بسمه تعالی

جلسه اول :

سیستم ذخیره و بازیابی اطلاعات در معنای عام:

هر سیستمی که به کاربر برنامه‌ساز یا نابرنامه‌ساز امکان دهد تا اطلاعات خود را ذخیره، بازیابی و پردازش کند

رده‌های تکنولوژیکی سیستم مدیریت پایگاه داده‌ها:

1- **سیستم فایلینگ** : یعنی اطلاعاتی که میخواهیم را در یک فایل انند word , Excel , ... ذخیره کنیم.

2- **سیستم مدیریت داده‌ها** : از حجم اطلاعات کم باشد و نیز داده ها به هم ارتباط نداشته باشند،از آن استفاده میشوند.

3- **سیستم مدیریت پایگاه داده‌ها** : اگر حجم داده ها زیاد و به هم مرتبط باشند استفاده میشوند.

4- سیستم مدیریت پایگاه شناخت

5- سیستم مدیریت پایگاه داده‌های شیئ‌گرا

6- سیستم هوشمند مدیریت پایگاه داده‌ها

7- سیستم معنایی مدیریت پایگاه داده‌ها

8- سیستم مدیریت پایگاه داده‌های زمانبند

9- سیستم مدیریت پایگاه داده‌های نیم‌ساختمند و ناساختمند

10- سیستم مدیریت پایگاه داده‌های بی درنگ

11- سیستم داده‌کاوی و کشف شناخت

12- سیستم مدیریت چند پایگاهی

13- سیستم اطلاعات اجرائی

14- سیستم فعال مدیریت پایگاه داده‌ها

15- سیستم مدیریت پایگاه داده‌های شیئ-رابطه‌ای

مفاهیم اولیه .....

بانک اطلاعاتی (پایگاه داده data base) :

فایل های اطلاعاتی که به نوعی به هم مرتبط هستند ، تشکیل یک بانک اطلاعاتی را می دهند.(که این فایل شامل مجموعه ای از فیلد های به هم مرتبط می باشد، است)

مثلا در بانک اطلاعاتی دانشگاه ، یه سری دانشجو و اساتید داریم که در ان یه فایل برای مشخصات دانشجویان(فیلد های مربوط به هر دانشجو مانند: نام ، نام خانوادگی، معدل و ...) ، یک فایل برای مشخصات اساتید ، یک فایل برای اطلاعات دروس ، یک فایل برای دانشکده ها و یک فایل برای گروه آموزشی و.... است .

داده:

مجموعه ای از بوده ها و مفاهیمی که توسط انسان یا هر موجود خودکار(ماشین) دیگر برای برقراری ارتباط و یا پردازش مورد بررسی قرار می گیرد.

اطلاع:

داده ای است که روی آن پردازش انجام شده است و یا به عبارت دیگر داده سازمان یافته ای که شناختی را منتقل می کند.

- داده همان مقدار واقعا ذخیره شده و اطلاع معنای داده است.

تعریف دقیق پایگاه داده (DB) :

مجموعه‌ای است از داده‌های ذخیره شده و پایا **(یعنی داده هایی که از بین نمیروند چه با خاموش کردن سیستم و چه با restart کردن آن)** ، به صورت یکپارچه **(نه لزوما فیزیکی، بلکه حداقل به طور منطقی)**، بهم مرتبط **(چون ارتباط بین داده ها هم در پایگاه داده ذخیره می شود.)** ، با کمترین افزونگی **(با کمترین تکرار داده ها)**، تحت مدیریت یک سیستم کنترل متمرکز (توسط **"Data Base Management System" - >DBMS)** ، مورد استفاده یک یا چند کاربر از یک یا بیش از یک "سیستم کاربردی"، به طور همزمان و اشتراکی**(پایگاه داده باید این قابلیت را داشته باشد که چندین کاربر به صورت همزمان قابلیت استفاده ار آن را داشته باشند)**

برای ایجاد یک سیستم کاربردی دو رهیافت وجود دارد:

1- رهیافت سنتی یا مشی فایلینگ

مراحل اولیه طراحی و تولید هر قسمت به طور کلاسیک انجام شده و بعد از طراحی، مشخصات هر سیستم همراه با وظایف آنها مشخص می شود. برای ایجاد محیط ذخیره سازی اطلاعات از یک سیستم فایل و برای برنامه سازی از یک زبان سطح بالا استفاده می شود و در نهایت برای هر قسمت، یک سیستم کاربردی ایجاد می شود.

یا به عبارت کلی تر اطلاعات در فایل ها و در سیستم های جداگانه ذخیره می شود و هر سیستم متناسب با اطلاعاتی که می خواهد دارای برنامه های منحصر به خود است .

مراحل کلی کار:

- تحلیل و بررسی نیازهای اطلاعاتی و پردازشی هر قسمت به طور جداگانه
- اجرای مراحل کلاسیک اولیه لازم برای طراحی و تولید یک سیستم کاربردی
- تعیین مشخصات هر سیستم و وظایف آن
- طراحی تعدادی فایل
- نوشتن مجموعه‌ای از برنامه‌های ایجاد، کنترل و پردازش فایل
- استفاده از یک پیکربندی سخت‌افزاری و نرم‌افزاری مشخص
- انجام تستهای لازم و تنظیم سیستم کاربردی
- ایجاد یک سیستم کاربردی برای هر قسمت و برپایی محیط فیزیکی ذخیره و بازیابی اطلاعات و سیستم بهره‌برداری از آن خاص همان قسمت.

معایب این روش:

- سخت افزار زیادی احتیاج دارد
- برنامه نویسی زیادی میخواهد
- افزونگی زیادی دارد
- ناسازگاری دارد**(یعنی تغییر یک بخش از اطلاعات در یک سیستم باعث تغییر در سیستم دیگر نمی شود)**
- یک سیستم یکپارچه ندارد **(یعن نمی تواند کنترل متمرکزی روی داده ها داشته باشد)**
- عدم وجود ضوابط ایمنی
- عدم امکان اشتراکی شدن داده ها
- وابستگی برنامه های کاربردی به محیط ذخیره سازی داده ها **(یعنی اگر یک فایل از یک درایو به جای دیگری منتقل شود دیگر کار نمی کند)**

عکس مربوط به نمایش مشی فایلینگ

2- رهیافت (مشی) پایگاهی

در این روش نیاز های اطلاعاتی تمامی قسمتها مورد مطالعه قرار می گیرد تا بتوان یک سیستم یکپارچه طراحی کرد .
تمامی اطلاعات به صورت منطقی در یک جا ذخیره می شود که مدیریت آن به عهده ی DBMS (**مجموعه نرم افزار هایی که در سیستم عامل نصب می شوند و وظیفه آن ذخیره و بازیابی اطلاعات و مدیریت پایگاه داده است**) است و کاربران بر اساس نیاز خود، پایگاه خود را تعریف کرده و هر کاربر تصور می کند که پایگاه خود را دارد.

در این نوع نمایش باید تمامی اطلاعات به صورت جامع و کامل ذخیره شود و همه این داده ها در اختیار همه قرار نمی گیرد و به هر قسمت تنها داده های مربوط به خودش داده می شود.

مراحل کلی کار:

- بررسی و تحلیل نیازهای پردازشی و اطلاعاتی همه قسمت ها توسط یک گروه
- مدلسازی معنایی داده‌ها
- تعیین مشخصات جامع (یکپارچه) کاربردی و وظایف آن
- انتخاب یک یا چند پیکربندی سخت‌افزاری-نرم‌افزاری
- استفاده از یک یا چند DBMS
- طراحی پایگاه داده‌ها در سطوح لازم
- تولید مجموعه‌ای از برنامه‌های ایجاد و کنترل پایگاه داده
- ایجاد محیط واحد و مجتمع ذخیره‌سازی و مشترک بین کاربران
- طراحی و تولید واسطهای کاربرپسند مورد نیاز
- تعریف پایگاه داده هر قسمت توسط کاربر مربوطه
- طراحی برنامه‌های عملیات در پایگاه داده
- بهره‌برداری واقعی از سیستم پس از تستهای لازم

قسمت های مختلف یک سیستم پایگاهی:

1- نرم افزار (قسمتی که مدیریت داده ها را بر عهده دارد)

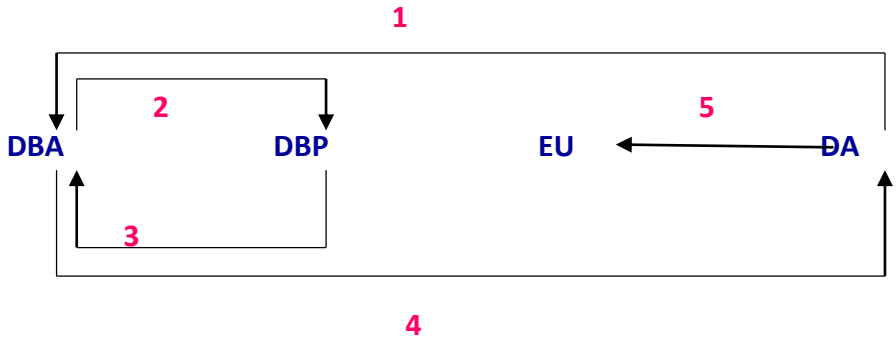
- سیستم مدیریت پایگاه داده‌ها (DBMS)
- نرم‌افزارهای کاربردی
- نرم افزار شبکه
- برنامه‌های کاربردی قابل اجرا در محیط DBMS
- رویه های ذخیره شده

2- سخت افزار (قسمتی که داده ها در آن ذخیره می شود)

- سخت افزار ذخیره ساز
- سخت افزار پردازشگر - > به پردازش داه ها می پردازد
- سخت افزار همرسانش(ارتباط) - > شامل تمامی گذرگاه ها مانند گذرگاه بین مادر بورد و .....

3- کاربر (کاربری که از داده ها استفاده می کند)

- **DBA (Data Base Admin)** – > وظیفه آن طراحی پایگاه داده و ایجاد ارتباط بین DBP ها است.
- **DBP (Data Base Programmer)** – > وظایف را از Admin و اقدام به نوشتن برنامه اون قسمت می کنند
- **EU (End User)** – > کاربر نهایی است
- **DA (Data Admin)** – > دستورات را به DBA داده و آن اطلاعات را به DBP می دهد و سرانجام از آن تحویل گرفته و به DA میدهد و در نهایت در اختیار کاربر یعنی EU قرار می گیرد.



