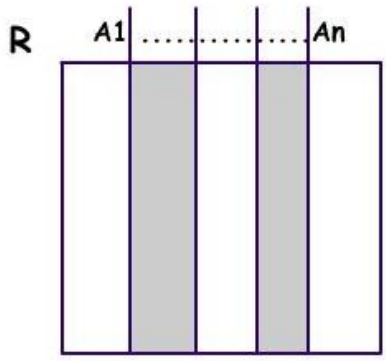
۲- عملگر پرتو : PROJECT [تصویر]

نماد: $\prod_{<L>}$

L : لیست صفات پرتو

شکل کلی: $\prod_{<L>}(R)$ یا Project R over L یا R)[L]

عملکرد: در نمایش جدولی رابطه زیرمجموعه عمودی می دهد : عملگر ستون(ها) یاب



جواب مجموعه است : عملگر پرتو تکراری ها را حذف می کند چون رابطه است

مثال : درخواست کاربر چه بوده است:

$\prod_{<S\#\ ,\ STATUS>}(S)$

جواب : شماره و وضعیت تمام تهیه کنندگان را بدهید.

مثال :

$R = \prod_{<S\#>}(S) - \prod_{<S\#>}(SP)$

شماره تهیه کنندگانی که قطعه تهیه نکرده اند.

مثال : اگر کاردینالیتی $CARD(R)>0$ باشد چه می توان گفت؟

جواب: وجود دارد حداقل یک تهیه کننده که قطعه تهیه نکرده اند.

مثال : اگر $CARD(\prod_{<S\#>}(S))=a$ و $CARD(\prod_{<S\#>}(SP))=b$ و $a<b$ باشد چه می توان گفت؟

جواب : قاعده C2 حداقل در یک عمل ذخیره سازی رعایت نشده است.

۳-۱- عملگر پیوند: (الصاق)

شکل ریاضی عمومی:

مفروضند رابطه های :

R1(A1,A2,……..,An)
R2(B1,B2,……..,Bn)

شکل کلی:

R1 theta-join R2
R1 $Join_{\theta}$ R2
R1 $\bowtie_C$ R2

C شرط های پیوند است. یک شرط به شکل زیر است:

$C := R_1.A_i$ theta $R_2.B_j$

نکته : ازنظر ریاضی صفاتی که شرط پیوند روی آنها اعمال می شود باید هم دامنه و ناهم نام باشد زیرا جواب پیوند رابطه است و heading رابطه مجموعه است عنصر تکراری ندارد.

شرط پیوند:

$H_{R1} \cap H_{R2} = 0$

اگر صفت (صفات) همنام در H دو رابطه باشد یکی از آنها را باید دگر نام کرد.

عملکرد:

$H_R = H_{R1} \cup H_{R2}$

دربدنه R: تاپل های حائز شرط باهم پیوند می شوند .

نکته بسیار مهم: پیوند در شکل عمومی حالتی است از ضرب کاترین

$R1 \bowtie_C R2 = P\varsigma_c(R1 \times R2)$

پیوند در شکل عمومی حالت خاصی است از ضرب کارتزین

نکته:

$R1 \bowtie_C R2 = R2 \bowtie_C R1$

مثال:

S(S# , …. , city)

S#	…	…	City
S1	…	…	C1
S2	…	…	C2
S3	…	…	C3
S4	…	…	C4
S5	…	…	C5

P(P# , …. , W , city)

P#	…	…	W	City
P1	…	…	5	C1
P2	…	…	7	C2
P3	…	…	5	C3
P4	…	…	6	C4
P6	…	…	10	C6

دو رابطه صفت مشترک دارند و باید یکی از آنها دگرنام شود:

$R1 = S \bowtie_{S.City=P.City} (P \text{ RENAME City AS PCity})$

جدول پاسخ:

R1(S# , …. , City , P# , ….., W , PCity)

S#		City	P#		W	City
S1		C1	P1		5	C1
S2		C2	P2		7	C2
S3		C3	P3		5	C3
S4		C4	P4		6	C4

۴۹:۰۰

گونه های پیوند:

۱. پیوند طبیعی: natural join

- Theta : عملگر مساوی است
- صفت پیوند در رابطه جواب تکرار نمی شود بنابراین صفت یا صفات پیوند در اینجا همدامنه و هم نام هستند.

نکته : اگر $H_{R1} = H_{R2}$ باشد:

$$R1 \bowtie R2 = R1 \cap R2$$

نکته : در پیوند طبیعی وقتی نام صفت پیوند را قید نکنیم پیوند روی تمام صفات مشترک خواهد بود.

نکته : اگر $R1 \cap R2 = 0$ آنگاه $R1 \bowtie R2 = R1 \times R2$

مثال: مشخصات کامل جفت تهیه کننده – قطعه هر شهر را بدهید:

$$R2 = S \bowtie_{(S.City=P.City)} P$$

جواب :

R2: (S#,……,City, P#, ….., W)

S#		City	P#		W
S1		C1	P1		5
S2		C2	P2		7
S3		C3	P3		5
S4		C4	P4		6

۳-۲- عملگر نیم پیوند Semi-Join

نماد: $\ltimes_c$

Theta ={= . != , < , …}

R1 $\ltimes_c$ R2 = $\Pi_{<HR1>}$(R1 |><$_c$ R2)

یعنی تاپل های پیوند شدنی فقط از رابطه چپ

نکته : heading نتیجه همان heading ، R1 است.

در بدنه نتیجه : تاپل های پیوند شدنی از رابطه چپ

مثال:

R3 = S $\ltimes$ P

تاپل های پیوند شدنی از رابطه چپ:

R3=(S#, …. , City)

S#		City
S1		C1
S2		C2
S3		C3
S4		C4

٤- عملگر نیم تفاضل Semi-Minus

R1 SemiMinus R2 = R1 – (R1 SemiJoin R2)

در بدنه رابطه نتیجه : تاپل های پیوند نشدنی از رابطه چپ

مثال:

S SemiMinus P =
R4= (S# , ….., City)
S5, ……, C5 فقط

۵- عملگر فراپیوند یا پیوند بیرونی Outer-Join

این عملگر در سه شکل LEFT و RIGHT و FULL تعریف شده است.

LEFT : $\bowtie_C$
RIGHT : $\bowtie_C$
FULL : $\bowtie_C$

عملکرد Left-Outer-Join

$R' = R1 \bowtie_C R2$
$HR' = H_{R1} \cup H_{R2}$

در بدنه R' : تاپل های پیوند شدنی از دو رابطه و تاپل های پیوند نشدنی رابطه چپ گسترش یافته با NULL

عملکرد RIGHT قرینه عملکرد چپ است.

عملکرد FULL ترکیب دوتای آن هاست.

مثال :

$R4 = S \bowtie_C P$
R4 = (S# , … , City , P# , …., W)

S#		City	P#		W
S1		C1	P1		5
S2		C2	P2		7
S3		C3	P3		5
S4		C4	P4		6
S5		C5	NULL		NULL

کلید این رابطه حاصل از Join ترکیب کلید هاست.

آقای Date چون این پیوند ها تعداد زیادی NULL ایجاد می کند این روابط را قبول ندارد.

یکی از معایب Outer-Join این است که مخرب کلید است. کلید پیوند ترکیب کلید هاست و از طرفی جزءکلید نمی تواند NULL باشد اما در Outer-Join هیچ ستونی از قاعده NULL شدن در امان نمی ماند.

بنابر این رابطه جواب از نظر تئوریک رابطه واقعی نیست. چون از نظر تئوری هر رابطه حداقل یک کلید دارد.

تمرین : عملگر های OuterJoin را در جبر و SQL شبیه سازی کنید.

کنجکاوی : OuterUnion چیست؟

راهنمایی : در Union رابطه ها باید Type Compatible باشند اما در اینجا نیازی نیست رابطه ها Heading شان یکی باشد.

$$R' = R1 \text{ OUTER UNION } R2$$
$$R' = (H_{R1} \cup H_{R2})$$

ابتدای صدای ۳ جلسه هفتم

۶- عملگر تقسیم

مفروضند رابطه های :

مقسوم : $R1(A1,A2,\ldots\ldots,An,B1,B2,\ldots\ldots,Bm)$

مقسوم علیه : $R2(B1,B2,\ldots\ldots,Bm)$

برای سهولت در نوشتن:

$X = A1,A2,\ldots\ldots,An$
$Y = B1,B2,\ldots\ldots,Bm$

شرط تقسیم:

$H_{R2} \subseteq H_{R1}$

$R1(X,Y) : R2(Y) = R3(X)$

عملکرد:

$$H_{R3} = H_{R1} - H_{R2}$$

در بدنه R3 : بخش X از تاپل هایی از R1 که تمام مقادیر Y از R2 را داشته باشند.

مثال ۱:

R1(S# , P#)	: R2(P#)	= R3(S#)
S1 ,P1	P1	S1
S1 ,P2	P2	
S1 ,P3	P3	
S2 ,P1		
S2 ,P2		
S3 ,P3		

مثال ۲:

R1(S# , P#)	: R4(P#)	= R5(S#)
S1 ,P1	P1	S1
S1 ,P2	P2	S2
S1 ,P3		
S2 ,P1		
S2 ,P2		
S3 ,P3		

کنجکاوی : عملگر تقسیم را به کمک عملگر های دیگر بنویسید؟(جواب در یادداشت تکمیلی)

مثال هایی از کاربرد جبر رابطه ای در عملیات در RDB :

۱- بازیابی:

حال بررسی مثال هایی در ضمیمه A3 :

RDB:

S(S#,SNAME,STATUS,CITY)

P(P#,PNAME,COLOR,WEIGHT,CITY)

SP(S#,P#,QTY)

مثال هایی از جبر رابطه ای :

Q1 : نام تهیه کنندگانی را بدهید که قطعه P2 را تهیه می کنند. (قبلا در SQL با بیش از ۸ روش نوشتیم)

پاسخ : از آنجایی که نام در S است و تهیه شدن در SP به این دو رابطه نیاز داریم:

```
((SP JOIN S) WHERE P# = 'P2')[SNAME]
```

پاسخ با نماد های ریاضی:

$\Pi_{<SNAME>}(\sigma_{<P\# = 'P2'>}(SP \bowtie S))$

Q2 : نام تهیه کنندگانی را بدهید که حداقل یک قطعه قرمز رنگ را تهیه می کنند.

پاسخ : از آنجایی که نام در S است و تهیه شدن در SP و رنگ در P به هر سه رابطه نیاز داریم:

ابتدا قطعات قرمز را جدا می کنیم . سپس با SP ، JOIN می کنیم و S# را Project می کنیم . تا اینجا شماره تهیه کنندگان قطعات قرمز بدست می آید و ...

توجه داشته باشید که اکثر Package ها توصیه می کنند که ابتدا گزینش اعمال شود و حتی الامکان پرتو. بویژه وقتی که Join داریم.

(((P WHERE COLOR = 'RED') JOIN SP)[S#] JOIN S)[SNAME]

سئوال: آیا در Q2 منطقا project روی S# لازم است؟

منطقا لازم نیست بلکه برای کوچک شدن رابطه سمت چپ Join آن را می زنیم.

Q3 : نام تهیه کنندگانی را بدهید که تمام قطعات را تهیه می کنند.

((SP[S# , P#] DIVIDE BY P[P#]) JOIN S)[SNAME]

Q4: شماره تهیه کنندگانی را بدهید که حداقل تمام قطعات تهیه شده توسط 'S2' را تهیه می کنند.

توضیح : نظیر Q3 است الا اینکه ۲ ارجاع به یک رابطه (SP) داریم.

SP[S# , P#] DIVIDE BY (SP WHERE S# = 'S2')[P#]

Q5: نام تهیه کنندگانی را بدهید که قطعه 'P2' را تهیه نمی کنند.

پاسخ : ابتدا شماره آنهایی که P2 را تهیه می کنند بدست آورده و سپس با MINUS آنهایی که P2 را تهیه نمی کنند بدست آوردیم:

((S[S#] MINUS (SP WHERE P# ='P2')[S#]) JOIN S)[SNAME]

سئوال: در Q5 کدام یک از Project ها منطقا لازم است؟

هر دو لازم است چون در MINUS باید Heading هر دو رابطه یکی باشد.

یادآوری : جواب هر عملگر باز یک رابطه است.

عملگر EXTEND :

عملگر گسترش است صفت یا صفاتی به رابطه اضافه می کند و رابطه جدید به ما می دهد.

یکی از پیاده سازی های ممکن آن در SQL ، ALTER است.

مثال:

1. EXTEND S ADD 'supplier' AS TAG

ستون TAG اضافه شده و تنها نکته این است که 'supplier' به عنوان Deafault Value آن می باشد.

2. (EXTEND S ADD CITY AS SCITY)[S#,SNAME,STATUS,SCITY]

در واقع شبیه سازی عملگر RENAME است.

معادل است با : (این RENAME ، SQL نیست)

S RENAME CITY AS SCITY

تست : در شبیه سازی عملگر RENAME از چه عملگر هایی استفاده می شود؟

پاسخ : EXTEND و PROJECT

3. EXTEND (P JOIN SP) ADD [WEIGHT * QTY] AS SHIPWT

در اینجا رابطه ای به ما می دهد :

R(P#,…,WEIGHT,CITY,S#,QTY,SHIPWT)

اگر به شکل Base Table تعریف کنیم مثالی است از صفت محاسبه شده که ما آن را به هر دلیل ذخیره (Stored) کرده ایم. ممکن است VIEW یا ... تعریف کرده باشیم.

عملگر SUMMARIZE : تلخیص یا گروه بندی

نمایش در برخی از متون : γ (گاما)

یک پیاده سازی ممکن از این عملگر ریاضی در SQL : GROUP BY

SUMMARIZE تاپل های رابطه را گروه بندی می کند به نحوی که مقدار صفت یا صفات گروه بندی در هر گروه یکسان است و معمولا با تابع جمعی (گروهی) مثل SUM و MAX و... استفاده می شود.

مثال:

- SUMMURIZE SP BY (P#) ADD SUM(QTY) AS TOTQTY

در واقع مدل تئوری این است:

SUMMURIZE R BY R' $\rightarrow H_{R'} \subseteq H_R$

رابطه جواب : برای بدست آوردن جواب ابتدا باید Heading را بطه را پیدا کنیم.