4- داده

جلسه دوم :

معماری پایگاه داده ها :

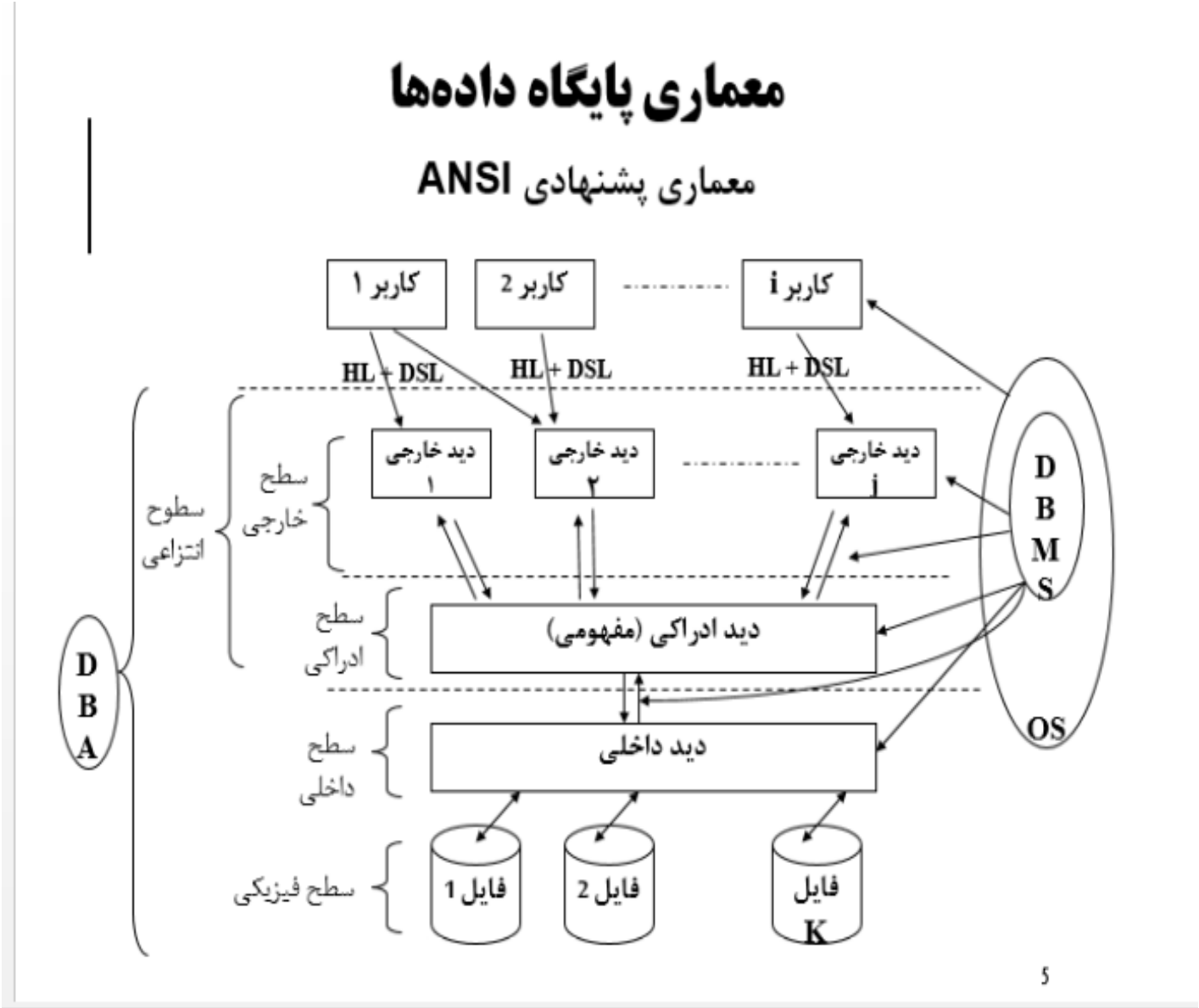
معماری استاندارد پایگاه داده ها که توسط ANSI پیشنهاد شد ، یک معماری سه سطحی است:

سطح انتزاعی یا غیر واقعی (فقط برای درک مفهوم و کارایی طراحی پایگاه داده است.)

1- سطح خارجی (External Level) - > شامل دید های خارجی DBP ها است.

2- سطح ادراکی (Conceptual Level)

3- سطح داخلی (Internal Level) - > در این لایه داده ها در پایگاه داده ها در قالب فایل ذخیره می شود که این سطح یک سطح واقعی است (یعنی حتما وجود دارد و حجمی از فضا را اشغال می کند)



دید : پنجره ای است که از آن کاربر می تواند محدوده پایگاه خود را ببیند و خارج از این محدوده، چیزی نمی بیند .

دید خارجی

1- دید کاربر خاص نسبت به داده‌های ذخیره‌شده در پایگاه داده است.
2- اطلاعات جزئی است.
3- در سطح انتزاعی مطرح است.
4- روی دید ادراکی طراحی و تعریف می‌شود.
5- شمای خارجی نوعی برنامه است حاوی دستورات تعریف و کنترل داده‌ها در سطح خارجی که توسط کاربر این سطح نوشته می‌شود.
6- به تعریف مجموعه دیدهای خارجی کاربر، سطح خارجی گفته می‌شود.
7- دید خارجی منحصر به یک کاربر نیست پس چند کاربر می توانند از یک دید خارجی استفاده کنند .
8- چند کاربر می‌توانند در یک دید مشترک باشند.

- هر دید خارجی حاوی یک بخش از داده است . گفتیم که در سیستم پایگاهی تمامی داده ها جامع و کامل ذخیره می شوند ولی همه اطلاعات در اختیار کاربر قرار نمی گیرد پس هر دید خارجی حاوی بخشی از این اطلاعات است.
- برای هر دید خارجی از 1 تا n کاربر وجود دارد و حداقل باید هر دید یک کاربر داشته باشد .

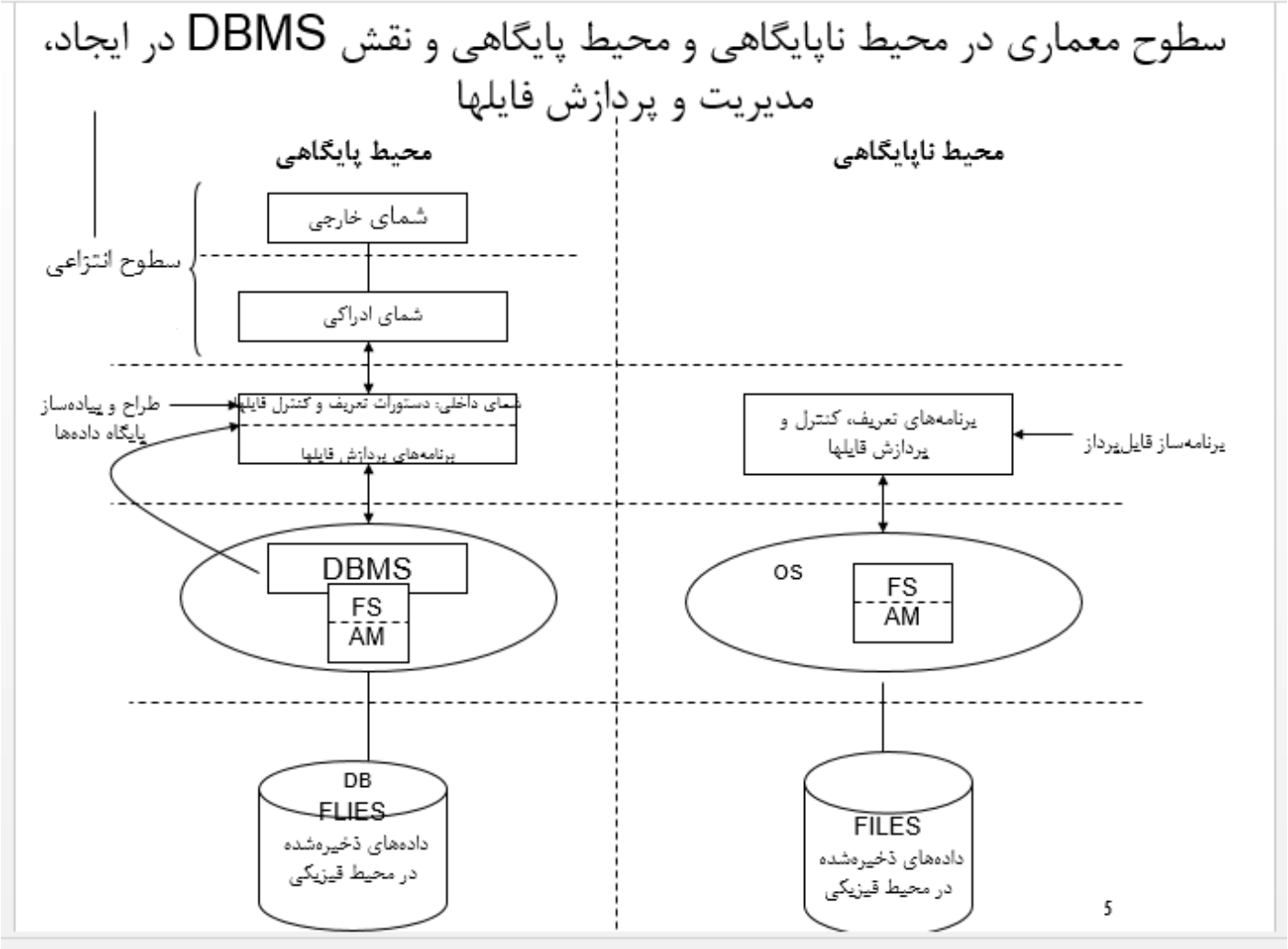
دید ادراکی (مفهومی)

1- دید طراح پایگاه داده‌ها (DBA) نسبت به داده‌های ذخیره‌شده است.
2- جامعو کامل و منحصر به فرد است. (چون در طراحی پایگاه داده هم داده ها و هم ارتباط بین آنها ذخیره می شود پس Admin یاسد یک دید جامع در این زمینه داشته باشد .)
3- در یک محیط انتزاعی مطرح است.
4- با عناصر ساختاری اساسی همان ساختار داده‌ای طراحی می‌شود.
5- شمای ادراکی نوعی برنامه است حاوی دستورات تعریف و کنترل داده‌ها. سطح ادراکی در واقع همان شمای ادراکی است.
9- شمای ادراکی به سیستم داده می‌شود و اطلاعات مربوط به همه دید ها مانند اطلاعات سخت افزار در کاتالوگ سیستم نگهداری می‌شود.
10- ممکن است حاوی اطلاعاتی باشد که در هیچ دید خارجی وجود نداشته باشد.

- در این لایه DBA ها و DBP ها ممکن است دیدگاه های خارجی متفاوتی برای در اختیار گذاشتن داده ها به کاربر نهایی (EU) داشته باشند

دید داخلی

1- دید جامع و کامل و منحصر به فرد DBMS و طراح پایگاه داده‌ها(DBA) است و در سطحی پایین‌تر از سطح ادراکی، نسبت به کل داده‌های ذخیره‌شده است.
2- در سطح فایلینگ منطقی مطرح است.
3- مبتنی بر یک ساختار فایل است که با نظر طراح پایگاه طراحی می‌شود و به طراحی فیزیکی موسوم است.
4- در سطح داخلی پایگاه داده‌ها، فایلینگ منطقی تعریف می‌شود.
5- شمای داخلی نوعی برنامه است که توسط خود DBMS تولید می‌شود و شرح فایلینگ منطق پایگاه است.



مدیر پایگاه داده :

کسی است که اطلاعات زیادی در اختیار دارد و به دید های خارجی دسترسی دارد و به کمک آن سیستم را مدیریت می کند و در واقع باید متخصص پایگاه داده باشد

UIF :

واسط کاربر و دید خارجی است که اطلاعات را از دید خارجی در اختیار کاربر و برعکس قرار می دهد که به دو زبان زیر نوشته می شود.

1- زبان میزبان HL (Host Language)

در واقع برنامه هایی هستند که نیاز به پایگاه داده ندارند و داده ها را در فایل ذخیره کرده یا از پایگاه داده خارج می کنند.
یکی از زبانهای برنامه‌سازی متعارف مانند کوبول، PL1، فرترن، پاسکال، C و زبانهایی مثل ADA، LISP، JAVA و نیز زبان اسمبلی است.

2- زبان داده ای DSL (Data sub Language)

مانند SQL

- از زبان میزبان برای طراحی فرم ها و از DSL برای ذخیره و بازیابی اطلاعات استفاده می شود .
- UIF ها را DBP ها طراحی می کنند .

زبان داده ای شامل سه دستور زیر است :

1- دستورات تعریف داده‌ها **Data Definition Language (DDL)**

دستوراتی که برای تعیریف هر نوع شی در پایگاه داده استفاده می شود .

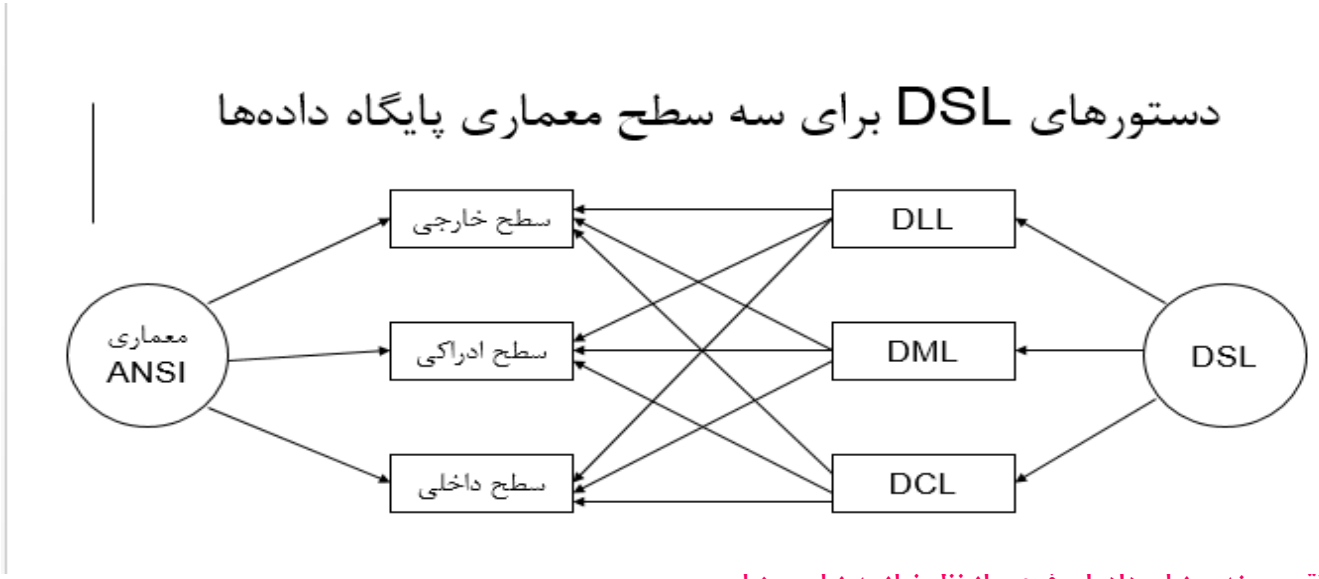
2- دستورات عملیات روی داده‌ها **Data Manipulation Language (DML)**

دستوراتی که داده را پردازش و به اطلاعات تبدیل می کند . در واقع مجموعه دستوراتی که روی داده ها پردازش انجام می دهد و آنها را در پایگاه داده ذخیره می کند.

3- دستورات کنترل داده‌ها **Data Control Language (DCL)**

دستوراتی که به کمک آن می توان داده ها را کنترل کرد.

- برای طراحی هر یک از لایه های خارجی ، ادراکی ، داخلی از DDL ، DML ، DCL استفاده می شود .



تقسیم‌بندی زبان داده‌ای فرعی از نظر نیاز به زبان میزبان

- **مستقل (I.DSL)**: به زبان میزبان نیاز ندارد و به صورت تعاملی استفاده می‌شود. در واقع یک زبان پرس وجو است.
- **ادغام‌شدنی (E.DSL)**: دستورهایش در متن برنامه‌ای به زبان میزبان به کار می‌رود و مستقلا قابل استفاده نیست.

یعنی از دستورات DSL در همان محیطی که زبان میزبان نوشته می شود ، استفاده می شود و هنگام کامپایل دستورات DSL و HL به صورت جداگانه اجرا می شود پس به دو کامپایلر احتیاج است .

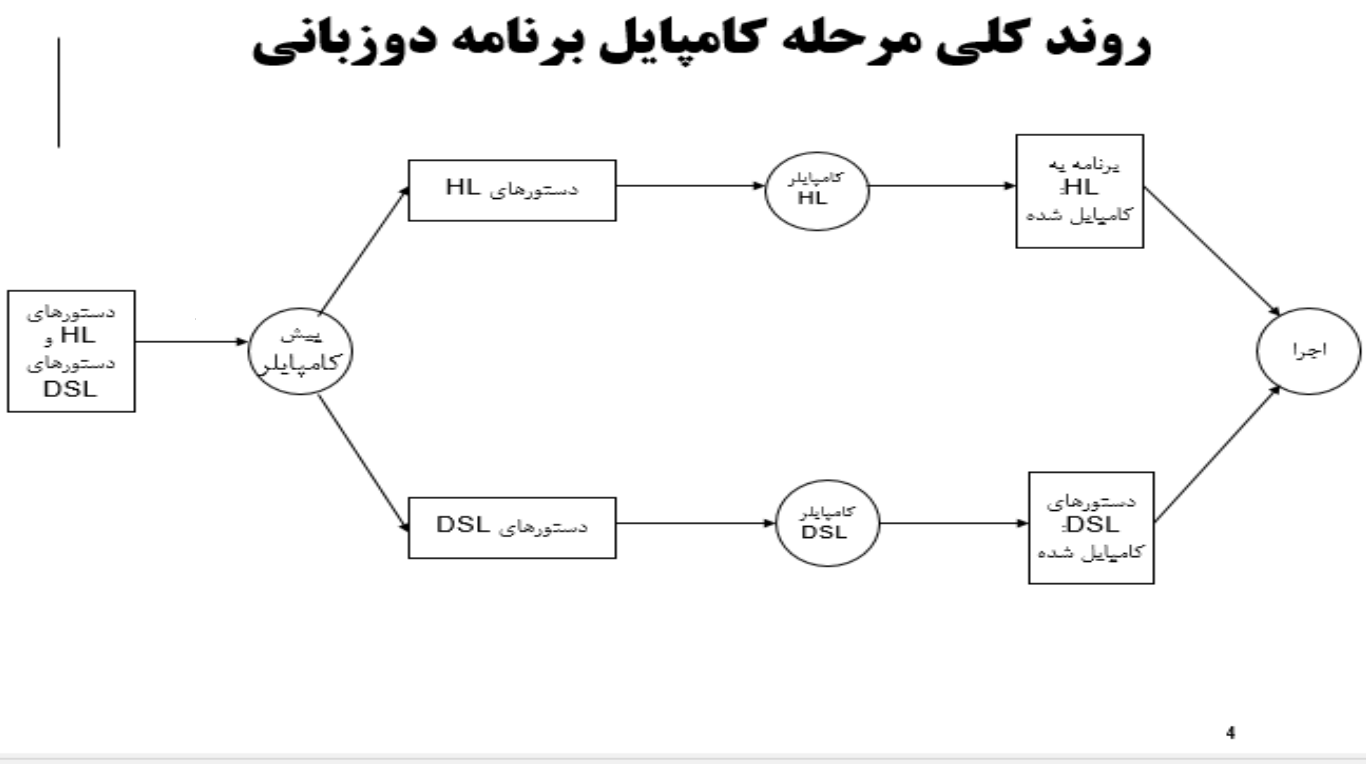
ادغام می تواند صریح یا ضمنی باشد . در حالت ادغام صریح ، عین دستورات DSL در برنامه میزبان نوشته می شود و در حالت ادغام ضمنی، دستورات DSL به صورت توابع فراخوانده می شوند .

در این روش مامعماری متفاوتی داریم :

معماری سه لایه معماری پنج لایه

در معماری سه لایه ، DAL پایین ترین لایه و سطحی است که شامل دستورایت برای ذخیره و بازیابی اطلاعات است. بعد از آن سطح UIF است که در اینجا فرم ها و کلاس ها طراحی و مدیریت می شوند و سطح دیگر Busines شامل مجموعه کلاسهایی است که اطلاعات را از UIF به DAL منتقل می کند.

- **هم مستقل و هم ادغام‌شدنی (I/E.DSL)**



ویژگیهای زبان داده‌ای فرعی:

1- تعداد دستورهایش باید کم باشد.

2- دستورهایش باید شبه زبان طبیعی باشد.

3- یادگیری و استفاده آن باید ساده باشد.

4- در طراحی آن باید اصل وحدت دستور رعایت شود.

5- دستورهایش باید مبتنی بر عناصر ساختاری اساسی ساختار داده‌ای طراحی شوند.

6- بهتر است نارویه‌ای (ناروشمند) باشند.

7- بهتر است کامپایلری باشد و نه مفسری.

8- بهتر است از نظر ساختاری کامل باشد.

9- بهتر است از نظر برنامه‌سازی و محاسباتی کامل باشد.

10- زبان باید از نظر تعداد دستورهای کنترل داده‌ها و عملکرد هردستور، غنی و قوی باشد.

11- باید از نظر انواع داده‌ای و به ویژه انواع داده‌ای انتزاعی و انواع داده‌ای پیچیده غنی باشد.

جلسه سوم :

مدل سازی معنایی داده ها

برای اینکه بخواهیم کاری را انجام دهیم نخست باید مدل سازی کرده و مطابق آن پیاده سازی کنیم.

برای پایگاه داده ها داده‌های ذخیره‌شدنی در پایگاه داده‌ها ابتدا باید در بالاترین سطح انتزاع مدلسازی معنایی شوند که برای این منظور ابتدا DBP مدل سازی می کند و DBA آن پایگاه داده را پیاده سازی می کند .