نکته : در Heading صفت یا صفات گروهبندی می آید.

در نتیجه رابطه جواب خواهد شد: شماره هر قطعه و کل تعداد تهیه شده از آن.

- SUMMURIZE SP BY (S#) ADD COUNT AS NP

سئوال : درخواست کاربر چه بوده است؟

شماره تهیه کنندگان و تعداد قطعاتی که تولید کرده.

- SUMMURIZE (P JOIN SP) BY (CITY) ADD COUNT AS NSP

سئوال : درخواست کاربر چه بوده است؟

نام شهر و تعداد قطعات تهیه شده از هر شهر

```
SUMMURIZE SP BY(P#) ADD

SUM(QTY) AS TOTQTY ,

AVG(QTY) AS AVGQTY
```

عملگر های جمعی : Aggregate Operator

از جمله : COUNT ، SUM ، MIN ، MAX ، AVG و ....

مثال :

```
SUM (SP WHERE S# =S1 ,QTY)
```

کل تعداد قطعات تهیه شده توسط S1 را بدهید.

کنجکاوی : عملگر های GROUP و UNGROUP چه عملکردی دارند؟

GROUP رابطه نرمال را غیر نرمال می کند.

UNGROUP رابطه غیر نرمال را نرمال می کند.

مثال:

```
GROUP SP BY (P#,QTY) AS PQTY;
```

می توانیم در رابطه بالا به جای BY از PER استفاده کنیم.

نتیجه رابطه زیر خواهد شد: در صفحه A4 مثال موجود می باشد:

SP' (S# , PQTY)

S1	p1	100
	P2	90
	P3	80
S2	....	

توجه: تقسیم رابطه ای لزوما همیشه عکس ضرب رابطه ای نیست.

نکته ۱:

$(R_1 \div R_2) \times R_2 \subseteq R_1$

نکته ۲: شبیه سازی عملگر تقسیم:

$R_1(X,Y) \div R_2(Y) = \prod_{<X>}(R_1) - (\prod_{<X>}(R_1) \times R_2 - R_1))$

تست: در شبیه سازی عملگر تقسیم به فرض استفاده از عملگر پرتو و MINUS ، چند بار پرتو و تفریق لازم است؟

- ۲و۳ *
- ۲و۳
- ۳و۳
- ۱و۲

جلسه هشتم چهارشنبه ۳۱/٤/۱۳۸۸ ساعت ۷:۳۰

مقاسه دو رابطه R_1 و R_2 :

شرط مقایسه : $H_{R1} = H_{R2}$

بدنه ها مقایسه می شود از نظر:

- زیر مجموعه بودن
- هم مجموعه بودن
- مساوی بودن
- .....

نتیجه مقایسه :

- True
- False

مثال : کدام می تواند صحیح باشد:

$\prod<S\#>(S) < \prod<S\#>(SP)$?
$\prod<S\#>(S) = \prod<S\#>(SP)$?
$\prod<S\#>(S) > \prod<S\#>(SP)$?

پاسخ : مساوی یا بزرگتر است اگر قاعده C2 رعایت شده باشد. و اگر کوچکتر بود قاعده C2 حداقل دریک عمل ذخیره سازی رعایت نشده است.

کنجکاوی : در SQL چگونه دو جدول را مقایسه می کنیم ؟

۲– عملیات ذخیره سازی:

سئوال: آیا با جبر رابطه ای می توان انجام عملیات درج – حذف – به هنگام سازی انجام داد ؟

بله

به جدول زیر توجه کنید:

عمل	عملگر
درج	UNION
حذف	MINUS (در SQL ، EXCEPT است)
به هنگام سازی	ابتدا MINUS سپس UNION

نکته : مدل کاملا تئوریک است یعنی هیچ گاه در عمل از INSERT برای درج استفاده نمی شود.

مثال :

S := S UNION {<S# = S7>,<Sname=Sn7>,<Status=10>,<City=C7>}

توضیح:

عملگر : انتساب =: می باشد.

S متغیری است از نوع رابطه (Relation Variable)

عبارت داخل {} یک مقدار است که به آن متغیر منتسب می شود.

آنچه داخل {} است رابطه ای نوع سازگار با S و با کاردینالیتی ۱ می باشد.

عبارت های داخل {} خلاصه نوشته شده اند وگرنه می بایست دامنه مقادیر نیز مشخص می شد:

[Sname:دامنه]

در صورتی که بخواهیم عمل حذف انجام شود از MINUS استفاده می کنیم.

برای UPDATE ابتدا با MINUS تاپل قدیم را حذف و سپس با UNION تاپل جدید را اضافه می کنیم.

جمع بندی:

۱- جبر رابطه ای "ز بانی" است از نظر رابطه ای کامل (کمال رابطه ای دارد Relational Completeness) : یعنی هر رابطه متصور در universe رابطه را می توان با یک عبارت رابطه ای بیان کرد .

۲- با توجه به ۱ جبر رابطه ای ضابطه تشخیص کمال رابطه ای زبان های رابطه ای است.

۳- کاربردها:

۱- عملیات بازیابی

۲- عملیات ذخیره سازی

۳- تعریف انواع رابطه ها ی مشتق مثل :

رابطه مجازی View (Non Stored)

رابطه لحظه ای Snapshot که گاهی به آن Materialized View نیز می گویند.

یادآوری : View = Relational Expression

هر عبارت جبری که حاصل ارزیابی آن یک رابطه است می تواند view باشد.

۱۹:۵۰

حساب رابطه ای

۱. امکانی دیگر برای بازیابی از RDB

۲. شاخه ای است از منطق ریاضی (منطق گزاره ها - منطق محمولات – منطق مسندات)

۳. ازنظر کاربر : معادل است با جبر رابطه ای [از نظر توان تعریف رابطه ها] یعنی هر رابطه ای که با یک عملیات جبری بیان شود را می توان با یک عبارت حساب رابطه ای هم بیان کرد و بالعکس .

تفاوت جبر رابطه ای و حساب رابطه ای:

- **جبر** حالت دستوری دارد PRESCRIPTIVE
- حساب حالت توصیفی دارد (داده ها را به نحوی توصیف می کنیم) DESCRIPTIVE

با توجه به نکته ٤ : حساب رابطه ای از جبر رابطه ای نا رویه ای تر است. (nonProcedural)

می توانیم بگوییم درجه نا رویه ای بودن حساب بیشتر است.

نکته تکمیلی : زبان های پایگاهی (DSL) ممکن است رویه ای یا نارویه ای باشند.

- زبان رویه ای زبانی است که در برنامه سازی با آن، برنامه ساز نه تنها باید مشخص کند چه داده ای می خواهد بلکه باید رویه دستیابی به داده درکادر DS را مشخص کند.(رویه رسیدن به یک نود دلخواه از ریشه)
- زبان نارویه ای زبانی است که در برنامه سازی با آن کاربر به نحوی می گوید . چه داده ای می خواهد . رویه دست یابی به آن را مشخص نمی کند . مثل زبان های مبتنی بر حساب رابطه ای و نیز جبر رابطه ای.

حساب رابطه ای :

- حساب تاپلی Tuple Oriented (در SQL پیاده سازی شده)
- حساب دامنه ای Domain Oriented (میدانی)

در حساب رابطه ای دو نوع سور داریم : (چندی نما یا Quantifier)

- سور وجودی ∃ (Exists)
- سور عمومی ∀ (For All)

به کمک این سورها عبارات منطقی ریاضی می نویسیم موسوم به فرمول های خوش ساخت well form formula یا WFF

یادآوری:

Exists X(f) :

وجود دارد حداقل یک مقدار برایX به نحوی که ...

ForAll X(f) :

بازای تمام مقادیر X ...

مثال :

فرض : X از مجموعه اعداد صحیح مثبت مقدار بگیرد وجود دارد یک X به نحوی که X کوچکتر از ۱۰

Exists X(X<10)

حاصل ارزیابی TRUE

For All X(X<10)

حاصل ارزیابی FALSE

بین این دو سور روابط هم از وی وجود دارد. (برگه C1)

روابط هم ارزی بین سور های وجودی و عمومی:

FORALL T(f) ≡ NOT EXISTS T(NOT f)
EXISTS T(f) ≡ NOT (FORALL (NOT f))
FORALL T((f) AND (g)) ≡ NOT EXISTS T(NOT(f) OR NOT(g))
FORALL T((f) OR (g)) ≡ NOT EXISTS T(NOT(f) AND NOT(g))
EXISTS T((f) OR (g)) ≡ NOT FORALL T(NOT(f) AND NOT(g))
EXISTS T((f) AND (g)) ≡ NOT FORALL T(NOT(f) OR NOT(g))

در روابط ردیف ۳ تا ٦، f می تواند خود رابطه بولی باشد.

همچنین روابط ایجابی زیر را نیز داریم:

FORALL T(f) ⇒ EXISTS T(f)
NOT EXISTS T(f) ⇒ NOT FORLL T(f)

مثال هایی از کاربرد حساب رابطه ای در عملیات در دیتابیس:

برای استفاده از این سور ها در حساب تاپلی باید از مفهوم متغیر تاپلی استفاده کنیم.

متغیر تاپلی Range Variable (لازم درحساب تاپلی) روی یک عبارت رابطه ای (رابطه) تعریف می شود و هر لحظه مقدارش یک تاپل از رابطه است .

RANGEVAR SX RANGES OVER S;

یک متغیر به نام SX تعریف کردیم که روی S رنج می شود.

RANGEVAR SPX RANGES OVER SP;
RANGEVAR C2X RANGES OVER (S WHERE CITY='C2');

برگه C2 مثال ها از date:

رابطه های S و P و SP مفروضند: سعی شده همان پرسش هایی را که در جبر دیدیم در حساب نوشته شود:

Q1 : شماره تهیه کنندگان ساکن C2 با وضعیت بیش از ۲۰ را بدهید:

SX.S# WHERE SX.City = ‘C2’AND SX.Status > 20

این یک عبارت حساب رابطه ای است که در آن متغیر (SX) ما تحت سور نیست. از سور استفاده نکردیم و به آن متغیر آزاد می گویند. (حالت خاص است) اما این عبارت را می توان به صورت زیر نیز نوشت:

SX.S# WHERE EXISTS SX (SX.City=’C2’ AND SX.Status>20)

عبارت (SX.City=’C2’ AND SX.Status>20) درواقع f ماست.

و همچنین عبارت EXISTS SX متغیر تحت سور یا Bound (مقید یا تصویر شده)

عبارت EXISTS SX (SX.City=’C2’ AND SX.Status>20) همان WFF است که ارزیابی می شود به TRUE یا FALSE . یعنی عبارت SQL ی که در آن EXIST هست دیتا نمی دهد بلکه EXISTS ارزیابی می شود اگر TRUE بود آن وقت SELECT اجرا می شود.

اصل این است که از سور استفاده شود و مثال بالا که سور نداشت حالت خاص است.

Q2 : نام تهیه کنندگان قطعه P2 را بدهید:

SX.Sname WHERE EXISTS SPX (SPX.S# = SX.S# AND SPX.P# = ‘P2’)

توضیح : SPX.S# = SX.S# یعنی تهیه کننده قطعه تولید کرده باشد و SPX.P# = ‘P2’ یعنی قطعه P2 را تولید کرده باشد.

عبارت SQL معادل:

```
SELECT S.Sname
FROM S
      WHERE EXIXTS (SELECT *
            FROM SP
            WHERE SP.S# = S.S#              ←  Correlated Queries
            AND
            SP.P# = ’P2’);
```

WFF با این رنگ مشخص شده است.

پس EXISTS در SQL با ساختار SELECT تطبیق داده شده است.

طرز اجرا از دید کاربر(نه درون سیستم) : بازای هر سطر از جدول S ، WFF داده شده بررسی می شود اگر TRUE باشد حاصل ارزیابی Sname از آن سطر جواب است.

Q3 : نام تهیه کنندگانی که حداقل یک قطعه قرمز رنگ تهیه می کنند:

```
SX.Sname WHERE EXISTS SPX (SP.S# = SPX.S# AND

EXISTS PX(PX.P# = SPX.P# AND PX.COLOR='RED')
```

Q4 : نام تهیه کنندگانی که حداقل یک قطعه تهیه شده توسط S2 را تهیه می کنند:

Q4 شبیه Q3 است منتها دو نکته دارد:

نکته اول :EXISTS ها ابتدا در پی هم آمده اند و بعد شرط ها آمده اند. در منطق به این روش می گویند فرم پیشوندی. در SQL به این شکل نمی توانیم بنویسیم.

```
SX.Sname WHERE EXISTS SPX (EXISTS SPY(SX.S# = SPX.S#

AND SPX.P#=SPY.P#

AND SPY.S#='S2'))
```

نکته دوم : استفاده از دو متغیر تاپلی روی یک رابطه است (SPX و SPY). از دیدگاه کاربردی دو بار کلمه تهیه آمده است پس باید دو بار ارجاع به SP داشته باشیم.

Q5 : نام تهیه کنندگانی که تمام قطعات را تهیه می کنند:

مثالی است از FORALL :

```
SX.Sname WHERE FORALL PX (EXISTS SPX(SPX.S# = SX.S#

AND SPX.P#=PX.P#))
```

معادل آن با EXISTS :

```
SX.Sname WHERE NOT EXISTS PX (NOT EXISTS SPX(SPX.S# = SX.S#

AND SPX.P#=PX.P#))
```

این مثال را در جبر با تقسیم نوشته بودیم.