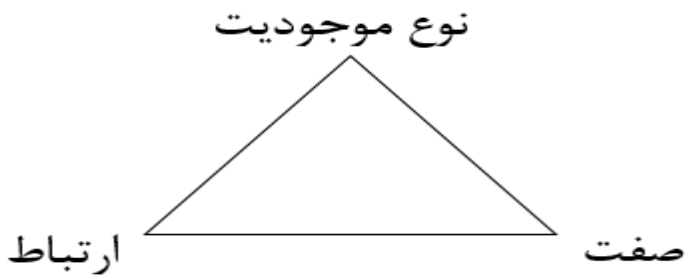
انواع روشهای مدلسازی معنایی داده‌ها

- **روش موجودیت- ارتباط (ER)**
- **روش زبان عمومی مدلسازی (UML)**
- **روش تکنیک مدلسازی شیئی (OMT)**

مدل سازی معنایی :

روی لایه ادراکی عمل می کند و برای ذخیره سازی داده ها در این لایه کاربرد دارد.

سه مفهوم معنایی موجود در روش ER



در مدل ER سه مفهوم اساسی وجود دارد :

1- نوع موجودیت

مفهوم کلی شیئ، چیز، پدیده و به طور کلی هر آنچه که می‌خواهیم در موردش اطلاع داشته باشیم و شناخت خود را در موردش افزایش دهیم.(در واقع هر چیزی که بخواهیم در مورد آن اطلاعات داشته باشیم موجودیت است)

- تعریف موجودیت همان تعریف داده است.
- موجودیت پایگاه داده ها متمایز از هم است .

2- صفت

خصیصه یا ویژگی یک نوع موجودیت است و هر نوع موجودیت مجموعه‌ای از صفات را دارد. هر صفت یک نام، یک نوع و یک معنای مشخص (یعنی هر صفت چه کاری انجام می دهد) دارد. (در واقع صفت ویژگی های مربوط به هر موجودیت است)

- در مدل سازی معنایی همه ی موجودیت ها ، صفات موجودیت ها و رابطه بین موجودیت ها را مدل سازی می کنیم .
- هر کدام از موجودیت ها صفات خود را دارند.(مثلا دانشجو: شماره دانشجویی ، آدرس ،.....)

3- نوع ارتباط

اندرکنش (تعامل) بین دو یا بیش از دو نوع موجودیت است و ماهیتا نوعی بستگی بین انواع موجودیتهاست.

مثلادر سیستم دانشگاهی بین دانشجو و درس ، دانشجو و استاد ، استاد و درس رابطه وجود دارد

نکته :

- موجودیت ها باید دارای چندین نمونه باشندو نمونه ها متمایز باشند .
- موجودیت و نمونه متفاوت اند (نمونه زیر مجموعه ای از موجودیت است)

مثلا در سیستم دانشگاه ، رئیس دانشکده موجودیت نیست بلکه نمونه ای از کارمند است و کارمند موجودیت این سیستم است.

سه ضابطه در رابطه با تشخیص یک نوع موجودیت

1- معمولا نمونه‌هایی متمایز از یکدیگر دارند.

2- معمولا بیش از یک صفت دارد و کاربر به مجموعه‌ای از اطلاعات در مورد آن نیاز دارد.

3- معمولا حالت کنشگری (فاعلیت) یا حالت کنشپذیری (مفعولیت) دارد.

انواع موجودیت :

1- موجودیت مستقل(قوی)

موجودیتی است که مستقل از هر موجودیت دیگر و به خودی خود، در یک محیط مشخص مطرح باشد.

به عبارت خلاصه موجودیتی که وجود آن به خودی خود ارزش دارد.

نکته – > موجودیت های مستقل می توانند از هم ارث بری (لازمه این ارث بری وجود صفات مشترک است) داشته باشند.

2- موجودیت وابسته(ضعیف)

موجودیتی است که وجودش وابسته به یک نوع موجودیت دیگر است.

به عبارت دیگر موجودیتی است که وجود نمونه های این نوع موجودیت وابسته به وجود نمونه های دیگر است و اگر آن نمونه حذف شود ، نمونه های زیر دست آن نیز حذف می شود.

مثلا در سیستم دانشگاه ، کارمند یک موجودیت مستقل است و افراد تحت تکفل کارمند موجودیت وابسته اند و وجود آنها وابسته به وجود کارمند است .

در این سیستم اول اطلاعات کارمند را ذخیره می کنیم و سپس اطلاعات افراد تحت تکفل آن را و اگر یک کارمند اخراج شود ، اطلاعات مربوط به افراد تحت تکفل آن هم حذف می شوند.

انواع صفات :

- **صفت ساده**

مقدار صفت ساده از لحاظ معنایی تجزیه‌نشدنی یا اتومیک است. مانند نام ، روز و ....

- **صفت مرکب**

صفت مرکب از چند صفت ساده تشکیل شده است. مانند آدرس ، تاریخ تولد و .....

- **صفت تک مقداری**

صفتی است که برای یک نمونه از یک نوع موجودیت حداکثر یک مقدار از دامنه مقادیر را می‌گیرد. یعنی در یک زمان به این نمونه یک مقدار به صفت آن اختصاص می دهیم . مثلا شماره دانشجویی هر نمونه از دانشجو (هر دانشجو فقط یک شماره دانشجویی منحر به خود را دارد)

- **صفت چند مقداری**

صفت چندمقداری بیش از یک مقدار از دامنه مقادیر می‌گیرد.

مثلا آدرس (هر نفر ممکن است چند آدرس داشته باشد) یا مدرک تحصیلی (هر نفر میتواند چندین مدرک داشته باشد)

- **صفت شناسه (کلید)**

بین تمامی نمونه ها منحصر به فرد است . مثلا شماره شماره تلفن همراه

دارای دو ویژگی زیر است :

1- یکتایی مقدار دارد.

2- حتی الامکان طول مقادیرش کوتاه است.

- **صفت هیچ مقدار پذیر**

اگر مقدار یک صفت در یک یا بیش از یک نمونه از یک نوع موجودیت، برابر با هیچ‌مقادر (هیچ مقدار یعنی مقدار ناشناخته، مقدار غیرقابل اعمال، مقدار تعریف‌نشده)باشد، آن صفت هیچ‌مقدارپذیر است.

به عبارت خلاصه تر صفتی است که ممکن است برای بعضی از نمونه ها مقدار نداشته باشد. مثل معدل داشجوی ترم اول

- **صفت هیچ مقدار ناپذیر**

مثل اسم هر فرد و شماره دانشجویی هر دانشجو

- **صفت ذخیره شده**

صفتی است که مقادیرش در پایگاه داده‌ها ذخیره شده باشد.

- **صفت مشتق شده**

صفتی است که مقادیرش در پایگاه داده‌ها ذخیره نشده باشد، بلکه حاصل یک پردازش روی فقره‌هایی از داده‌های ذخیره شده باشد.

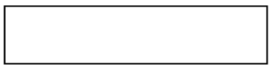
(نمره دانشجو صفت ذخیره شده و معدل دانشجو صفت مشتق شده است چون نیاز به ذخیره آن نیست و از روی نمرات ذخیره شده و با محاسبه بدست می آید.)

نکته :

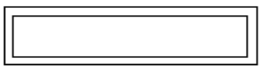
در سیستم خرید ماشین ، ماشین یک موجودیت و رنگ یک صفت به حساب می آید اما در سیستم کارخانه رنگ سازی ، رنگ (چون محصول است) موجودیت و مثلا میزان رنگ صفت است.

نمادهای رسم نمودار ER

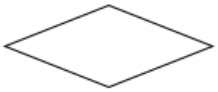
نوع موجودیت



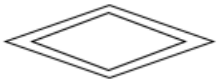
نوع موجودیت ضعیف



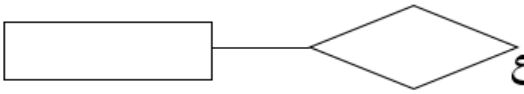
نوع ارتباط



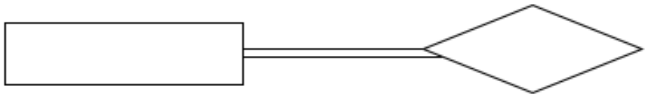
نوع ارتباط با موجودیت ضعیف



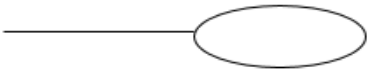
مشارکت نوع موجودیت در نوع ارتباط



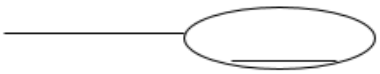
مشارکت الزامی



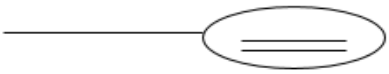
صفت



صفت شناسه اول

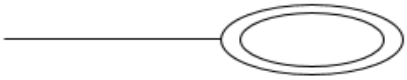


صفت شناسه دوم



صفت شناسه مرکب

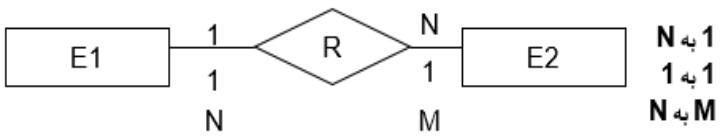
صفت چندمقداری



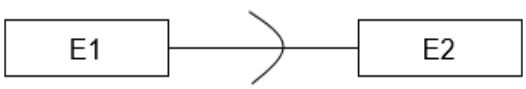
صفت مرکب

صفت مشتق

چندی ارتباط



ارتباط "گونه‌ای است از"



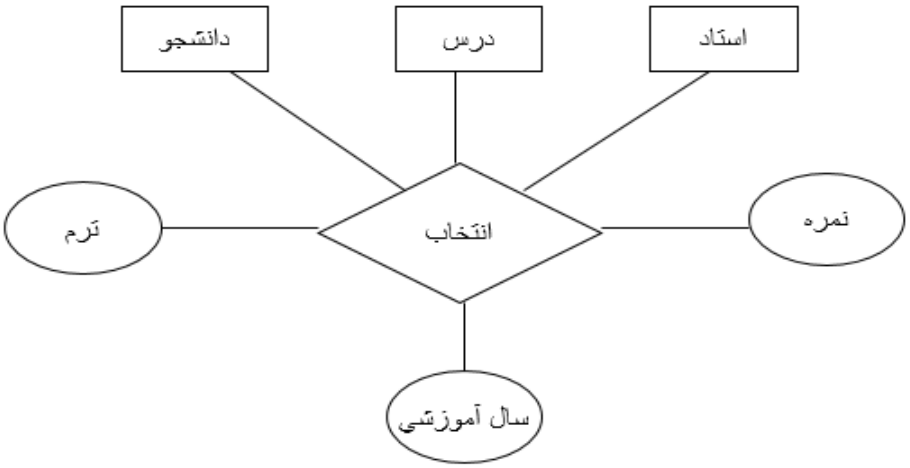
وضع مشارکت در ارتباط

مشارکت یک نوع موجودیت در یک نوع ارتباط را الزامی گویند، اگر تمام نمونه‌های آن نوع موجودیت در آن نوع ارتباط شرکت کنند. در غیر این صورت مشارکت غیرالزامی است. (DA مشخص می کند که مشارکت الزامی است یا نه)

نمایش مشارکت الزامی

درجه ارتباط:

تعداد شرکت‌کنندگان (موجودیت ها) در یک ارتباط را درجه آن ارتباط می‌گویند.



ارتباط بین سه موجودیت

درجه ارتباط = 3

- چندی یا ماهیت نوع ارتباط عبارتست از چگونگی تناظر بین دو مجموعه نمونه‌های آن دو نوع موجودیت.

انواع چندی ارتباط:

1- یک به یک 1:1

اگر یک نمونه از موجودیت اول با یک نمونه از موجودیت دوم در ارتباط باشد .

2- یک به چند 1:N

اگر یک نمونه از موجودیت اول با چند نمونه از موجودیت دوم ارتباط داشته باشد

3- چند به چند N:M

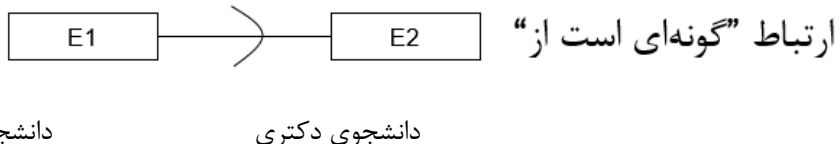
اگر چند نمونه از موجودیت اول با چند نمونه از موجودیت دوم ارتباط داشته باشد

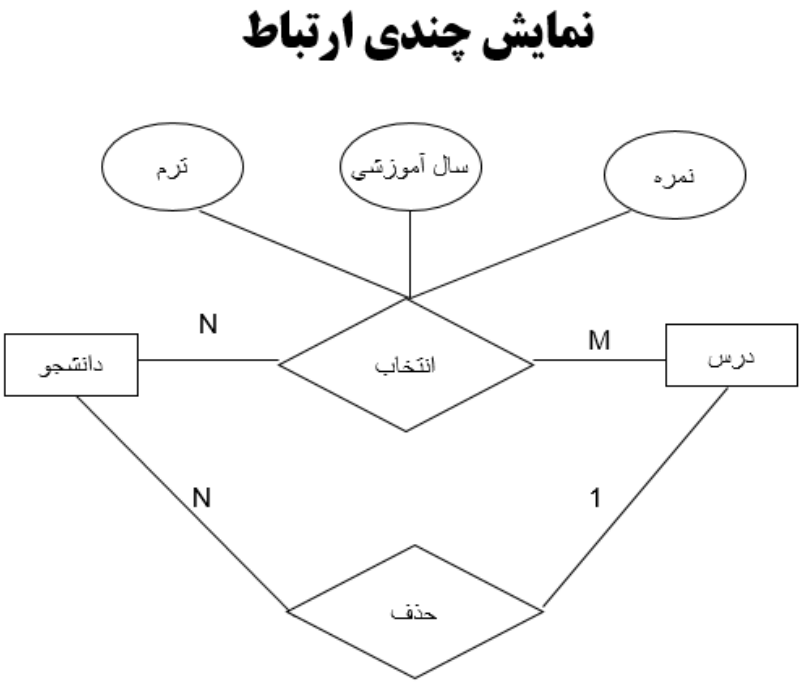
چند نمونه مثال از ارتباطات

- **ارتباط بین استاد راهنما و دانشجو** – > ارتباط **یک به چند** است چون هر دانشجو یک استاد راهنما و هر استاد راهنما چند دانشجو دارد
- **ارتباط بین فرد و عکس موجود در ثبت احوال** – > ارتباط **یک به یک** است چون هر فرد یک عکس و هر عکس موجود منحصر به فرد خاصی است.
- **ارتباط بین درس و دانشجو** – > ارتباط **چند به چند** است چون هر دانشجو چندین درس و هر درس در دانشگاه میتواند چندین دانشجو داشته باشد .

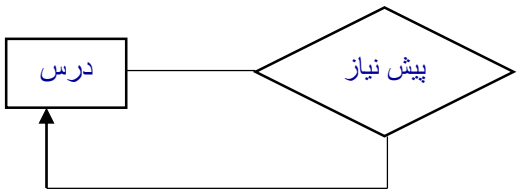
ارتباط گونه ای :

ارتباطی که در آن موجودیت دوم گونه ای از موجودیت اول است و E2 از تمامی صفات E1 ارث می برد .





- **بین دو موجودیت میتواند چندین رابطه وجود داشته باشد**
- **رابطه ممی تواند صفت داشته باشد**
- **اصولا کم پیش می آید که درجه یک رابطه از 4 بیشتر شود و عموما کمتر از آن است .**
- **رابطه درجه یک هم داریم مانند رابطه یک درس با پیش نیازش**



جلسه چهارم :

انواع دیگری از روش های مدل سازی :

OMT -- > مدل سازی شی گرا

UML -- > مدل سازی مخصوص رشته صنایع

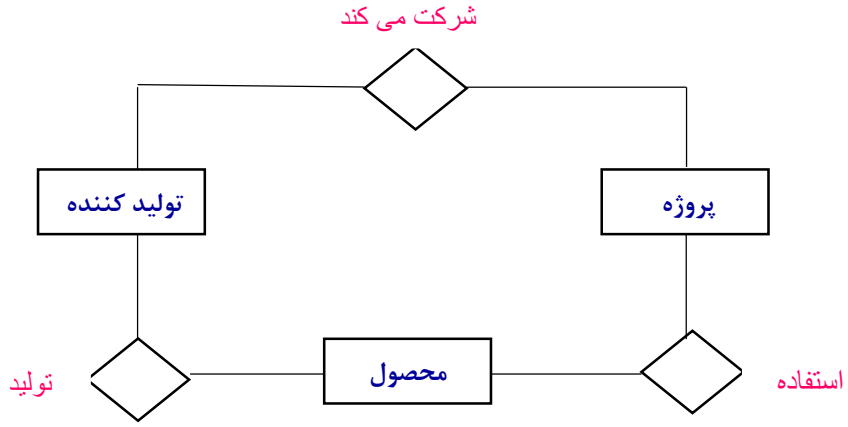
EER -- > روش ER توسعه یافته که مشکلات روش ER را برطرف کرده است.

مشکلات روش ER :

1- دام حلقه

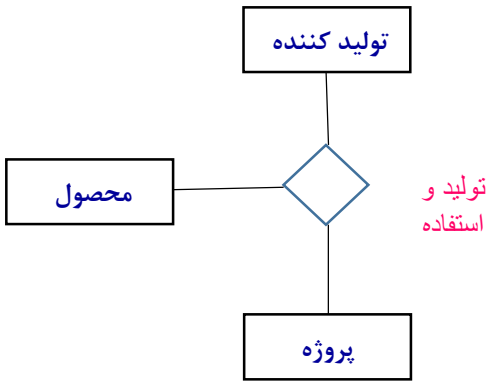
اگر سه موجودیت داشته باشیم که دو به دوی آنها با هم ارتباط داشته باشند پیش می آید .

در این حالت ممکن است یک ارتباط از درجه 2 به اشتباه ارتباط از درجه 3 برداشت شود .



برداشت درست :

درجه ارتباط = 2



برداشت غلط :

درجه ارتباط = 3

اگر تولید کننده را شرکت تورین تن ، محصول را پیچ و پروژه را سد کرج در نظر بگیریم :

مفهوم شکل 1 : شرکت تورین تن پیچ را منحصرا برای استفاده سد کرج تولید می کند.

مفهوم شکل 2 : شرکت تورین تن پیچ را تولید می کند اما برایش مهم نیست که در سد کرج استفاده شود یا نه

** تمرین : دام شکاف و دام چتری (چند شاخه ای) چیست ؟؟؟

2- دام شکاف

3- دام چتری(چند شاخه ای)

محدودیتهای روش ER که در روش EER برطرف گردیدند:

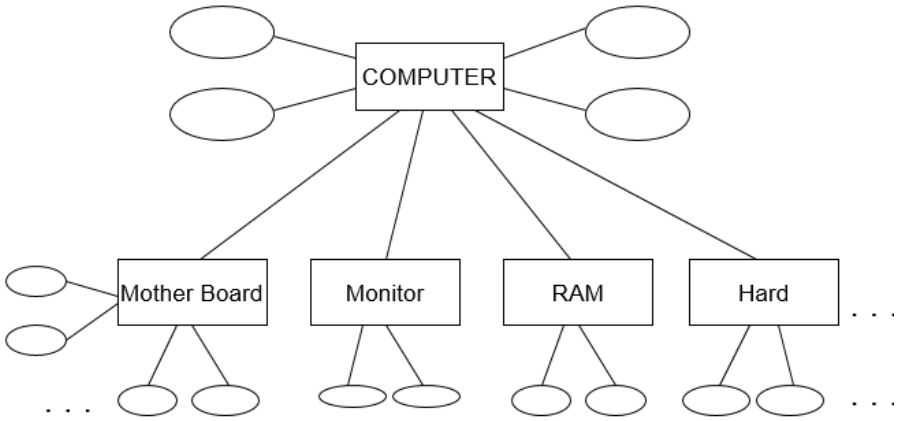
1- تجزیه

یعنی یک شیئ کل را به اجزاء تشکیل‌دهنده آن تقسیم کنیم. شیئ کل صفات، ساختار و رفتار خود را دارد و هریک از اجزاء نیز صفات، ساختار و رفتار خاص خود را دارند. شیئ کل شامل اجزاء خود است و بین شیئ کل و اجزایش، ارتباط شمول وجود دارد. به این نوع ارتباط در EER، ارتباط "جزئی است از ..." گفته می‌شود.(موجودیت های بزرگ به موجودیت های کوچک تقسیم می شود)

2- ترکیب

ترکیب، عکس عمل تجزیه است و در این عمل، با داشتن Ei(i=1 , 2 , ...) یک نوع موجودیت E را بازشناسی می‌کنیم به نحوی که Eiها اجزاء تشکیل‌دهنده آن باشند (موجودیت های کوچک کنار هم قرار می گیرند و موجودیت بزرگتر را به وجود می آورند.)

مثال تجزیه و ترکیب



3- تخصیص

عبارتست از مشخص کردن گونه‌های خاص یک شیئ براساس یک یا چند ضابطه مشخص، مثلا اگر شیئ موجود زنده را درنظر بگیریم، سه گونه خاص آن عبارتند از: انسان، حیوان و نبات. در روش EER هر یک نوع موجودیت می‌تواند خود زیرنوع موجودیتهایی داشته باشد. بین هر زیرنوع و زبرنوع ارتباط "گونه‌ای است از ..." وجود دارد.

4- تعمیم

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

مثلا در مثال صفحه بعد دانشجو به 3 موجودیت تخصیص پیدا می کند .