Q6 : نام تهیه کنندگانی که قطعه P2 را تهیه نمی کنند:

به این مثال فرم نایی EXISTS می گویند.

```
SX.Sname WHERE NOT EXISTS SPX(SPX.S# = SX.S#

AND SPX.P#='P2')
```

برگه C3 مثال از رانکوهی

ابتدای صدای ۲ جلسه ۸

طراحی RDB

روش بالا به پایین

برای طراحی RDB دراساس دو روش وجود دارد:

۱. روش بالا به پایین

۲. روش سنتز(نرمال تر سازی)

روش بالا به پایین:

تبدیل نمودار مدلسازی معنایی به مجموعه ای از رابطه ها ی نرمال (و نه لزوماً در نرمالترین فرم)

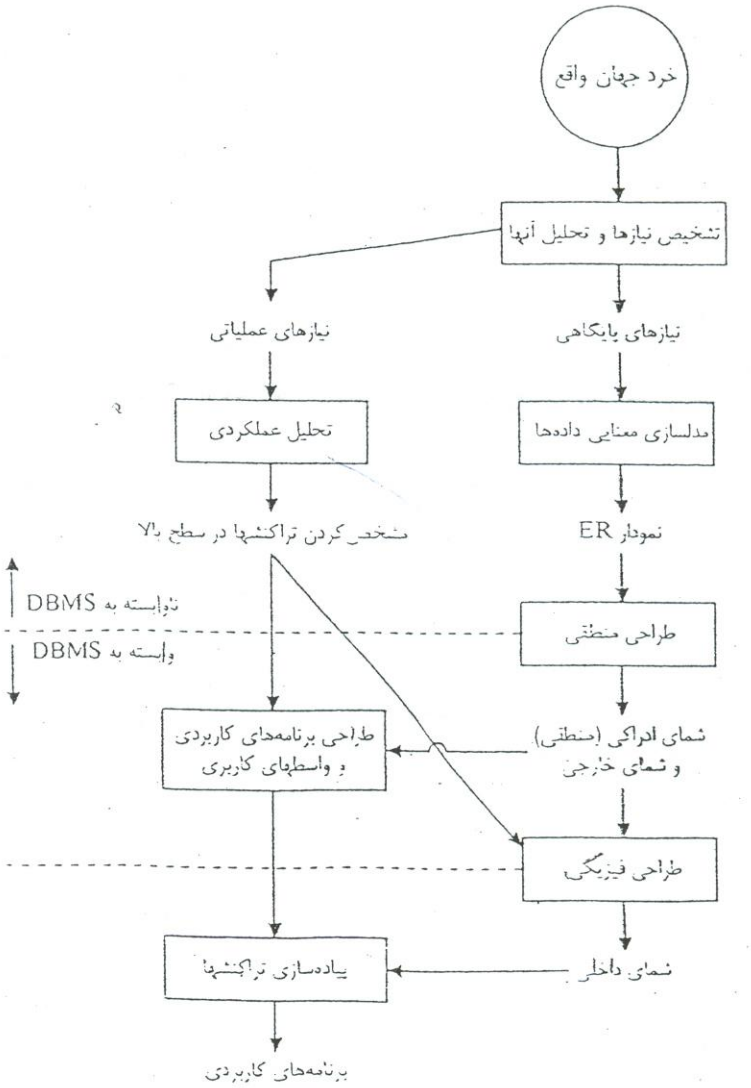
نکته : در روش Top-Down با ایده های نرمالتر سازی کاری نداریم.

در عمل : روش ترکیبی ← ابتدا Top-Down و بعد در صورت لزوم رابطه ها را نرمال تر می کنیم.

طراحی

۱- مراحل اساسی طراحی پایگاه داده ها (ساده شده)

۲- برای جزئیات کار به وظایف DBA (در یاد داشت های تکمیلی) مراجعه شود.



۲- خصوصیات طراحی خوب:

طراحی منطقی خوب باید خصوصیات زیر را داشته باشد:

۱. نمایشی واضح از "خرد جهان واقع" باشد.

۲. نمایشی صحیح از "خرد جهان واقع" باشد به گونه ای که هیچ اشتباه معنایی (سمنتیک) در پاسخگویی به پرسش های کاربران بروز نکند.

۳. نمایشی جامع از "خرد جهان واقع" باشد. دربرگیرنده تمام صفات مورد نیاز کاربران، به گونه ای که تولید همه برنامه های کاربردی به آسانی امکان پذیر باشد.

٤. تمام محدودیت ها (قواعد) جامعیتی که قابل اعمال در هر مرحله ای از طراحی منطقی باشند، در طراحی منظور شده باشند(به ویژه وابستگی های بین صفات که در بحث نرمالترسازی رابطه ها خواهیم دید)

٥. معنای هر یک از صفات از هر نوع موجودیت به درستی رعایت شده باشد.

٦. کمترین میزان افزونگی را داشته باشد.

۷. کمترین میزان اختلاط اطلاعات را داشته باشد(به بحث نرمالتر سازی مراجعه شود)

۸. کمترین دشواری در انجام عملیات ذخیره سازی در پایگاه داده ها بروز کند.

۹. انعطاف پذیری داشته باشد به گونه ای که بتوان نیاز های جدید کاربران را به آسانی منظور کرد.

۱۰. کمترین میزان "هیچمقدار" یا "اطلاع نیست" در پایگاه داده ایجاد شود.

۱۱. هیچ اطلاع حشو(زائد) در اثر انجام عملیات در رابطه ها (مثلا عمل پیوند دو رابطه) در پایگاه داده ها پدید نیاید. بنابراین حتی الامکان باید از تجزیه یک رابطه به دو یا بیش از دو رابطه به گونه ای که بدست آوردن اطلاعات مورد نظر نیازمند پیوند رابطه ای حاصل از تجزیه روی صفت(صفات) غیر کلید باشد اجتناب کرد(به گفتار های چهاردهم مراجعه شود)

۱۲. با در نظر گرفتن طراحی فیزیکی و تاثیر آن در طراحی منطقی بیشترین کارایی برای سیستم کاربردی پایگاه داده ها تامین شود.

بعضی از این خصوصیات لزوما مستقل از یکدیگر نیستند.

کنجکاوی : آیا خصوصیات دیگری هم مطرح است؟

حتی امکان در عملیات بازیابی سریع باشد.

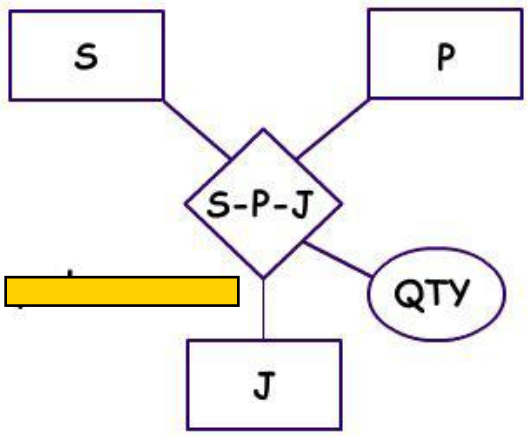
تمام داده های محیط قابل نمایش باشد.

در نمودار ER حالات متعدد داریم:

حالت اول:

چندی ارتباط M:N درجه $N \geq 1$ ← $N+1$ رابطه لازم است.

یک رابطه برای هر نوع موجودیت و یک رابطه برای خود ارتباط .



S(S#,…)
P(P#,…)
J(J#,…)
SPJ(S#,P#,J#,QTY)

در این مثال خاص این ترکیب S#,P#,J# ، Ck است وچون تنها CK است می شود PK و S# و P# و J# هر یک FK هستند.

نکته مهم در این حالت یافتن کلید است.

کلید نمایشگر ارتباط N گانی (با چندی عمومی …N:M:P) تشکیل شده است از کلید هر یک از رابطه ها واحیاناً صفت یا صفاتی از خود ارتباط **(حداقل ترکیب کلید ها)**

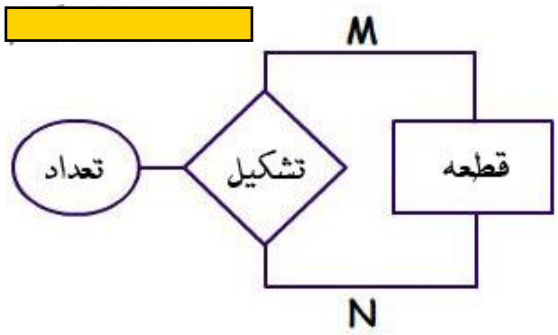
حالت دوم :

چندی 1:N

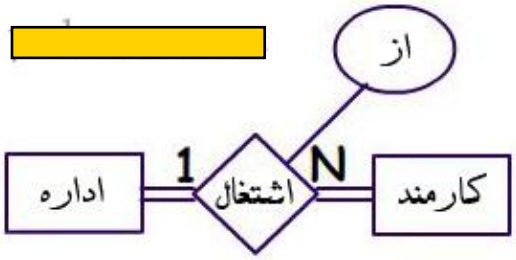
درجه $N \geq 1$: درجه N=1 یا N=2

N رابطه لازم است.

مثال: N=1



مثال:



در یک اداره N کارمند مشغولند.

```
DEPT(D#,…)

EMPL(E#,…,D#,F)   → FK : D#
```

تکنیک این است: در این حالت سمت N از سمت 1 ، FK می گیرد.

رابطه نمایشگر نوع موجودیت است سمت N از رابطه نمایشگر نوع موجودیت سمت ۱، FK می گیرد و این FK خارج از کلید است.

در این مثال D# در سمت EMPL ، FK است و چون مشارکت الزامی است NULL نیز نمی تواند باشد.

در تعریف رابطه EMPL باید دقت کنیم برای D# ، NOT NULL بزنیم، این محدودیت مشارکت الزامی مستقیماً با طراحی نمایش داده نمی شود. باید در شمای رابطه مشخص کنیم.

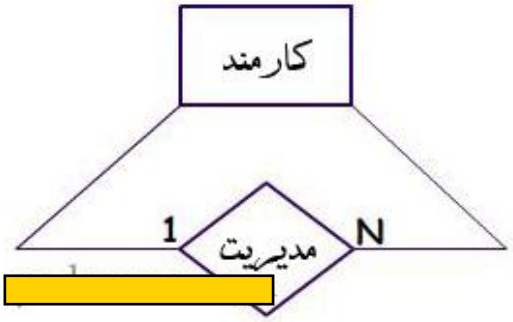
نکته : همین حالت 1:N با درجه ۲ را ممکن است با ۳ رابطه هم طراحی کنیم وقتی که مشارکت سمت N غیر الزامی باشد و درصد مشارکت ضعیف (پایین) باشد در این حالت مثل حالت N:M باید طراحی شود.

```
DEPT(D#,…)

EMPL(E#,…)

DEEM(E#,D#,F)      → FK : D#
```

مثال: 1:N درجه N=1

EMP(E#,…,EMNGR#) → EMNGR# : FK شماره مدیر

رابطه به خود رجوع کننده

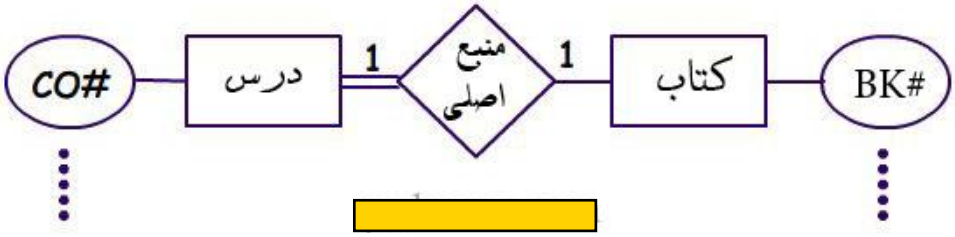


وقتی که مشارکت غیر الزامی و درصد مشارکت ضعیف باشد ، می توان با دو رابطه نوشت .

حالت سوم :

چندی 1:1

- درجه N=1 : با ۱ یا ۲ رابطه بسته به وضعیت می توان طراحی کرد.
- درجه N=2 : با ۱ یا ۲ یا ۳ رابطه بسته به وضعیت می توان طراحی کرد.



هر درس یک منبع اصلی باید داشته باشد اما هر کتابی منبع اصلی نیست.

طراحی ۱:

تکنیک: سمت با مشارکت الزامی FK می گیرد:

COUR(CO#,….,BK#)
BOOK(BK#,…)

سئوال: اگر عکس عمل می کردیم یعنی به BOOK ، FK می دادیم چه می شد؟

پاسخ: NULL ایجاد می شود چرا که تعداد زیادی کتاب است که تعداد محدودی از آنها منبع اصلی هستند.

طراحی ۲:

COBO(CO#,…,BK#,…)

توجیه : وقتی که مشارکت طرفین الزامی باشد و ملاحظات دیگری مطرح نباشد. مثل بزرگ شدن درجه رابطه. مثل کاهش کارآیی سیستم در بازیابی.

طراحی ۳:

CO(CO#,…)
BK(BK#,…)
CB(CO#,BK#)

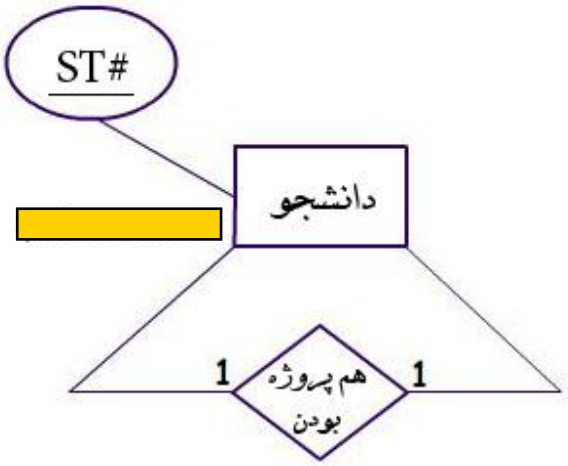
توجیه : وقتی مشارکت طرفین غیر الزامی باشد و به ویژه فرکانس ارجاع به رابطه CB بالا باشد.

کنجکاوی : آیا همین حالت طراحی های دیگری هم دارد؟

۴۲:۰۰

مثال حالت سوم درجه ۱:

با دو یا یک رابطه طراحی می شود.



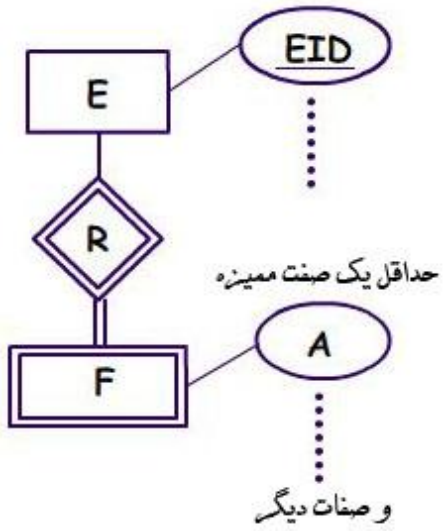
طراحی ۱:

STJST(ST# , STNAME , … , JST#)

شماره دانشجویی هم پرونده: #JST. برای پروژه یک نفره می تواند NULL می باشد.