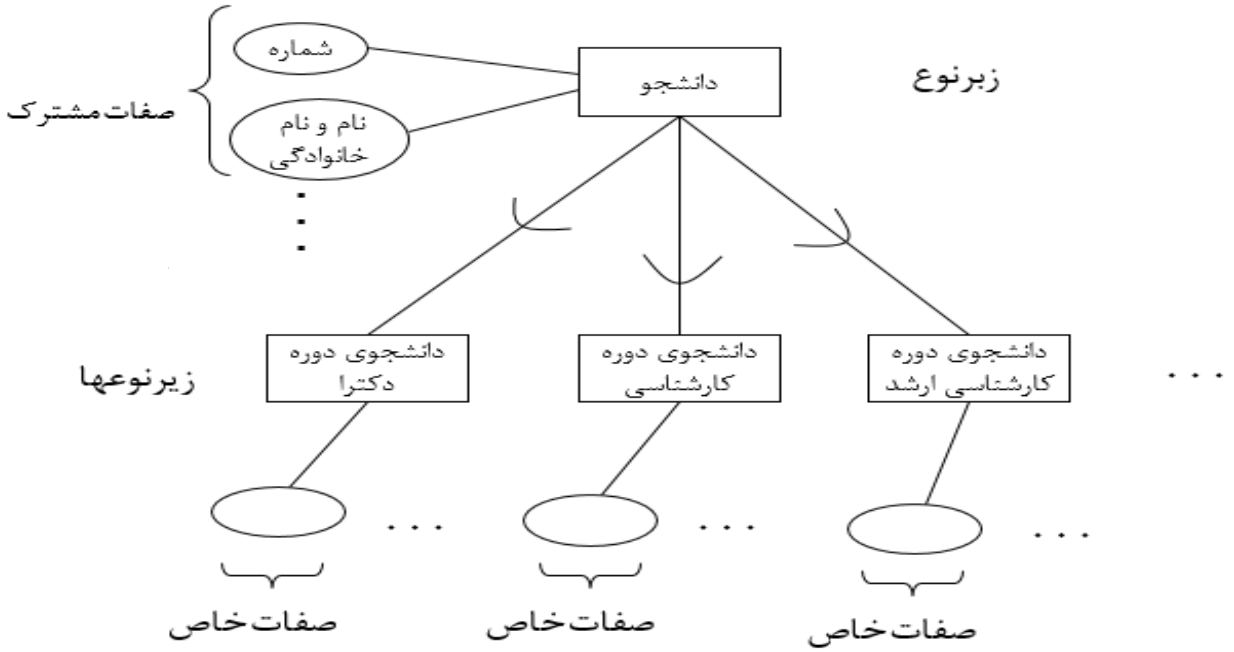
به عبارت دیگر **دانشجو تعمیمی از دانشجوی ارشد است**

دانشجوی ارشد تخصیص پیدا می کند به دانشجو

موجودیت های پایین زیر نوع و موجودیت های بالایی زبر نوع اند .

اگر موجودیت دو تا زبر نوع داشته باشد (گونه ای از دو موجودیت باشد) می گوییم احتیاج به دسته بندی داریم.

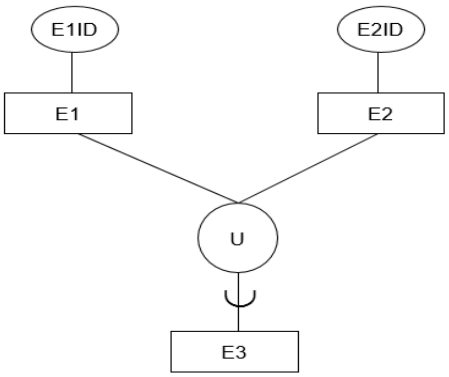
مثال تخصیص و تعمیم



5- دسته بندی

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

دسته‌بندی



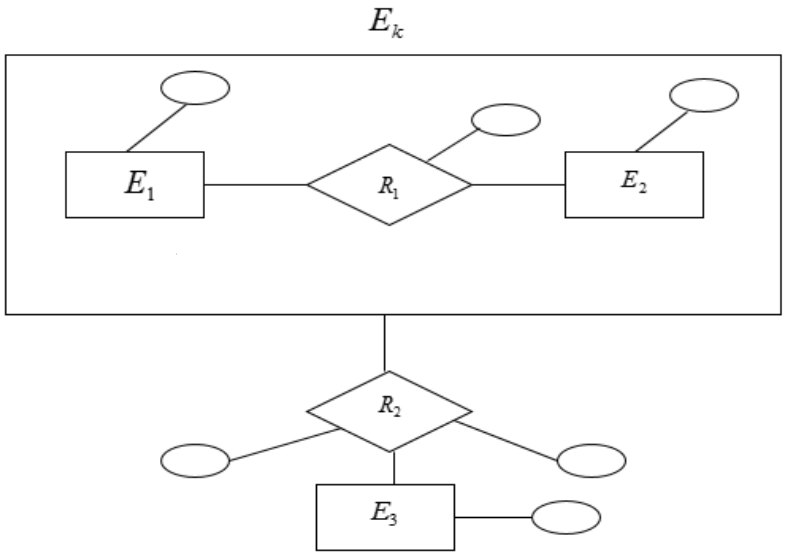
6- تجمع

عبارتست از ساختن یک نوع موجودیت جدید با دیدن دو یا بیش از دو نوع موجودیت، که خود باهم در یک ارتباط شرکت دارند، به صورت یک نوع موجودیت واحد. در واقع مجموعه‌ای از موجودیتهای مرتبط را باهم مجتمع کرده و به عنوان یک نوع موجودیت واحد، درنظر می‌گیریم و این نوع موجودیت واحد خود می‌تواند با نوع موجودیت دیگری ارتباط داشته باشد.

تجمع با ترکیب فرق دارد . در تجمع نه تنها موجودیت های بزرگتر از موجودیت های کوچکتر تشکیل شده بلکه با یکدیگر ارتباط هم دارند (یعنی همه ی موجودیت هایی که تاکنون یاد گرفته ایم را مدل می کنیم و موجودیت بزرگتر را می سازیم).

- **ممکن است موجودیت بزرگتر نیز خود دارای ارتباط با موجودیت های دیگر باشد .**

نمایش تجمع



مراحل مدلسازی معنایی داده‌ها

1- **مطالعه، تحلیل و شناخت محیط**

در این مرحله اول باید راجب سیستمی که می خواهیم برای آن DB طراحی کنیم ، اطلاعات کسب کنیم و این کار به پیدا کردن موجودیت ها کمک می کند.

2- **برآورد خواسته‌ها و نیازهای اطلاعاتی و پردازشی همه کاربران و تشخیص محدودیتهای معنایی(نیاز سنجی)**

از DA از طریق مصاحبه ، پرکردن فرم های مختلف و پرسش سعی می کنیم اطلاعات و نیاز های خود را بدست آوریم و هدف آن شناسایی موجودیت ها ، صفات و ارتباط بین آنهاست.

3- **بازشناسی انواع موجودیتهای مطرح و تعیین وضع هریک**

4- **تعیین مجموعه صفات هر نوع موجودیت**

هر صفتی که مدنظرمان است را از DA می گیریم و ذخیره می کنیم.

5- **بازشناسی انواع ارتباطات بین انواع موجودیتها، تشخیص نوع مشارکت و چندی ارتباط**

یعنی بفهمیم یک موجودیت از چه درجه ای است و ارتباطات آن به چه شکل است.

6- **رسم نمودار ER**

7- **فهرست کردن پرسشهایی که پاسخ آنها از نمودار ER بدست می‌آید.**

موجودیت ها و صفات را بررسی میکنیم که ایا تمامی نیاز های سیستم با این مدل بر طرف شده است یا نه ؟؟ و در واقع این ER چه سوالاتی را پاسخ می دهد.

- **تفاوت این مرحله با مرحله 2 در این است که نیاز سنجی قبل از تشکیل مدل و این مرحله بعد از تشکیل مدل است .**

8- **وارسی مدلسازی انجام شده تا اطمینان حاصل شود که مدلسازی پاسخگوی نیاز کاربران است.**

اضافه کردن موجودیت ها و صفات و ارتباطات در صورت نیاز در این مرحله انجام می شود.

در DBMS تمامی اسامی که برای موجودیت های مدل ER (چه اسامی که برای مدل سازی و چه اسامی که برای پیاده سازی در نظر گرفته شده است) و تمامی نکته ها و Meta Data ها (اطلاعات راجب داده ها) و اسامی جدول ها و اسامی ستونها و .... در Dictionary (کاتالوگ) ذخیره می شود.

اطلاعاتی که در دیکشنری داده‌ها نگهداری می‌شود:

- **شماهای خارجی**
- **شمای ادراکی**
- **شمای داخلی**
- **توابع و رویه‌های مربوط به تبدیلات بین سه سطح معماری**
- **شرح ساختار فیزیکی داده‌های ذخیره‌شده**
- **مشخصات و حقوق دستیابی کاربران به داده‌ها**

با شناسایی نوع کاربر به آن اجازه و دسترسی استفاده از DB داده می شود که اطلاعات مربوط به آن در Dictionary ذخیره می شود.

- **مشخصات برنامه‌های کاربردی تولید شده و ارتباط آنها با درخواستهای کاربران**

اطلاعات مربوط به اینکه چه برنامه کاربردی حق استفاده از DB را دارد.

- **مشخصات پایانه‌های متصل به سیستم**
- **ارتباط بین برنامه‌های کاربردی و داده‌های ذخیره‌شده**
- **قواعد جامعیت**

قواعد مربوط به داده ها در DB (اجازه خراب شدن داده های DB را نمی دهد.)

- **ضوابط کنترل ایمنی داده‌ها**

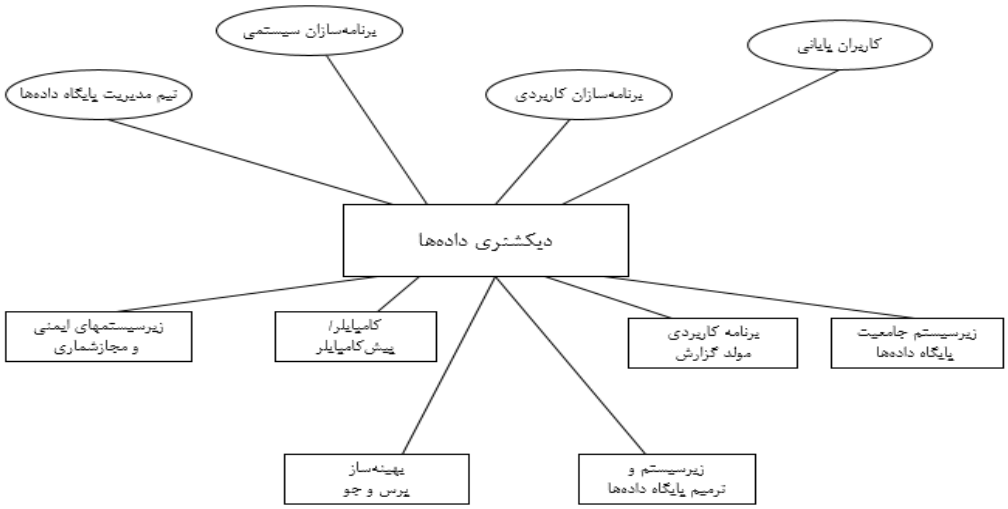
ضوابطی که برای ایمنی تعریف می شود. مثلا اینکه یک کاربر نتواند خارج از حدش مثلا جدولی را حذف کند در واقع کار بر هایی که در سیستم هستند را با حقوق دسترسی متفاوت کنترل می کند.(اجازه خراب شدن Schemaها یعنی اطلاعات مربوط به ستون های جدول را در DB را نمی دهد.)

- **مشخصات پیکربندی سخت‌افزاری سیستم و رسانه‌های ذخیره‌سازی**
- **اطلاعات متنوع آماری پایگاه داده و کاربران**

مثلا چند درصد دانشجویان به موقع فارغ التحصیل شده اند ، پایین تربن نمره ، میانگین نمرات و ... که در اختیار EU ها قرار می گیرد.

-
- **توابع تعریف‌شده توسط کاربران**

دیکشنری داده‌ها و استفاده‌کنندگان آن



جلسه پنجم:

پایگاه داده شامل دو قسمت است :

1- Schema -- > اطلاعات مربوط به سطر ها و ستون های جدول های موجود در پایگاه داده

2- Data -- > داده های موجود در پایگاه

- داده ها در Dictionary ذخیره نمی شوند بلکه اطلاعات راجب داده ها (Meta Data) در آن ذخیره می شود. به عبارت بهتر شمایی از پایگاه داده در آن ذخیره می شود. (اینکه چند تا جدول داریم ، چند تا ستون داره و .... اینا توی دیکشنری ذخیره میشه)
- اسم ها (توجه کنید که اسامی پیاده سازی با اسامی ذخیره شده ممکن است متفاوت باشد) و صفت ها و اینکه این صفت ها از چه نوع است ، همگی در دیکشنری ذخیره می شود.
- دیکشنری با قسمت های مختلف در ارتباط است و فقط اطلاعات در مورد این قسمت ها را در خود نگهداری می کند و نمی تواند آنها را پیاده سازی کند. (حتی اطلاعات مربوط به کامپایلر ها)
- کاربران با داده هایی که در سیستم ذخیره شده اند سرو کار دارند و با سطر ها و ستونهای جدول کاری ندارند.

تراکنش (Transaction) :

- **هر برنامه‌ای که توسط کاربر در محیط بانک اطلاعاتی اجرا می‌شود**
- **واحد منطقی کار**

در برنامه نویسی همان اضافه کردن دستورات است ولی در سیستم پایگاه داده همان پردازه است.

- **معمولا شامل چندین دستور العمل که یک کار را برای ما انجام می دهند**
- **مانند انتقال وجه**
- **داده مهمتر است از برنامه**

برای کاربرانی که با سیستم عامل کار می کنند ، برنامه مهم تر است تا داده های آن .

مثلا اگر یک نفر در حین انجام بازی برنامه اش قطع شود ، خود برنامه نسبت به داده های آن برایش اهمیت بیشتری دارد اما در سیستم پایگاه داده ، داده ها مهم تر از برنامه ها هستند .

یعنی اگر برنامه قطع شد داده ها از بین نروند و به عبارت دیگر داده ها در پایگاه داده باید سازگار باشند و ایجاد اختلال در برنامه منجربه خراب شدن و یا تغییر داده ها نشود.

در اینجا ممکن است DBP ها بگویند که برای حفظ سازگاری داده ها ، اینترفیس ها و یا برنامه ها را ببندیم . اما در حالت کلی سیستمی مناسب است که در عین سازگار بودن داده هایش برنامه ها نیز در آن به درستی اجرا شوند. (بنابراین اگر مشکلی پیش آمد ، برنامه را قربانی داده ها می کنیم. ☺)

- **دارای خاصیت ACID است**

به عبارت خلاصه :

تراکنش واحد منطقی کار در پایگاه داده است که دارای خاصیت ACID باشد . اگر هر مجموعه عملیات یکی از چهار ویژگی زیر را نداشته باشد ، تراکنش نیست.

- **Atomicity: یکپارچگی**

یا همه دستورات انجام شود و یا اگر قسمتی از دستورات اجرا نشد ، بقیه دستورات را اجرا نکند و در اصطلاح همه دستورات به عقب برگردد. (roll back)

مثلا در انتقال وجه از حساب A به حساب B در سیستم بانکی ، اگر تنها از حساب A کم شود و به حساب B اضافه نشود ، چون عمل انتقال درست انجام نشده است ، باید اطلاعات roll back شود.

➢ **Consistency: سازگاری**

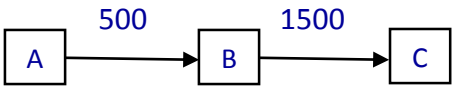
یعنی بعد از اجرای تمامی دستورات حتی اگر همه اجرا شود ، باید یکپارچگی رعایت شود و داده های داخل پایگاه تغییر نکند.
به عبارت بهتر ؛ صحت اعتبار داده های داخل پایگاه داده تغییر نکند.

➢ **Isolation: انزوا**

هر تراکنش باید به گونه ای باشد که تراکنش های دیگر را تحت تاثیر خود قرار ندهد یا اگر مجموعه ای از تراکنش ها به صورت همزمان و هم روندا اجرا شوند ، نتیجه آن با زمانی که تک تک تراکنش ها به صورت جداگانه اجرا شوند ، فرقی نداشته باشد .

مثلا در همان مسئله انتقال وجه ؛ اگر 500 تومان از A به B و از 1500 تومان از حساب B به حساب C منتقل شود ، نتیجه نهایی باید به صورت زیر باشد و هر نتیجه ای غیر از این تراکنش محسوب نمی شود :

از حساب A ، 500 کم شده
از حساب B ، 1000 کم شده
به حساب C ،1500 اضفه شده



➢ **Durability: پایائی**

اگر نتیجه تراکنش در سیستم پایدار باشد، دارای این ویژگی است .

- اگر تراکنش با موفقیت انجام شد (commit) ، نتیجه آن در پایگاه داده ثابت باقی می ماند . (یعنی اگر تغییری در سیستم انجام شده تغییر نمی کند مگر آنکه تراکنش دیگری آنها را تغییر دهد .
- اگر تراکنش با موفقیت انجام نشد ، roll back می شود.

CASE (Computer Aided Software Enginiering -- > نرم افزار های مهندسی کمکی کامپیوتر)

در واقع نرم افزار هایی اضافه بر نرم افزار های DBMS هستند که برای کمک به طراحی و پیاده سازی پایگاه داده و یا نرم افزار هایی برای مدیریت و کار با دیکشنری و یا برای تهیه گزارش ها (چه گزارش های روی داده های ذخیره شده و چه گزارش هایی روی Meta data ها (مثلا داده در چه سطر و ستون جدول ذخیره شده)) و نیز برای رسم نمودار ها EER استفاده می شوند.

BI -- > مجموعه نرم افزار های مربوط به هوش تجاری که به DB وصل می شود و آمار های تخصصی از DB به ما می دهند. (مثلا چند درصد از کارمندان به موقع کارهایشان را انجام داده اند.)

مزایای سیستم بانک اطلاعات

- **مدل سازی داده‌ها بر اساس مفاهیم آن‌ها**

کمک می کند که دید درستی نسبت به سیستم داشته باشیم .
** در مشی فایلینگ ما مدل سازی نداشتیم اما در سیستم پایگاهی ، مدل سازی داریم .

- **وحدت ذخیره‌سازی کل داده‌ها**

کل داده ها در یک سطح به نام سطح ادراکی در نظر گرفته می شود و در اختیار دید های مختلف قرار می گیرد .

- **اشتراکی شدن داده‌ها**

ممکن است همه کاربران از یکسری اطلاعات مشترک استفاده کنند.

- **کاهش میزان افزونگی**

جون داده ها در یک جا ذخیره می شوند به حالت مجتمع هستند و دیگر افزونگی اطلاعات نداریم .

- **تسهیل دستیابی به داده‌ها**
- **عدم وجود ناسازگاری در داده‌ها**

به علت نبود افزونگی و نیز به علت سازگار بودن تراکنش ها داده های داخل پایگاه دچار ناسازگاری نمی شوند.

- **تضمین جامعیت**
- **امکان اعمال ضوابط دقیق ایمنی**
- **تأمین استقلال داده‌ای**

داده ها وابسته به مکان ذخیره سازی در حافظه نیستند یعنی اگر پارتیشن یا درایو را عوض کنیم ، تاثیری در پایگاه داده و برنامه های متصل به آن ندارد.

- **حفظ محرمانگی داده‌ها**

با وجود اینکه دید های ادراکی شامل همه داده ها هستند اما چون دید های خارجی زیر مجموعه ای از آنها هستند ممکن است به همه اطلاعات دسترسی نداشته باشند و به هر قسمت اطلاعات مربوط به آن قسمت داده می شود.

- **تسهیل توسعه و رشدپذیری**
- **تسریع دریافت پاسخ پرسش و جوها**

چون سرعت DBMS بالا است و می تواند به سرعن گزارشات آماری و تراکنش ها را بررسی کند ، سرعت پرس و جو در آن سریع است.

- **تسهیل در ارائه گزارشات آماری**
- **پشتیبانی از تراکنش**

معایب سیستم بانک اطلاعات

- **امکان نیاز به سخت‌افزاز و نرم افزاراضافی**

در سیستم پایگاهی چون اطلاعات در یک جا هستند با از دست رفتن سیستم کلیه اطلاعات از بین می رود پس نیاز به سخت افزار و نرم افزار های اضافه دارد.

- **آسیب‌پذیر بودن امنیت به خاطر متمرکز بودن داده‌ها**

در سیستم های توزیعی چون داده ها در سیستم های جدا قرار می گیرد ، کمتر در خطر است اما در سیستم پایگاهی به علت متمرکز بودن داده ها امنیت کمتر است.

- **پیچیدگی برخی تراکنش‌ها و عملیات**
- **هزینه‌های نصب DBMS**

البته نه تو ایران ☹

بعد از اینکه data model را ارائه دادیم و مدل مشخص شد باید data structure یا ساختار داده ای را مشخص کنیم .
تا اینجا Modeling (ارائه مدل هایی برای مدل سازی) را گفتیم که بعضی ها آن را با data model هم عرض می دانند . اما خیلی از طراحان پایگاه داده ، data model را مجزا از modeling می دانند.