طراحی ۲: طراحی دیگر کدام است و کِی؟

حالت چهارم : نوع موجودیت ضعیف داریم



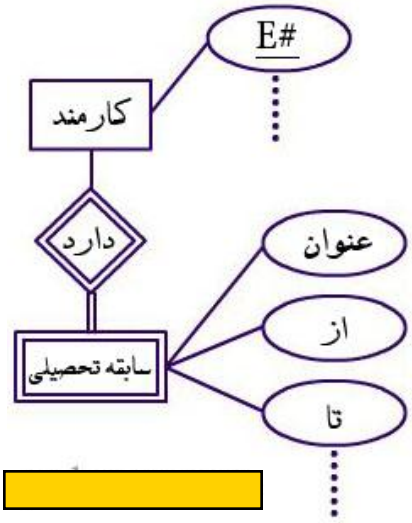
دو رابطه :

- یک رابطه برای قوی

- یک رابطه برای ضعیف (و ارتباط شناسا)

تکنیک: رابطه نمایشگر ضعیف از رابطه نمایشگر قوی FK می گیرد جزء کلید

مثال :



EMPL(E#,…)
EEDUCHIS(E#,Title,From,To) → E# : FK

شناسه در رابطه دوم جزء کلید می شود.

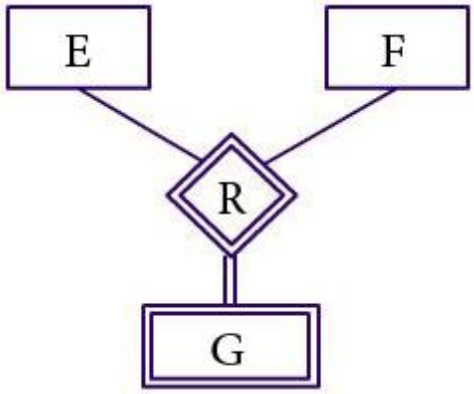
سئوال : شناسه تاکنون در چند حالت جزء کلید شده است؟

۱. ارتباط N:N
۲. ارتباط شناسا

طراحی شود:

نکته: تعیین کلیدهای رابطه ای نمایشگر G ؟؟؟

G(E#,F#,G کلید) → E#,F# : FK



نکته : برای طراحی صفت چندی مقداری سه تکنیک وجود دارد.

حالت پنجم :

صفت چند مقداری

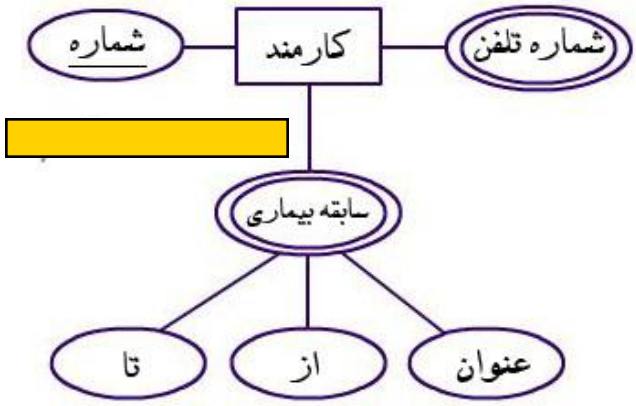
سه تکنیک داریم:

۱- تکنیک عمومی

۲- دو تکنیک خاص

تکنیک عمومی : اگر نوع موجودیت E تعداد n صفت تک صفت تک مقداری وتعداد m صفت چند مقداری داشته باشد M+1 رابطه طراحی می کنیم: یک رابطه برای صفات تک مقداری، یک رابطه برای هرصفت چند مقداری. تکنیک طراحی مثل حالت Weak Entity است.

مثال:



EMP(E#,Ename,…)

ETEL(E#,PHONE) → E# : FK جزء کلید

EDHIS(E#,Title,From,To,…) → E# : FK

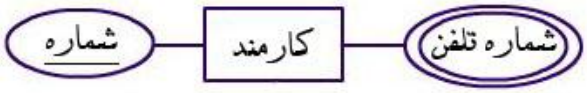
در مورد رابطه EDHIS ممکن است صفات دیگری هم ممکن است جزء کلید باشد. مثل وقتی که یک نفر در دو تاریخ مختلف بیماری بگیرد.

معایب تکنیک ۲ : اگر اطلاعات کامل بیمار را بخواهیم، باید رابطه ها را JOIN کنیم و عمل join در حالت کلی بسیار زمان گیر است .

تکنیک های خاص: (برای یک صفت چند مقداری برای بیش از یک صفت توصیه نمی شود)

- فقط یک رابطه برای نوع موجودیت و خود صفت :
  - تکنیک ۱: تمام صفات نوع موجودیت + خود صفت : کلیدمرکب یعنی خود صفت : جزء کلید
  - تکنیک ۲ : تمام صفات نوع موجودیت و به تعداد حد اکثر تعداد مقدار که صفت می گیرد صفت در نظر گیریم: (کلید ساده) کلید وصفت چند مقداری دیگر جزء کلید نمی شو د.

مثال: (حالت۲)



فرض: یک کارمند حداکثر ۳ تلفن دارد:

ETELPHO(E#,…..,Phone1,Phone2,Phone3)

100	N1	N2	?
101	N3	?	?
102	N4	N5	N6

از این تکنیک وقتی استفاده می کنیم که :

حداکثر تعداد کم و مشخص باشد.

توزیع مقدار کم و بیش یکنواخت باشد.(زیاد NULL نگیرد)

تکنیک سوم :

مثال:

ETELPHON(E#,Ename,…,Phone)

100	e1	….	N1
100	e1	….	N2
101	e2	….	N2

همانگونه که در داده های بالا می بینیم در این تکنیک افزونگی بسیار بالاست.

از این تکنیک وقتی استفاده می کنیم که :

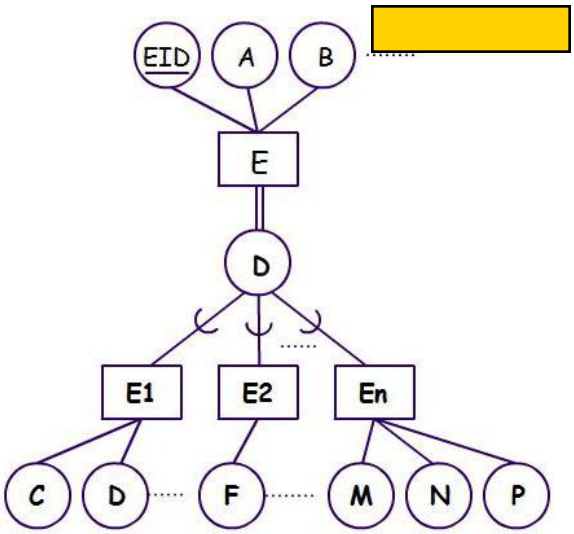
۱- مجموعه صفات نوع موجودیت کم باشد.

۲- توزیع ترجیحاً یکنواخت باشد.

۳- تعداد مقداری که صفت چند مقداری می گیرد کم باشد.

ابتدای صدای ۳ جلسه هشتم

چهار تکنیک دارد: یک تکنیک عمومی و سه تکنیک خاص



نوع موجودیت E، n زیرنوع دارد

تکنیک عمومی : طراحی با n+1 رابطه:

- یک رابطه برای زبر نوع
- یک رابطه برای هر زیر نوع

کلید در تمام این رابطه ها یکسان است:

ERel(EID,A,B,…)
E1R(EID,C,D)
E2R(EID,F)
…
EnR(EID,M,N,P)

در رابطه زبرنوع شناسه و صفات مشترک می آید

در رابطه مربوط به زیر نوع شناسه و صفات خودش می آید.

کنجکاوی: درچه وضع دیگری بیش از یک رابطه کلید یکسان دارند ؟ (در طراحی RDB)

۱. وقتی ارتباط IS-A داریم

۲. ؟

مزیت تکنیک عمومی

✓ شرط خاصی لازم نیست: از نظر نوع تحضیص (تخصیص اگر کامل یا ناقص باشد جواب می دهد)

عیب تکنیک عمومی

✓ اگر اطلاعات کامل در مورد نمونه هایی از یک زیر نوع بخواهیم ، باید عمل پیوند انجام شود .

تکنیک ۲:

طراحی با n رابطه

دراین تکنیک برای زیر نوع رابطه طراحی نمی کنیم . برای هر زیر نوع یک رابطه .

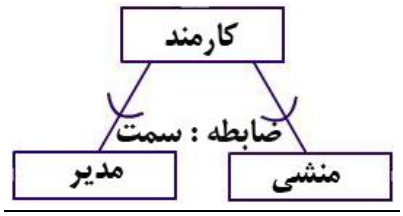
صفات مشترک در همه رابطه ها می آید:

E1R(EID,A,B,C,D)
E2R(EID,A,B,F)
…
EnR(EID,A,B,M,N,P)

شرط استفاده از این تکنیک چیست؟

تخصیص باید کامل باشد. اگرناقص باشد، بخشی از داده ها ی محیط قابل نمایش نیستند .

مثلا در شکل زیر تخصیص ناقص است و همه انواع کارمند را پوشش نمی دهد.



سئوال: آیا الزامی است که تخصیص مجزا باشد ؟

الزامی نیست ولی بهتر است تخصیص مجزا باشد تا افزونگی پدید نیاید.

نکته : صرف این که در heading دو یا بیش از دو رابطه صفات مشترک وجود داشته باشند، لزوماً به معنای ایجاد افزونگی در DB نیست، باید دید وضع داده ها چگونه اند. مثال:

R1:

A	B
A1	B1
A2	B2
A3	B3

R2:

C	B
C1	B7
C2	B8
C3	B9

✓ صرف این که دو یا چند جدول ستون تکراری داشته باشند ، لزوماً به معنای افزونگی نیست .

تکنیک ۳ :

طراحی با یک رابطه و با استفاده از یک صفت نوع (type Attribute)

مثال:

100 a1 b1 c1 d1 ? … ? ? ? → مدیر

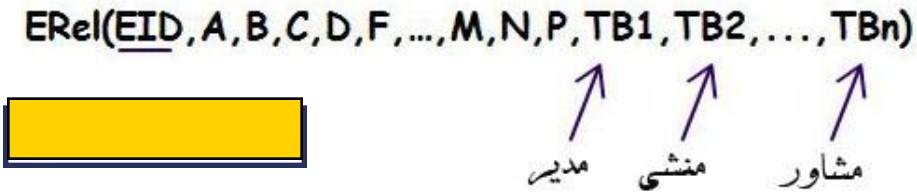
۲۰۰ a2 b2 ? ? ? … m2 n2 p2 → مشاور

هنگامی از این تکنیک می توانیم استفاده کنیم که تخصیص مجزا باشد. و بهتر است تعداد شاخه های تخصیص کم باشد(برای کاهش NULL). همچنین تعداد صفات هر زیرنوع کم باشد.

تکنیک ٤:

طراحی با یک رابطه و با استفاده از بیت نمایشگر نوع "دریک آرایه"

شبیه تکنیک سوم است اما به جای یک صفت نوع n بیت می گیریم برای نمایش نوع (نوع نمونه زیر نوع ها)



معنای هر صفت هم برای طراح پیاده ساز مشخص است.

فرض کد ۳۰۰ هم مدیر است هم مشاور و ۲۰۰ منشی است: حال می آییم بسته به اینکه یک نمونه زیرنوع از کدام نوع باشد آن بیت را با منطق مثبت مقدارگذاری می کنیم.

300	a3	b3	c3	d3	?	m3	n3	p3	1	0	…	0	1
200	a2	b2	?	?	f2	?	?	?	0	1	…	0	0

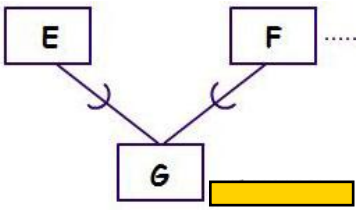
شرط استفاده از این تکنیک چیست؟

وقتی تخصیص هم پوشا باشد

سایر شرایط ارجحیت همان هاست که در تکنیک سه گفته شده.

نکته: محدویتهای جامعیتی محیط برخی در سطح مدلسازی قابل نمایش هستند، برخی دیگر در سطح طراحی منطقی و برخی دیگر درمرحله پیاده سازی.

تمرین – وراثت چندگانه طراحی شود:

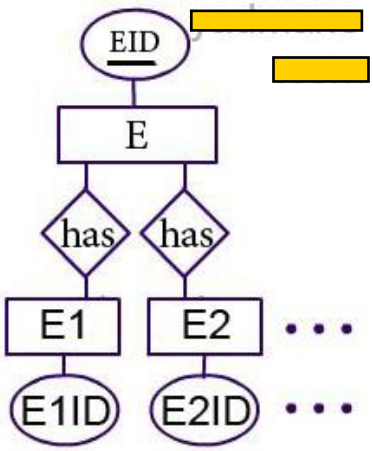


۳۳:۲۰

حالت هفتم: ارتباط IS_A_PART_OF

مثل حالت نوع ضعیف عمل می شود.

رابطه نمایشگر نوع جزء از رابطه نمایشگر نوع کل FK می گیرد جزء کلید.



طراحی شبیه weak است.

EREL(EID,…)
EIR(EID,E1ID,…)
…

حالت هشتم : ارتباط سلسله مراتبی

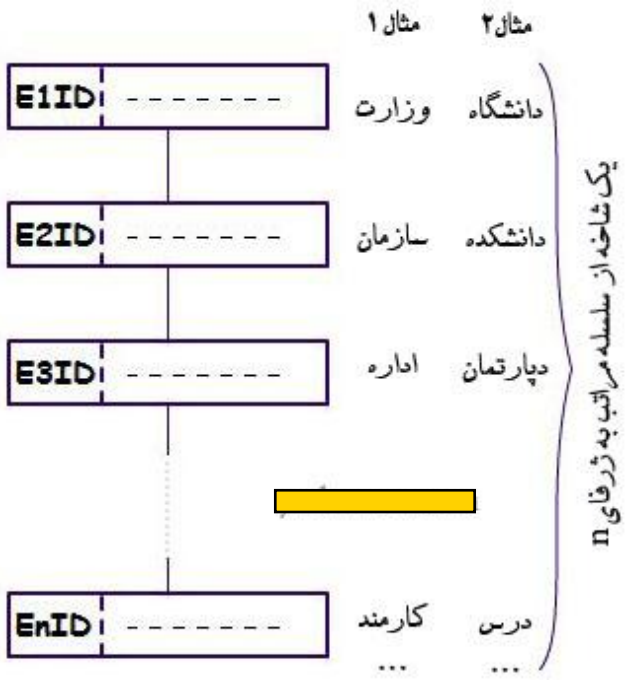
بین انواع رکوردها (هرنوع رکورد نمایشگر یک نوع موجودیت در HDS)

در واقع این تکنیک : تکنیک تبدیل طراحی با HDS به طراحی با RDS است.

کاربرد در تبدیل سیستم های گذشته (legacy) به سیستم جدیدتر

در HDS هرنوع موجودیت با یک نوع رکورد نمایش می دهیم.

نمایش سلسله مراتبی خاص ارتباط های 1:N است.



تکنیک طراحی: برای هر نوع رکورد یک رابطه طراحی می کنیم.

رابطه نمایشگر نوع رکورد سطح i ام به رابطه نمایشگر سطح (i+1)ام FK می دهد، جزء کلید.(پدر به فرزند FK می دهد(جزء کلید) تا معلوم نشود نمونه فرزند نمونه کیست)

E1R(E1ID ,…)
E2R(E1ID,E2ID ,…)
E3R(E1ID,E2ID,E3ID ,…)
…

نکته : این تکنیک آن عیب رابطه نرمال را نشان می دهد: دشواری دارد در نمایش ارتباط سلسله مراتبی (مربوط به مرحله بیت).

تست : دراین طراحی چند FK پدید می آید؟

پاسخ:

N(N-1)/2

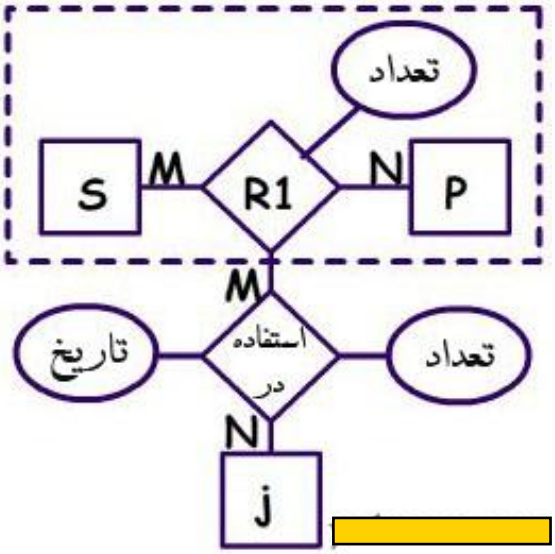
تست : در این طراحی گراف ارجاع چند یال دارد؟

تست : در این طراحی چند حلقه در گراف ارجاع پدید می آید؟

۵۵:۰۰

حالت نهم : تکنیک تجمیع (Aggregation)

ابتدا آن بخش از نمودار داخل مستطیل را طراحی می کنیم و سپس بخش بیرونی را طراحی می کنیم(یعنی ارتباط با ارتباط را) و در هر مورد با توجه به درجه و چندی ارتباط .



S(S#,...)

P(P#,...)

SP(S#,P#,QTY)

ابتدا داخل مستطیل را طراحی می کنیم

J(J#,...)

(استفاده در)

SPJ(S#,P#,J#,DATE,QTY)

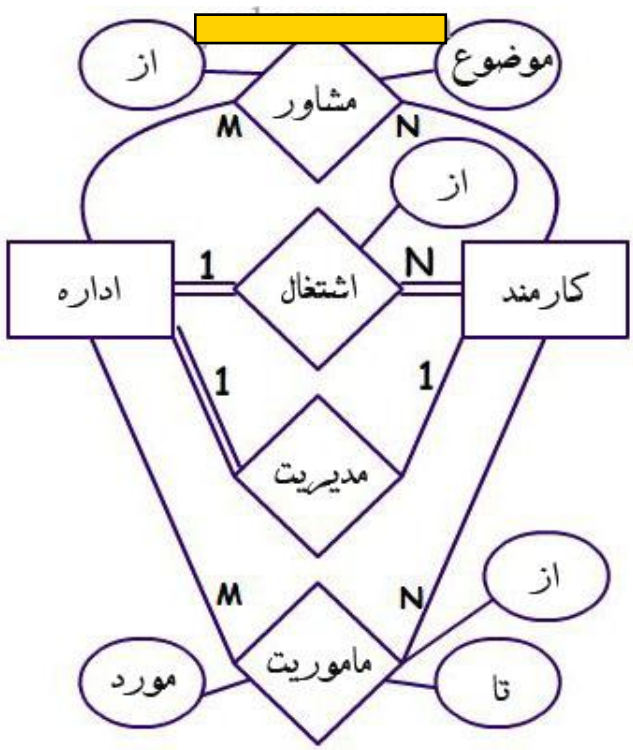
تست: در تبدیل شکل بالا چند FK پدید می آید؟

تست : در تبدیل شکل بالا چند FK مرکب پدید می آید؟

۱:۰۳:۰۰

حالت دهم : چند ارتباط بین مثلا دو نوع موجودیت

تکنیک : هر ارتباط با توجه به چندی اش طراحی می شود امّا توصیه می شود ابتدا ارتباط های با چندی M:N و سپس 1:N و سپس 1:1 را طراحی کنید (برای کاهش ریسک اشتباه)



DEPT(D#,...,E#)
شماره مدیر اداره

EMP(E#,...,D#)
موضوع-شماره اداره

DEMOSH(E#,D#,MOZOE,FROM,TO,...)
مشاور

DEMA(E#,D#,MORED,FROM,TO,...)
ماموریت

سئوال : در اینجا دو ارتباط M:N داریم، آیا می توانیم این دو را یکی بکنیم؟

این کار باعث ایجاد NULL ، افزونگی و اخطلات در معانی می آورد.

تست : نمودار بالا مفروض است حداکثر با چند رابطه طراحی می شود؟(شرط طراحی خوب مفروض است)

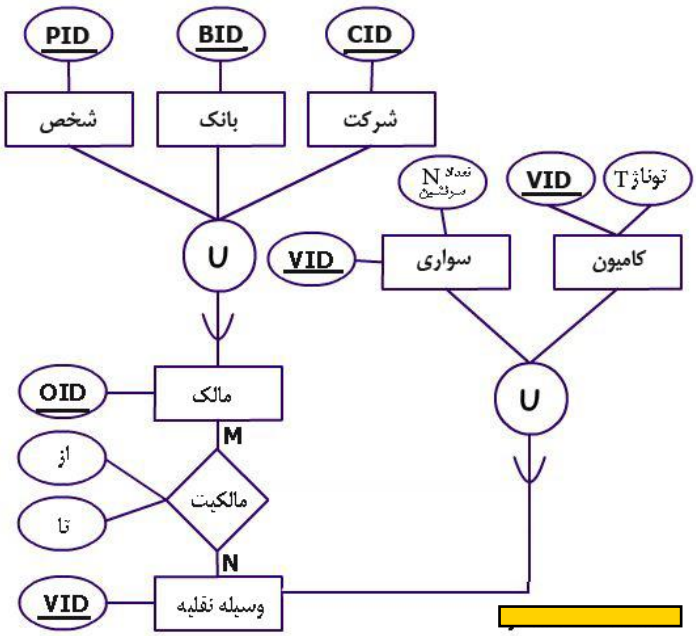
لوزی ها + مستطیل ها=٦

تست : اگر روی یا لها چندی را ندهند حداکثر با چند رابطه طراحی می شود؟

اگر چندی ارتباط داده نشده باشد N:M در نظر گرفته می شود.

تست : اگر روی یا لها چیز را ندهند حداقل با چند رابطه طراحی می شود؟

حالت یازدهم : U-Type



تکنیک :

١. برای هر زیر نوع یک رابطه (در این مثال ٥ تا)
٢. هر U-Type یک رابطه (در این مثال ٢ تا)
٣. اگر ارتباط داشته باشیم طبق چندی و درجه اش (در این مثال یکی)
٤. (خیلی مهم) در حالتی که شناسه زیر نوع ها متفاوت باشند رابطه نمایشگر U-Type به رابطه نمایشگر زیر نوع FK می دهد .

PERS(PID,…,OID)	شخص ⇦
BANK(BID,…,OID)	بانک ⇦
COMP(CID,…,OID)	شرکت ⇦
OWNER(OID,…)	مالک ⇦
KHODRO(VID,…)	خودرو ⇦
OWNS(OID,VID,FROM,TO,…)	مالکیت ⇦
SAVARI(VID,N,…)	سواری ⇦
BARI(VID,T,…)	باری ⇦

نکته: طبق تکنیک ٤، OID به زیرِنوع ها (رابطه اول تا سوم) داده شده.

کنجکاوی : چرا رابطه نمایشگر زیرِنوع به رابطه نمایشگر U-Type ، FK ندهد؟ (چرا مالک FK بدهد ، چرا برعکس نمی شود؟)

ممکن است زیرِنوع ها زیاد باشد و یا همه مالک نباشند.

جلسه نهم شنبه ۱۳۸۸/۵/۳ ساعت ۷

نرمالتر سازی رابطه ها

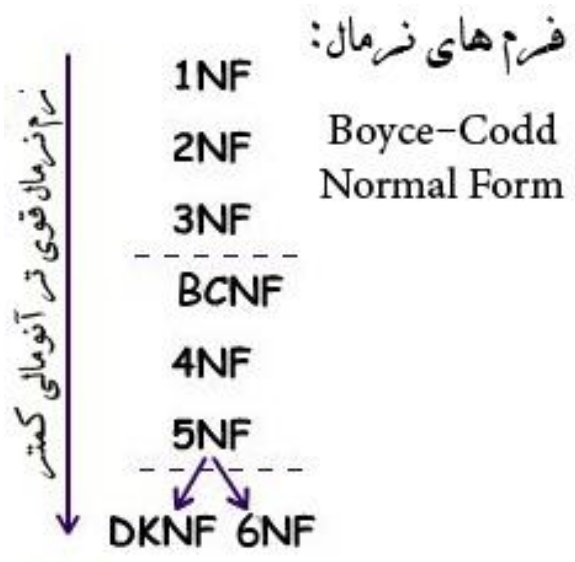
طراحی RDB (روش سنتز)

ایده اصلی: رابطه، هر چند نرمال، می تواند آنومالی در عملیات ذخیره سازی داشته باشد.

آنومالی به معنی : مشکل ، وضع نامطلوب و دشواری ...

نرمال بودن (نرمالیسیتی) درجات، صور، سطح و فرم هایی دارد.

فرم های نرمال عبارتند از:



نکته:

$$\text{6NF,DKNF} \subset \text{5NF} \subset \text{4NF} \subset \text{BCNF} \subset \text{3NF} \subset \text{2NF} \subset \text{1NF} \subset \text{All Relation}$$

هر 2NF حتما 1NF هست اما هر 1NF لزوما 2NF نیست و ...

مفاهیمی از تئوری وابستگی

وابستگی تابعی FD (بستگی وظیفه مندی- Functional Dependency)

تعریف : در رابطه R(A,B,C,...) صفت B باصفت A وابستگی تابعی دارد، هر گاه بازای یک مقدار از A فقط یک مقدار B متناظر باشد. این معنا را چنین نمایش می دهیم:

$A \rightarrow B$

و می گوییم:

B وابستگی تابعی دارد به A

A ، B را تعیین می کند.

مثال :

R(A , B , C)
a1,b1,c1
a1,b1,c2
a2,b1,c3
a3,b3,c3

آیا داریم $A \rightarrow B$ ؟ بله

آیا داریم $B \rightarrow A$ ؟ نه چون b1 به دو مقدار a1,a2 می رود.

آیا داریم $A \rightarrow C$ ؟ نه چون a1 به دو مقدار c1,c2 می رود.

نکات :

۱. طرفین FD می توانند صفات ساده باشند یا مرکب

مثل:

$A \rightarrow B$
$(x,y) \rightarrow z$

۲. اگر $B \subseteq A$ وابستگی تابعی $A \rightarrow B$ را نامهم(بدیهی- نامطرح-Trivial) گوییم. (این وابستگی ها در طراحی مد نظر نیستند)

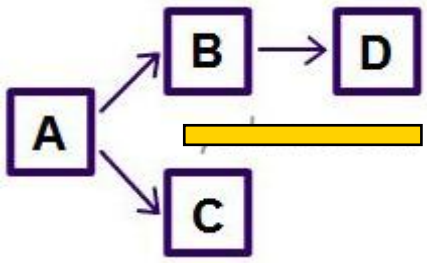
۳. اگر k [سوپرکلید/ کلیدکاندید/ کلید اصلی/ کلید بدیل] باشد، در این صورت داریم:

$K \rightarrow G \mid G \subseteq H_R$

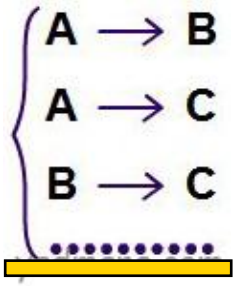
از کلید به همه فلش می زنیم. یعنی همه به کلید وابستگی تابعی دارند چون Unique است.(یکتایی مقادیر کلید)

نمایش FD های رابطه R

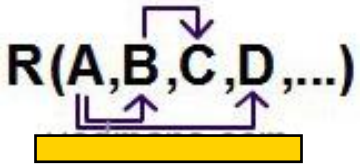
۱. با FDD : نمودار FD ها



۲. با مجموعه F

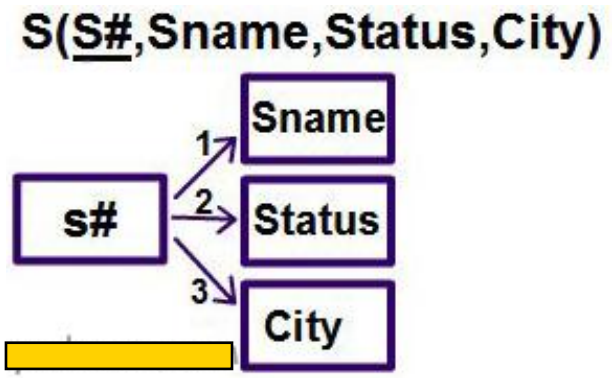


۳. در خود رابطه با فلش ها



تفسیر یا معنای FD : هر FD نمایشگر یک قاعده یا محدودیت معنایی از محیط است که باید به نحوی به سیستم داده شود و درطول حیات رابطه رعایت شود. (یک نوع قاعده جامعیت است)

مثال:



۱. یک شماره به یک تهیه کننده داده می شود.

۲. یک تهیه کننده یک مقدار وضعیت دارد.

۳. یک تهیه کننده دریک شهر ساکن است.

نکته : قواعد آرمسترانگ برای FD ها: (صفحات ضمیمه DY1 تا DY4)

۱- قاعده انعکاس:

If $B \subseteq A$ Then $A \rightarrow B$, $A \rightarrow A$

۲- قاعده متعدی(تراگذری):

If $A \rightarrow B$ And $B \rightarrow C$ Then $A \rightarrow C$

۳- قاعده افزایش:

If $A \rightarrow B$ Then $(A,C) \rightarrow (B,C)$

٤- قاعده تجزیه:

If $A \rightarrow (B,C)$ Then $A \rightarrow B$ And $A \rightarrow C$

٥- قاعده ترکیب:

If $A \rightarrow B$ And $C \rightarrow D$ Then $(A,C) \rightarrow (B,D)$

٦- قاعده اجتماع:

If $A \rightarrow B$ And $A \rightarrow C$ Then $A \rightarrow (B,C)$

۷- قاعده شبه تعدی:

If $A \rightarrow B$ And $(B,C) \rightarrow D$ Then $(A,C) \rightarrow D$

۸- قاعده یگانگی عمومی: (توسط Hugh Darwen اثبات شده است.)

If $A \rightarrow B$ And $C \rightarrow D$ Then $A \cup (C-B) \rightarrow (B,D)$

بیشتر قواعد آرمسترانگ (۱تا ۷) حالت خاصی از قاعده ۸ می باشند.

نکته: قواعد ۱و۲و۳ استوار کامل هستند، به این معنا که با داشتن یک مجموعه از وابستگی های تابعی F ، تمام وابستگی تابعی منطقا قابل استنتاج از F ، با همین سه قاعده بدست می آیند و هیچ وابستگی تابعی دیگر (که قابل استنتاج از F نباشد) نیز بدست نمی آید.

توجه : سه قاعده اول به آسانی قابل اثبات اند و قواعد دیگر از روی همان ها اثبات می شوند.

کاربرد های دیگر قواعد آرمسترانگ:

۱- یافتن بستار یک صفت : A^+ (تمامی صفاتی که با A ، FD دارند)

۲- یافتن بستار یک مجموعه از وابستگی های تابعی: F^+

۳- یافتن مجموعه کاهش ناپذیر وابستگی های تابعی یک رابطه (مجموعه بهینه)

٤- یافتن پوش کانونیک

٥- تشخیص خوب بودن تجزیه

٦- تشخیص FD افزونه

کاربرد A^+ :

- ✓ تشخیص سوپر کلید ها

نکته : اگر $A^+=H_R$ باشد آنگاه A سوپر کلید است (اما لزوما CK نیست چون CK کاهش ناپذیر است)

- ✓ تشخیص عضویت یک FD مثلا f در F

If $f \in A^+$ Then $f \in F$

تعریف F^+ :

مجموعه تمام FD هایی که از F منطقا استنتاج می شوند.

مثال:

$$F=\begin{cases} A \rightarrow B \\ B \rightarrow C \end{cases} \Rightarrow F^{+}=\begin{cases} A \rightarrow B \\ A \rightarrow C \\ B \rightarrow C \end{cases}$$

کاربرد F^{+} :

۱- تعریف بعضی فرم های نرمال

۲- در تشخیص معادل بودن دو مجموعه از FD های رابطه R

۳- درتشخیص FD افزونه

شرط معادل بودن دومجموعه F و G که دو مجموعه از FD های رابطه R هستند این است که:

$G^{+}=F^{+}$

یعنی لزومی ندارد FD ها یکسان باشد بلکه آن که از F نهایتا به دست می آید باید همان باشد که از G به دست می آید.

نکته : در صورتی که F و G معادل بودند لزومی ندارد هر دو را در طراحی دخالت دهیم.

تعریف FD افزونه:

وابستگی $f \in F$ افزونه است، هرگاه :

$(F-f)^{+} = F^{+}$

اگر آن را از F خارج کنیم در F^{+} تاثیری نداشته باشد.

۴۵:۰۰

تعریف مجموعه ی کاهش ناپذیر FD های رابطه ی R : (مجموعه کهینه-Minimal Set)

مجموعه ای که :

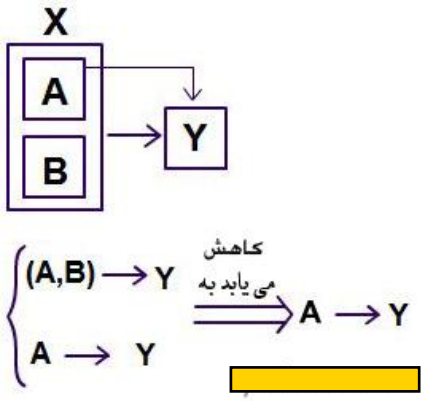
۱- هیچ FD در آن افزونه نباشد.

۲- سمت راست هر FD، صفت ساده باشد.

۳- سمت چپ هر FD، خود کاهش ناپذیر باشد.

✓ در $X \rightarrow Y$ ، X کاهش ناپذیر است، هرگاه: Y با هیچ زیرمجموعه از X غیر از خودش وابستگی تابعی نداشته باشد.

مثال: اگر داشته باشیم :



X کاهش پذیر است زیرا از A→Y داریم : (A,B)→Y

اثبات:

فرض داریم A→Y

طبق قاعده افزایش: (A,B)→(Y,B)

طبق قاعده تجزیه بدیهی است : (A,B)→B و (A,B)→ Y : رسیدیم به حکم

نکته : قاعده بالا را قاعده چپ افزایی نیز می گویند.

۵٤:۰۰

تعریف FD کامل (تام) Fully Functional Dependency

X→Y را تام گوییم هر گاه X کاهش ناپذیر باشد. و گرنه FD ناتام (ناکامل) است.

اگر سمت چپ FD، صفت ساده(Composite) آن FD خودبخود کامل است.

تعریف TFD:

FD با واسطه (تراگذری) TRANSITIVE:

اگر A→B & B→C آنگاه می گوییم صفت C با A، TFD دارد از طریق B.

اگر B→A نیز داشته باشیم، آن TFD نامهم (بدیهی) است. (دیگر در طراحی دخالت نمی دهیم)

نکته:

- ✓ مفهوم FD ناکامل در تبدیل 1NF به 2NF مطرح است.
- ✓ مفهوم TFD در تبدیل 2NF به 3NF مطرح است.

۱:۰۱:۳۰

شرح فرم های نرمال:

تعریف 1NF:

رابطه R در یک 1NF است اگر و فقط اگر تمام صفات آن تک مقداری باشند.

- ✓ هر رابطه نرمال در 1NF است.

چنین رابطه ای می تواند آنومالی داشته باشد. در این صورت باید نرمال تر شود.

نکته: در فرم های نرمال ۱و۲و۳ با مفهوم PK کار می کنیم.

به فرمهای نرمال ۱ و ۲ و ۳ فرم های کادی گویند.

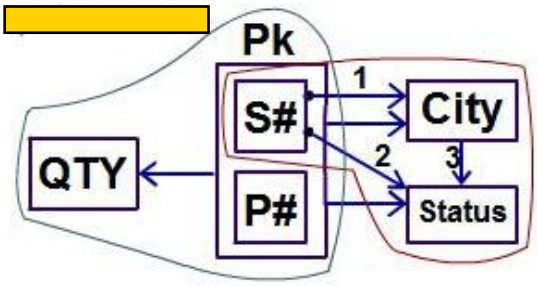
مثال:

قواعد معنایی این محیط:

- یک تهیه کننده در یک شهر ساکن است.
- یک تهیه کننده یک مقدار وضعیت دارد.
- تمام تهیه کنندگان ساکن یک شهر یک مقدار وضعیت دارند.(در این مثال)

FIRST(S#,P#,City,Status,QTY)

S#	P#	City	Status	QTY
S1	P1	C1	10	100
S1	P2	C1	10	80
S1	P3	C1	10	90
S2	P1	C2	15	80
S2	P2	C2	15	90
S3	P1	C2	15	70
S4	P1	C1	10	50
S4	P2	C1	10	90



رابطه FIRST در 1NF است(تمام صفات تک مقداری هستند) و آنومالی دارد.

در این نمودار:

- ✓ دو صفت عمده داریم (۱)
- ✓ سه صفت ناکلید داریم.(۲)

آنومالی های مربوط به رابطه FIRST:

۱- در درج :

درج کن این اطلاع را <S5,C5,20>

درج ناممکن است چون کلید S#P# است و برای درج تاپل حتما باید یکی از قطعاتی که S5 تهیه کرده را بدانیم. یعنی نیاز به P# داریم.

۲- در حذف:

حذف کن فقط این اطلاع را <S3,P1,70>

حذف انجام می شود اما اطلاع ناخواسته هم حذ ف می شود.

۳- در بهنگام سازی:

شهر S1 را عوض کنید.

برای عوض کردن S1 تعداد زیادی سطر باید بهنگام سازی شود یعنی عمل منطقا تاپلی تبدیل می شود به عمل مجموعه ای و این یعنی over head در سیستم. (بروز فزونکاری در سیستم)

آنومالی سه جنبه دارد:

۱- عدم امکان درج

۲- حذف اطلاع ناخواسته

۳- فزون کاری در انجام عمل بهنگام سازی

1NF باید نرمالتر شود:

الگوریتم عمومی نرمالتر سازی تجزیه عمودی رابطه به N≥2 رابطه. به طور مناسب یعنی رابطه های حاصل از تجزیه آنومالی های رابطه اولیه را نداشته باشند.

تجزیه عمودی با عملگر پرتو(project) انجام می شود.

سئوال: یک رابطه درجه n چند پرتودارد؟

2^n-1

برای اینکه بدانیم چگونه باید تجزیه کنیم ابتدا باید دلیل آنومالی های رابطه First را بدانیم.

دلیل آنومالی های رابطه FIRST:

۱- از نظر عملی: پدیده اختلاط اطلاعات

در یک جدول اطلاعات در مورد تهیه کننده و عمل تهیه می شود را اختلاط کرده ایم.(این دلیل از نظر تئوری پذیرفته نیست. چون رابطه می تواند اختلاف اطلاعات داشته باشد و مثلا 3NF هم باشد.)

۲- در تئوری: وجود FD های ناکامل (ناتام) بین صفات ناکلید و کلید اصلی

(S#,P#)→Status	**(S#,P#)→City**
S# → Status	**S# → City**

این FD های ناکامل باید از بین برود.

بنابراین برای تبدیل 1NF به 2NF باید FD ناکامل با کلید را از طریق پرتوگیری از بین ببریم.

تعریف صفت ناکلید:

در این بحث صفت ناکلید یعنی خود کلید نباشد و جزء کلید هم نباشد.

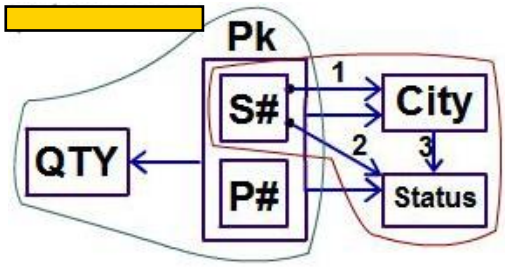
به صفت جزء کلید اصطلاحا صفت عمده می گوییم. Prime Attribute

سئوال: در مثال قبل چند صفت ناکلید داریم؟

پاسخ : ۳ صفت ناکلید داریم.

۱:۳۰:۰۰

رابطه را به شکل زیر تجزیه می کنیم و منشا FD های نا کامل را جدا می کنیم:



رابطه به دو رابطه SECOND و SP تجزیه می شود: این دو پرتو های FIRST هستند.

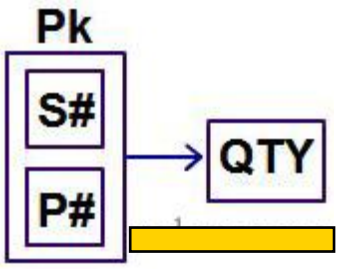
SECOND(S#,City,Status)

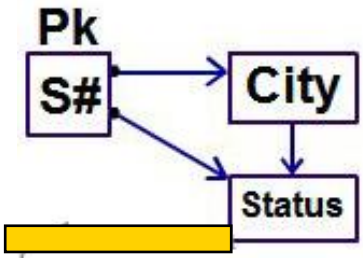
S#	City	Status
S1	C1	10
S2	C2	15
S3	C2	15
S4	C1	10

SP(S#,P#, QTY)

S#	P#	QTY
S1	P1	100
S1	P2	80
S3	P1	70
S4	P2	90
…	…	…

وقتی که در لیست پرتو کلید رابطه باشد همان تعداد تاپل بوجود می آید ولی با صفات پرتو.





به آسانی می توان دریافت که رابطه های حاصله آنومالی FIRST را ندارند.

برای اطمینان از این موضوع حذف می کنیم <S3,P1,70> اطلاعات خود S1 حذف نمی شود.

همچنین برای درج <S5,C5,20> مشکلی پیش نمی آید و دیگر نیازی نیست یکی از قطعاتی که تولید کرده را بدانیم.

همچنین برای به هنگام سازی شهر S1 عمل تاپلی انجام می شود.

در نتیجه SECOND و SP آنومالی های FIRST را ندارند پس از FIRST نرمالترند.

SECOND و SP در 2NF هستند.

تعریف 2NF

رابطه R در 2NF است، اگر و فقط اگر در 1NF باشد و هر صفت ناکلید، با کلید اصلی FD تام (کامل) داشته باشد.

1:42:00 -1:52:00 Break

در این مثال داریم:

$$SECOND \bowtie SP=FIRST$$

دلیل : این پیوند از نوع FK−PK است. به دلیل یکتایی مقادیر PK بازسازنده است. یعنی تمام تاپل های رابطه اولیه به دست می آید (با پیوند)، نه تاپلی از دست می رود و نه تاپل اضافی (حشو) پدید می آید. این مسئله عمومیت ندارد و ممکن است تاپل اضافی بروز کند.

نکته : اما در حالت کلی :

اگر $R_1,R_2,R_3,\ldots,R_n$، n پرتو دلخواه از R باشند . داریم :

$$R \subseteq R1 \bowtie R2 \bowtie \ldots. \bowtie Rn$$

شرط بر جایی این فرمول است که صفات پیوند null نداشته باشند . (صرف نظر از آن نکته مهم که Date می گوید: که رابطه اساسا نمی تواند null داشته باشد)

کنجکاوی: آیا در این تجربه تمام FD های رابطه ی FIRST محفوظ می مانند؟

پاسخ : بعد از تجربه داریم :

$$\begin{cases} S\# \rightarrow City \xrightarrow{\text{قاعده چپ افزایی}} \begin{cases} (S\#,P\#) \rightarrow City \\ (S\#,P\#) \rightarrow Status \end{cases} \\ S\# \rightarrow Status \\ City \rightarrow Status \\ (S\#,P\#) \rightarrow QTY \end{cases}$$

در این تجربه هم اطلاعات حفظ شدند و هم تمام FD ها محفوظ ماندند.

شرح 3NF:

سئوال: آیا رابطه هاس جدید آنومالی ندارند؟

رابطه SECOND هنوز آنومالی دارد.

رابطه SP مشکل ندارد.

آنومالی SECOND کدام است؟

۱- در درج : درج کن این اطلاع را :

<C16,16>

با توجه به قاعده ۳ منطقا باید بتوان این فقره اطلاع را درج کرد اما درج ناممکن

۲- درحذف : حذف کن :

<S5,C5>

وفرض می کنیم S5 در این لحظه تنها تهیه کننده ساکن این شهر باشد. این حذف منجر به حذف اطلاع نا خواسته می شود. وضعیت داده شده به ساکنان <C5,20> است.

۳- در بهنگام سازی :

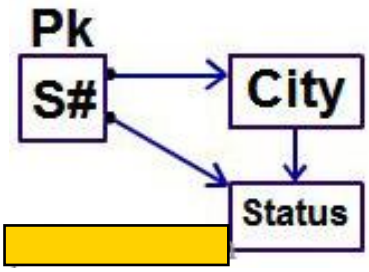
وضعیت ساکنان C1 را عوض کنید.

در اینجا تعداد تاپل ها که است اگر ۲میلیون رکورد داشته باشیم باید تعداد زیادی تاپل را به هنگام سازی کنیم. در نتیجه second باید نرمالتر شود. کماکان با تجزیه عمودی از طریق پرتو گیری مناسب .

برای این که طرز تجزیه را بدانیم باید ببینیم دلیل آنومالیهای SECOND چیست ؛

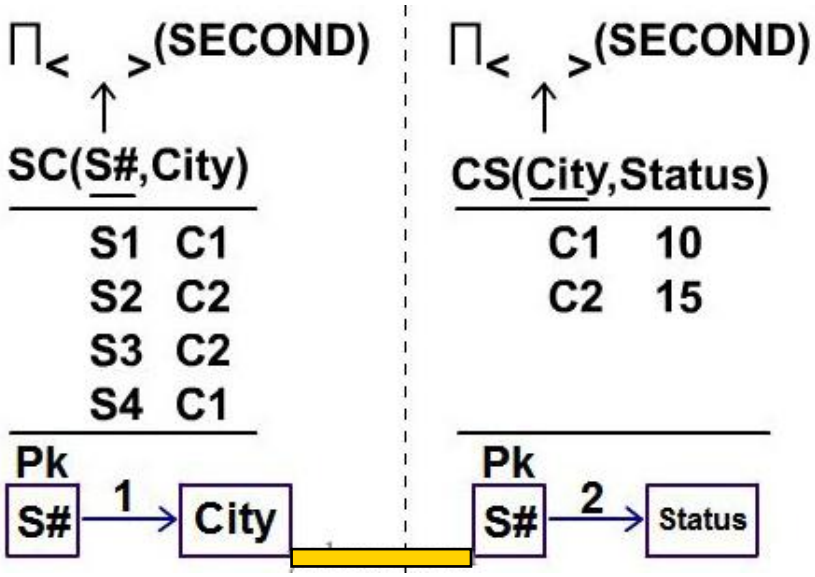
دلیل آنومالی SECOND:

وجود TFD یعنی FD با وابسته بین صفات نا کلید و کلید اصلی



یعنی STATUS نه تنها FD مستقیم دارد، با #S به طور Transitive هم FD دارد.این FD با واسطه باید از بین برود.

بنابرین SECOND را روی محور وابستگی با واسطه تجربه می کنیم.



در این تجزیه TFD را حذف کردیم.

SC و CS از SECOND نرمال ترند، چون آنرمالی ها SECOND را ندارند .

تمرین : صحت این ادعا که SC و CS از SECOND نرمال ترند بررسی شود:

درج کن :

<C6,16>

درج مشکلی ندارد.

حذف کن :

<C5,20>

حذف نیز بدون مشکل انجام می شود.

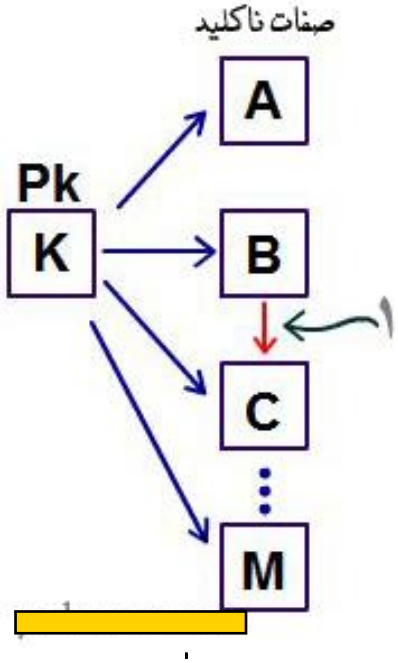
توجه دارید که در اینجا به دلیل پیوند Pk-Fk داریم:

$$SECOND = SC \bowtie CS$$

Status → S# طبق قاعده تعدی از 1 و ۳ بدست می آید.

تعریف 3NF: (تعریف کلاسیک) [تعریف کاد]

رابطه R در 3NF است اگر و فقط اگر اولا در 2NF باشد، ثانیا هر صفت ناکلید با کلید اصلی FD بی واسطه داشته باشد.



در صورتی که (۱) وجود داشته باشد رابطه دیگر در 3NF نیست.

تعریف 3NF و شکل بالا می گوید در رابطه 3NF هر چه FD داریم ناشی از PK است.

2:22:00

تعریف دوم 3nf (با استفاده از مفهوم SK):

رابطه R با مجموعه FD هایش در 3NF است ، اگر و فقط اگر بازای هر FD در F^+ به صورت $X \rightarrow A$ یکی از سه شرط زیر برقرار باشد:

۱. آن FD نامهم باشد.

۲. X خود سوپر کلید باشد.

۳. A جزء کلید (صفت عمده) باشد.

ضوابط و شرایط تجزیه خوب چیست ؟ (نادقیق) ح-ح-ح-ح

تجزیه بی کاست-بی گمشدگی-Non Loss Decomposition

۱. با پیوند پرتوها خود رابطه اولیه به دست آید : بی حشو باشد

۲. تمام FD ها برجا باشند (حفظ شوند) تجزیه حافظ FD ها باشد.

۳. با پیوند پرتو ها هیچ تا پلی ازدست نرود : بی حذف باشد.

٤. تمام صفات رابطه ای اصلی حفظ شوند: حافظ صفات باشد. یعنی :

$HR=HR_1 \cup HR_2 \cup \ldots \cup HR_n$

اکثر کتاب ها ۳ و ٤ را پیش فرض گرفته اند.

✓ از دیدگاه تئوریک تجزیه R به دو پرتو R1 و R2 خوب است. هرگاه پرتوها در یکدیگر مستقل باشند.

پرتوهای مستقل

پرتوهای مستقل R1 و R2 حاصل تجزیه R مستقلند هرگاه شرایط قضیه ریسانن را داشته باشد:

۱- صفت مشترک در R1 و R2: حداقل در یکی از آن ها CK باشد. (تضمین می کند پیوند بازسازنده می شود:-Ck Fk) : بی حشو بودن

۲- تمام FD های R یا در R1 و R2 وجود داشته باشند، یا از آن ها منطقا استنتاج شوند. (حافظ FD)

بر این اساس تجزیه SECOND به دو پرتو SC و CS خوب است. اما اگر SECOND را چنین تجزیه کنیم:

CS(S#,Status)
SC(S#,City)

آیا این تجزیه خوب است؟

نه – شرایط را بررسی می کنیم:

۱- شرط اول را دارد: صفات مشترک درهر دو CK است.

۲- شرط دوم را ندارد : FD از دست می دهد.(City→Status را از دست می دهد.)

کنجکاوی: تجزیه سوم کدام است و چه وضعی دارد؟

نکته: در رابطه R(A,B,C) اگر A→B→C آن گاه تجزیه R1(A,B) و R2(B,C) خوب است. (Non-Loss است)

قضیه Heath :

رابطه R(A,B,C) به دو پرتوش R1(A,B) و R2(A,C) تجزیه بی کاست می شود اگر داشته باشیم : A→B

عکس قضیه Heath صادق نیست .

نکته : تعریف رابطه ATOMIC (تجزیه ی نشدنی):

رابطه ای که به پرتو های مستقل تجزیه نشود (و چنین رابطه ای نباید هم تجزیه شود.)

2:51:00 -2:56:00 Break

شرح BCNF :

تمرین : قاعده به یک سِمت یک مقدار حقوق تعلق می گیرد.

EMP(E#,Job,SAL)

در این رابطه E# شماره ، Job سمت و SAL حقوق می باشد.

این رابطه در کدام فرم نرمال است؟ 2NF

قطعه برنامه ای بنوسید که میانگین حقوق ها را بدهد.

نکته : در A→B به A دترمینان می گوییم.

تعریف : رابطه R در BCNF است اگر و فقط اگر هر دترمینان در آن CK باشد. یعنی هر چه FD در رابطه داریم ناشی از CK ها باشد.

✓ BCNF قوی تر از 3NF است .

نکته : رابطه ای که در BCNF باشد در 3NF هم هست اما عکس مطلب همیشه صادق نیست.

برای بررسی صحت این مطلب دو حالت در نظر گیریم :

حالت اول : رابطه R فقط یک CK داشته باشد (می شود همان PK)، در این حالت اگر R در 3NF باشد در BCNF هم هست .

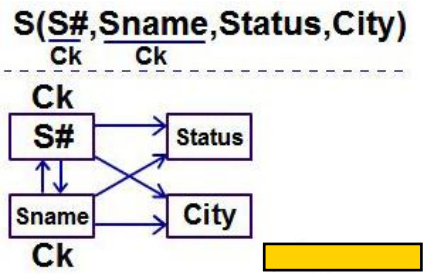
مثال: رابطه های SC و CS و SP

حالت دوم : رابطه بیش از یک CK داشته باشدکه دو حالت دارد:

CK ها صفت مشترک نداشته باشد در این حالت اگر R در 3NF باشد BCNF هم هست.

CK ها صفت مشترک داشته باشد در این حالت اگر R در 3NF باشد BCNF نیست.

مثال:



درکدام فرم است؟

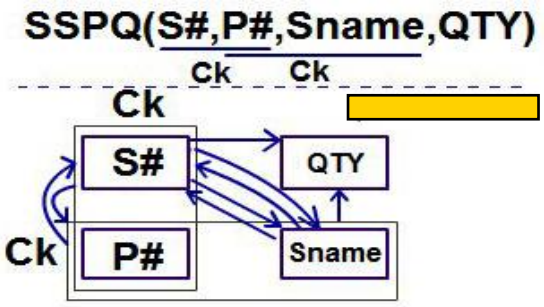
دو دترمینان دارد و هر دو CK پس BCNF است.

دقت کنید : ممکن است شبهه ایجاد شود که TFD دارد و 3NF نیست پس چگونه BCNF است، اما در اینجا TFD نداریم.

مثال ۲:

یک تهیه کننده بیش از یک قطعه تهیه می کند.

فرض می کنیم هیچ دو تهیه کننده نام یکسان ندارند.



این رابطه BCNF نیست .

چون دو دترمینان داریم که CK نیستند:

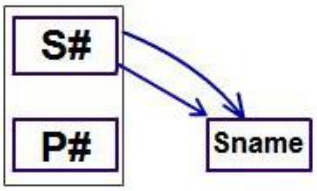
S# → Sname
Sname → S#

کنجکاوی: این رابطه در کدام فرم نرمال است ؟

در 3NF است، (علی رغم این که اختلاط اطلاعات هم دارد)

چرا 3NF است؟ (از تعریف کاد استفاده می کنیم)

- ✓ 1NF است چون صفات تک مقداری است.
- ✓ 2NF است چون در این جا FD ناکامل نداریم، زیرا Sname صفت عمده یا جزء کلید است (نا کلید نیست)



- ✓ 3NF است چون TFD نداریم.

اثبات به روش زانیولو:

در این روش نیاز به بررسی تک تک FD ها نیست بلکه تمام FD هایی که ناشی از CK هستند ، مشکل ندارند و بررسی نمی کنیم بلکه آنهایی که مشکوک هستند را بررسی می کنیم.

دو FD زیر را بررسی می کنیم، کافی است که یکی از آنها شرایط لازم را نداشته باشد:

S# → Sname
Sname → S#

- ✓ S# → Sname ، Trivial نیست.
- ✓ S# سوپر کلید نیست.
- ✓ Sname جزء کلید است.

در نتیجه 3NF است.

کنجکاوی : این رابطه چگونه تجزیه می شود؟ آیا تجزیه خوب دارد؟ بلی دارد

تجزیه ۱:

SP(S#,P#,QTY)
SS(S#,P#,Sname)

یا:

SP(SName,P#,QTY)
SS(S#,P#,Sname)

کنجاوی : فرض می کنیم در این محیط بیش از یک تهیه کننده نام یکسان داشته باشند . چه بیش می آید؟

(P#,Sname) دیگر نمی توانند CK باشند. رابطه حتی 2NF هم نیست .چرا ؟

3:29:00

: 4NF

نکات مهمتر :

تعریف ۱(عملی) : رابطه R در 4NF است هرگاه در BCNF باشد و وابستگی چند مقداری نداشته باشد.

وابستگی چند مقداری: MVD یا Multi Value Dependency

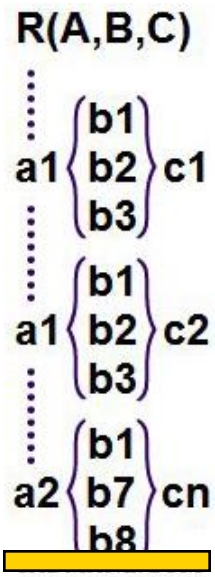
تعریف MVD : در رابطه R(A,B,C) (صفات ساده یا مرکب هستند) صفت B با صفت A ، MVD دارد و چنین نمایش می دهیم:

$A \rightarrow\rightarrow B$

هرگاه مجموعه ای از مقادیر B متناظر با یک مقدار A وجود داشته باشد که با تغییر مقدار A تغییر کند و نه با تغییر مقدار C

مثال:

در این مثال یک جفت a و c در نظر می گیریم و یک مجموعه b مثل مجموعه {b1,b2,b3} که به جفت a1 و c1 اختصاص یافته است. a1 و c1(c هر چیزی می تواند باشد) به همراه این مجموعه سه عضوی سه تاپل به ما می دهد. تا زمانی که a تغییر نکند مجموعه {b1,b2,b3} نمی تواند تغییر کند اما اگر a تغییر کند b می تواند تغییر کند. (صفحه DY-4 ضمیمه)



نکات :

۱. در $A\rightarrow\rightarrow B$ اگر $B\subseteq A$ یا اگر $A\cup B=H_R$ باشد MVD را نامهم گوییم.

۲. MVD تعمیم مفهوم FD است، به بیان دیگر FD حالت خاصی است از MVD که درآن ، آن مجموعه تک عنصری است . (هر FD یک نوع MVD است اما هر MVD ، FD نیست.)

$$\text{If } A\rightarrow B \Rightarrow A\rightarrow\rightarrow B$$

✓ عکس قضیه صادق نیست .

۳. MVD در رابطه های با سه صفت یا سه گروه صفات همیشه جفت است .

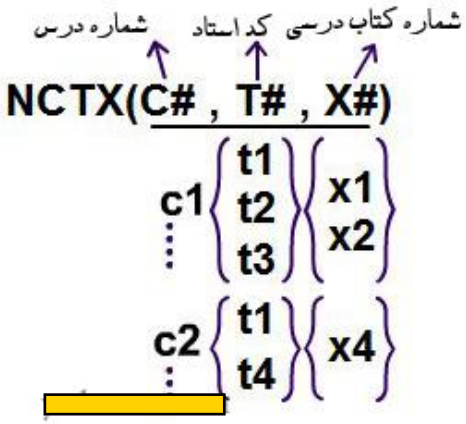
$$\text{If } A\rightarrow\rightarrow B \text{ Then } A\rightarrow\rightarrow C \text{ یا } A\rightarrow\rightarrow H_R-(A\cup B)$$

۴. برای MVD هم قواعد آرمسترانگ داریم (صفحه ی آخر DY-٤)

ایده اصلی در *4NF* :

رابطه BCNF می تواند باز هم آنومالی داشته باشد.

مثال :



درس توسط مدرس از روی کتاب درسی تدریس می شود.

در این محیط محدودیت معنایی خاصی وجود دارد.

یک درس توسط مجموعه ای از مدرسین از روی مجموعه ای از کتاب ها تدریس می شود.

همین محدودیت به ما امکان می دهد تا بسط نرمال این رابطه را چنین تولید کنیم:

C#	T#	X#
C1	T1	X1
C1	T1	X2
C1	T2	X1
C1	T2	X2
C1	T3	X1
C1	T3	X2
…	…	…
C2	T1	X4
C2	T4	X4
…	…	…

CTX تمام کلید است.

نکته : رابطه تمام کلید حداقل BCNF است چون یک دترمینان دارد که همان heading است و آن هم CK است.

- ✓ با این همه باز هم آنومالی دارد.

مثال: درج کن <C1,X5> (C1 از روی کتاب X5 تدریس می شود.)

باید آن محدودیت خاص رارعایت کنیم، در نتیجه باید برای همه مدرسین آن را درج کنیم:

<c1,t1,x5>
<c1,t2,x5>
<c1,t4,x5>

یعنی باید ۳ رکورد درج کنیم. البته اینجا ۳ تاپل است ۳۰۰۰۰۰۰ باشد چه باید بکنیم؟!؟

این آنومالی را شخصی به نام Fagin پیدا کرده است .

- ✓ در نتیجه CTX باید نرمال تر شود.
- ✓ دلیل آنومالی CTX پدیده MVD است.

در CTX داریم:

C# →→ T#
C# →→ X#

بنابراین CTX را به مبدا MVD تجزیه می کنیم . (مبدا در هر دو رابطه باشد)

CX(C#,X#)

C#	X#
C1	x1
C1	x2
C2	x4

CT(C#,T#)

C#	T#
C1	T1
C1	T2
C1	T3
C2	T4

- ✓ CT,CX آنومالی CTX را ندارند.
- ✓ CT,CX از CTX نرمال ترند.
- ✓ هر دو تمام کلیدند. پس حداقل BCNF هستند.
- ✓ MVD ندارند. پس 4NF هستند.

تعریف اول 4NF: رابطه R در 4NF است اگر BCNF باشد و MVD نداشته باشد.

تعریف دوم 4NF: رابطه R با مجموعه FD ها و MVD هایش در 4NF است اگر و فقط اگر بازای هر MVD در F^+ به صورت $X \rightarrow\rightarrow A$ ، یکی از دو شرط زیر قرار باشد:

۱. آن MVD نامهم باشد.(trivial)

۲. X خود سوپر کلید باشد. (یعنی Unique باشد و در صورتی که Unique باشد اجازه نمی دهد MVD بوجود آید چون تکرار نمی شود)

تست: تا کدام فرم نرمال مفهوم FD مستقیما مطرح است ؟

- ✓ تا BCNF

قضیه Fagin : رابطه $R(A,B,C)$ به دو پرتوش $R_1(A,C)$ و $R_2(A,B)$ تجزیه بی کاست می شود اگر و فقط اگر $A \rightarrow\rightarrow B$

- ✓ قضیه FAGIN تعمیم قضیه Heath است .

ابتدای صدای ۲ جلسه نهم

5NF : (PJ/NF) فرم نرمال پرتو پیوندی: نکات مهمتر (بحث اجمالی)

در 5NF مفهومی داریم به نام وابستگی پیوندی (Join Dependency)

تعریف: رابطه R وابستگی پیوندی به n پرتوش $R_1,R_2,\ldots,R_n$ دارد هرگاه R با پیوند n پرتوش به دست آید و نه کمتر.

و به شکل زیر نمایش می دهند:

$R=[JD]*(R_1,R_2,\ldots,R_n)$

گاهی [JD] نوشته نمی شود:

$R=*(R_1,R_2,\ldots,R_n)$

مثال:

CTX=JD*(CT,CX)

مثال دیگر:

FIRST=JD*(SC,CS,SP)

تعریف JD نامهم : وقتی که یکی از R_i ها (پرتوها) خود R باشد.

تعریف 5NF :

رابطه R در 5NF است، هر گاه هر چه JD داشته باشد، ناشی از CK باشد.

به بیان دیگر، در Heading همه پرتوها در همه GD ها CK وجود داشته باشد.

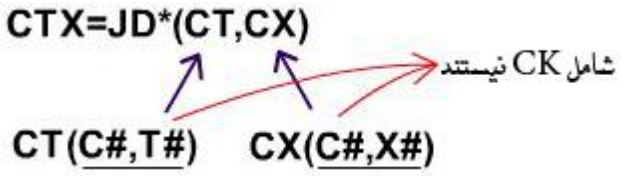
به عبارت دیگر Heading همه پرتوها در تمام GD ها خود سوپر کلید باشد.

مثال از رابطه غیر 5NF:

CTX(C#,T#,X#)

- ✓ در BCNF است.
- ✓ در 4NF نیست. طبعا نباید در 5NF باشد.

دقت کنید در Heading هیچ یک CK حضور ندارد(CK شکسته) پس 5NF نیست:



هنگاهی که JD پیدا کنیم و پرتو هایی از آن بگیریم و JOIN کنیم خودش را بدهد ولی پیوند روی CK نباشد 5NF نیست.

مثال از رابطه ای که 4NF است ولی 5NF نیست:

SPJ(S#,P#,J#)

S#	P#	J#
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

بحث غیر تئوریک(ساده شده)

رابطه SPJ تمام کلید است پس حداقل BCNF است.

MVD ندارد :نمی توان آن مجموعه ها را تشکیل داد(با توجه به دیتا) پس 4NF است . با این همه 5NF نیست.

فرض: بخواهیم SPG را تجزیه کنیم:

SP(S#,P#)

S#	P#
S1	P1
S1	P2
S2	P1

SPJ(P#,J#)

P#	J#
P1	J2
P2	J1
P1	J1

از پیوند این دو رابطه باید SPJ بدست آید اما:

SPJ'(S#,P#,J#)

S#	P#	J#
S1	P1	J2
S1	P1	J1
S1	P2	J1
S2	P1	J2
S2	P1	J1

تاپل اضافی ایجاد می شود(حشو)

$$R \subseteq R1 \bowtie R2 \bowtie \dots \bowtie Rn$$

پرتو دیگری را در نظر می گیریم:

SJ(S#,J#)

S#	J#
S1	J2
S1	J1
S2	J1

حال پیوند 'SPJ و SJ را بدست می آوریم:

SPJ' ⋈ SJ = SPJ"(S#,P#,J#)

S#	P#	J#
S1	P1	J2
S1	P1	J1
S1	P2	J1
S2	P1	J1

تاپل زیادی حذف شد!

*رابطه SPJ تجزیه شدنی به دو رابطه نیست، اگر بخواهیم تجزیه اش کنیم باید تجزیه شود به سه رابطه و نه کمتر.

- ✓ رابطه بالا 5NF نیست، زیرا یک JD دارد((SP,PJ,SJ)*=SPJ) که ناشی از CK نیست. یعنی در Heading آنها CK حضور ندارد.

جمع بندی:

۱- مزایای نرمالتر سازی:

۱- ارائه یک طراحی واضح (Clean Design) یعنی با کمترین اختلاط اطلاعات. در عمل رعایت اصل " One fact – One Table"

۲- کاهش بعضی افزونگی ها : آن افزونگی هایی که با پرتوگیری کاهش پیدا می کنند.

۳- کاهش بعضی آنومالی ها

٤- اعمال بعضی قواعد جامعیت (ناشی از وابستگی بین صفاتی)

٥- تسهیل برنامه سازی در مواردی(SQL)

- ✓ در عمل نباید در استفاده از این تئوری افراط کرد، زیرا معایبی دارد:

۲- معایب

۱- Overhead در بازیابی (بدلیل لزوم انجام join). به همین دلیل گاه لازم می شود در عمل رابطه را Denormalized کنیم. یعنی تبدیل N≥2 رابطه (i+1)NF به یک رابطه iNF

۲- فرض تئوری نرمالترسازی چندان واقع بینانه نیست.

در این تئوری فرض این است که ابتدا یک رابطه همگانی داریم شامل تمام صفات محیط و سپس مرحله به مرحله با سنتز مناسب صفات (جداسازی صفات) نهایتا به تعدادی رابطه در فرم نرمال مطلوب می رسیم.

حال آن که در عمل ابتدا Top-Down و سپس تست نرمالیتی هر یک از رابطه های به دست آمده.

و بهتر است ابتدا تست سریع تشخیص 5NF انجام شود.

تکنیک تشخیص سریع 5NF (Shortcut Date-Fagin)

اگر رابطه ی R در 3NF باشد و تمام CK های آن ساده باشند، در 5NF است.

۳- نرمالتر سازی ایجاد میزانی افزونگی می کند.

اگر بخواهیم پیوند بازسازنده باشد باید یا Ck-Ck باشد یا Ck-Fk و این یعنی میزانی افزونگی (با فرض تناظر ۱-۱یعنی هر رابطه یک فایل)

٤- تعدد تجزیه ها تصمیم گیری را دشوار می کند.

٥- فرآیند نرمال تر سازی زمان گیر است. به ویژه وقتی محیط بزرگ باشد.

٦- در این تئوری از جبر رابطه ای استفاده ی محدود شده است.

یعنی تجزیه با پرتو، بازسازی با پیوند. از عملگر های دیگر استفاده نشده است، حال آنکه در عمل گاه لازم می شود رابطه ها را تجزیه افقی کنیم.

$$\begin{aligned} &SC1=\sum_{City=C1}(S) \\ &SC2=\sum_{City=C2}(S) \\ &\dots \\ &SCn=\sum_{City=Cn}(S) \end{aligned} \quad \Rightarrow \quad S=\cup SC_i$$

نکته : رابطه هایی که مثل بالا با $\sum$ به دست می آیند در RUNF هستند.

Restriction Union Normal Form

- ✓ RUNF در امتداد فرم های نرمال نیست.
- ✓ به موازات فرم های نرمال مطرح می شود

۷- در این تئوری همه وابستگی های بین صفات مطرح نشده اند

وابستگی شمول و ... مطرح نشده است و فقط وابستگی های تابعی مطرح است.

توضیح صفحات ضمیمه:

خواص عملگر ها صفحه A1

مجموعه عملگر های کامل A2

مثال های آموزشی A5

از Q1 تا Q9 همان QUERY هایی که در SQL جواب داده شد در جبر و حساب دوباره نویسی شده است. این مجموعه بی نظیر از date می باشد.