در واقع :

ساختار داده‌ای، امکانی است **برای نشان دادن داده‌های** در مورد انواع موجودیت‌ها و **انواع ارتباطات بین آنها**

DM یا مدل داده ای از سه قسمت زیر تشکیل شده است :

1- **ساختار داده ای**
2- **عملگرها** --> $select - insert - \cap - \cup$ (عملگر هایی که روی داده ها اعمال می شوند.)
3- **قواعد جامعیت** --> شامل قوانین و مقررات حاکم بر داده ها

- اکثرا از ساختمان داده ای رابطه ای استفاده می شود مگر آنکه حجم داده ها خیلی زیاد باشد که در این صورت از no SQL استفاده می شود.
- برای هر سه سطح داخلی ، ادراکی و خارجی مدل داده ای داریم که این مدل ها می توانند متمایز و یا مشابه هم باشند ولی اکثرا مدل هایی که برای این سه سطح در نظر میگیرند ، مشابه اند.

با توجه به این ساختار مثلا می توان ساختار داده ای ادراکی را سلسله مراتبی ، و ساختار داده ای خارجی را رابطه ای در نظر بگیریم ولی کم پیش می آید .

ساختار داده‌ای رابطه‌ای

رابطه مفهومی **ریاضی** است. اما از دید کاربر، رابطه نمایشی **جدولی** دارد. (ساختاری که در آن همه چیز با جدول ها نشان داده می شوند که سطر های جدول حاوی داده های مربوط به نمونه ها و صفت ها را در ستون ها نمایش داده می شوند.)

به عبارت بهتر مفاهیم ساختار جدولی عبارتند از:

جدول (عنصر اصلی این ساختار) **سطر** **ستون**

شمای پایگاه جدولی

شمای پایگاه داده‌ها عبارتست از تعریف (توصیف) ساختهای انتزاعی طراحی‌شده. نوعی برنامه است شامل **دستورات تعریف داده‌ها** (دستوراتی که به خود داده ها کاری ندارند بلکه مربوط به سطر ها و ستون های جدول هتند مثلا دستورات مربوط به اضافه یا حذف سطر یا ستون جدول) و **کنترل داده‌ها** ، و **دستورات عملیات در داده‌ها** در آن وجود ندارد.

برخی ویژگیهای ساختار داده‌ای جدولی

1- از نظر کاربر نمایش ساده‌ای دارد.

2- داده‌ها و ارتباط بین آنها با مکانیسم واحدی نشان داده می‌شوند.

هر دو با جدول نشان داده می شوند.

3- عنصر ساختاری اساسی آن جدول است.

4- همه چیز با فقره داده‌های ساده نمایش داده می‌شود.

5- ارتباطات با چندیهای مختلف در آن قابل نمایش است.

با استفاده از جدول می توان انواع ارتباط 1-1 و 1- N و N-N را نشان داد.

6- منطق بازیابی آن ساده است.

میتوان روی آن به راحتی quary بزنیم.

7- در عملیات ذخیره سازی دشواری (Anomaly) ندارد مگر آنکه انجام آن عملیات سخت یا غیر ممکن باشد و سبب بروز وضعیت نامطلوب نمی‌شود.

دشواری در درج ، حذف و ویرایش جدول ها ندارد .

8- برای پرسشهای قرینه (پرس و جوهایی که عکس هم اند) **، رویه پاسخگوی قرینه دارد.**

مثلا در ارتباط بین دانشجو و درس یبار با داشتن دانشجو می توان به درس های آن دست یافت و بار دیگر با داشتن درس می توان به دانشجویانی که این درس را داشتند دسترسی پیدا کرد.

9- مبنای تئوریک قوی دارد (چون توسط مفاهیم ریاضی قابل تعریف است) **و بنابراین دارای تئوری نرمالسازی است.**

در این ساختار داده می توان همه صفت ها را در یک جدول قرار دهیم یا برای هر کدام یک جدول جداگانه در نظر بگیریم .

مثال

Student			
ID	Name	Family	Ave
1	اکبر	عبدی	19
2	مهناز	افشار	11

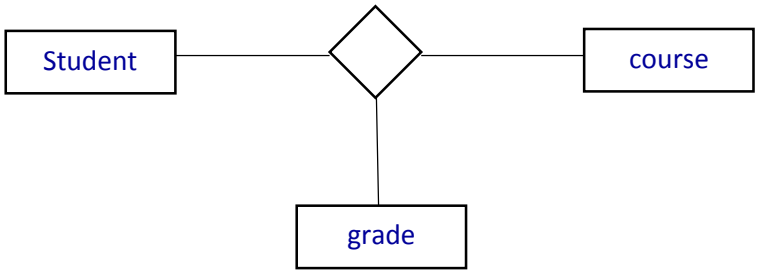
Cource			
ID	Name	Unit	Type
1	ریاضی 1	4	پایه
2	DB	3	الزامی

CS (ارتباط بین دو موجودیت بالا)

SID شناسه دانشجو	CID شناسه درس	Grade
1	1	20
2	3	10

یعنی اکبر عبدی درس ریاضی 1 را گرفته و نمره آن 20 شده است

- یک نمونه از موجودیت در یک سطر از جدول قرار میگرد و در واقع مشخصات نمونه های موجودیت در سط های جدول ذخیره می شود.



ساختار داده‌ای سلسله‌مراتبی

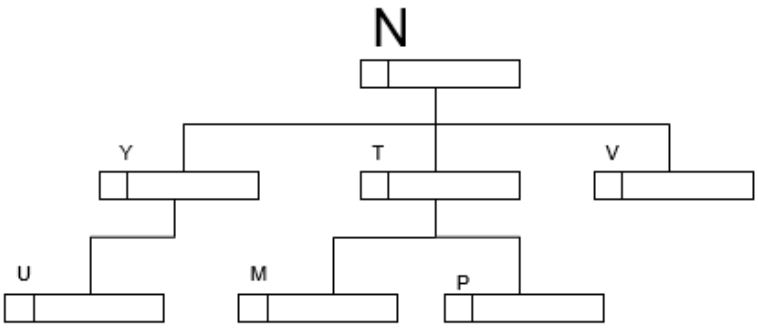
عناصر ساختاری اساسی

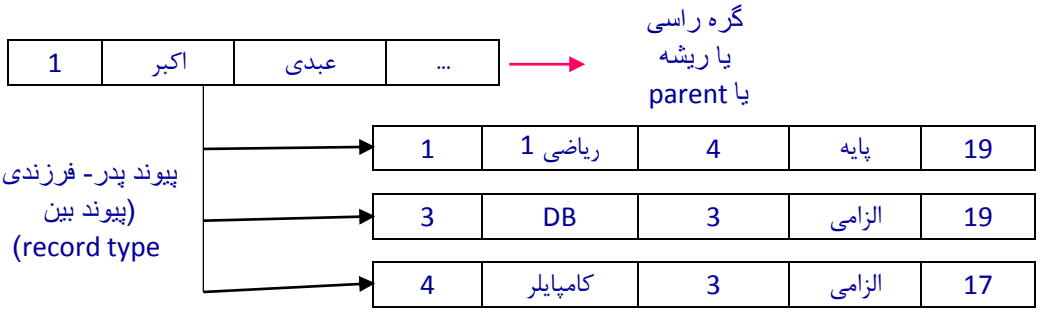
1- نوع رکورد 2- نوع پیوند پدر- فرزندی

نوع رکورد برای نمایش **نوع موجودیت** به کار می‌رود.

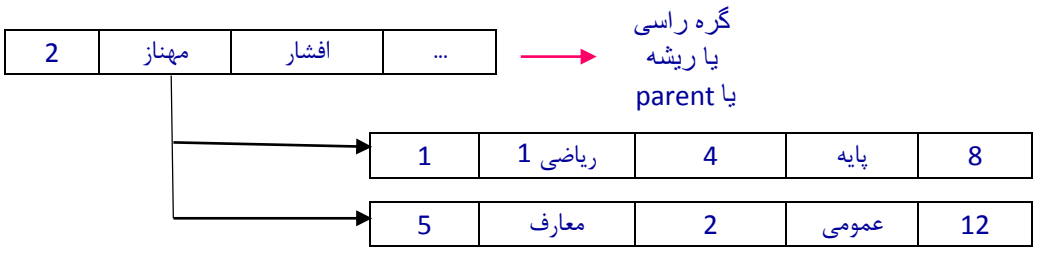
ارتباط بین دو نوع موجودیت به وسیله پیوند پدر– فرزندی نمایش داده می‌شود.

ساختار داده‌ای سلسله‌مراتبی





به هر کدام از این Nodeها، record type گفته می شود.



- در مدل سلسله مراتبی نمایش جدول نداریم و هر کدام که اینجا کشیدیم خود یک آرایه جدا است.
- بین گره های راسی هیچ ارتباطی وجود ندارد.
- در ساختار رابطه ا نمی توان سطر تکراری داشت ولی در مدل سلسله مراتبی می توان **record** تکراری داشته باشیم .

برخی ویژگیهای ساختار داده‌ای سلسله‌مراتبی

1- سادگی نمایش ساختار جدولی را ندارد.

2- مبنای ریاضی ندارد.

3- **دو عنصر ساختاری اساسی دارد.**

یکی برای نشان دادن نمونه ها و دیگری برای نشان دادن ارتباط بین نمونه ها

4- **ارتباط یک به چند را نمایش می‌دهد.**

برخلاف مدل رابطه ای که در آن تمامی چندی ارتباط ها نشان داده می شد ، در اینجا چون یک parent میتواند N ، child داشته باشد پس فقط برای نمایش یک به چند مناسب است .

5- **نمایش ارتباط چند به چند در آن دشوار است.**

ممکن است ماهیت یک ارتباط چند به چند باشد مانند ارتباط بین درس و دانشجو ولی چون در اینجا بین parent ها ارتباطی وجود ندارد ، نشان دادن ارتباط های چند به چند دشوار است و بیشتر ارتباط های این مدل یک به چند است .

6- **ناهمگن است.**

7- **روش پاسخ به پرس و جو های قرینه یکسان نیست .**

مثلا با داشتن دانشجو به راحتی می توانیم به درس های او دسترسی پیدا کنیم اما با داشتن درس برای آنکه به دانشجویان آن دسترسی پیدا کنیم باید کل DB را بررسی کنیم و دانشجویانی که آن درس را گرفته اند ، پیدا کنیم .

8- **در عملیات ذخیره‌سازی مشکلاتی (آنومالی) دارد.**

در این ساختار اگر بخواهیم parent جدید اضافه کنیم باید یک ریشه تنها (root only) در نظر بگیریم که در این نوع درج مشکلی نداریم . مثلا یک دانشجوی تنها را می توان در پایگاه درج کرد اما یک درس را تا زمانی که یک دانشجو آن را نگیرد و به اصطلاح به عنوان فرزندی برای یک ریشه نباشد ، نمی توان آن را به پایگاه داده اضافه کرد . بنابراین در **درج فرزندان** به ساختار آنومالی داریم .

همچنین در این ساختار در **حذف** نیز آنومالی داریم چون ممکن است با حذف از اطلاعات ، پایگاه هم حذف شود که منجر به بروز ناسازگاری می شود.

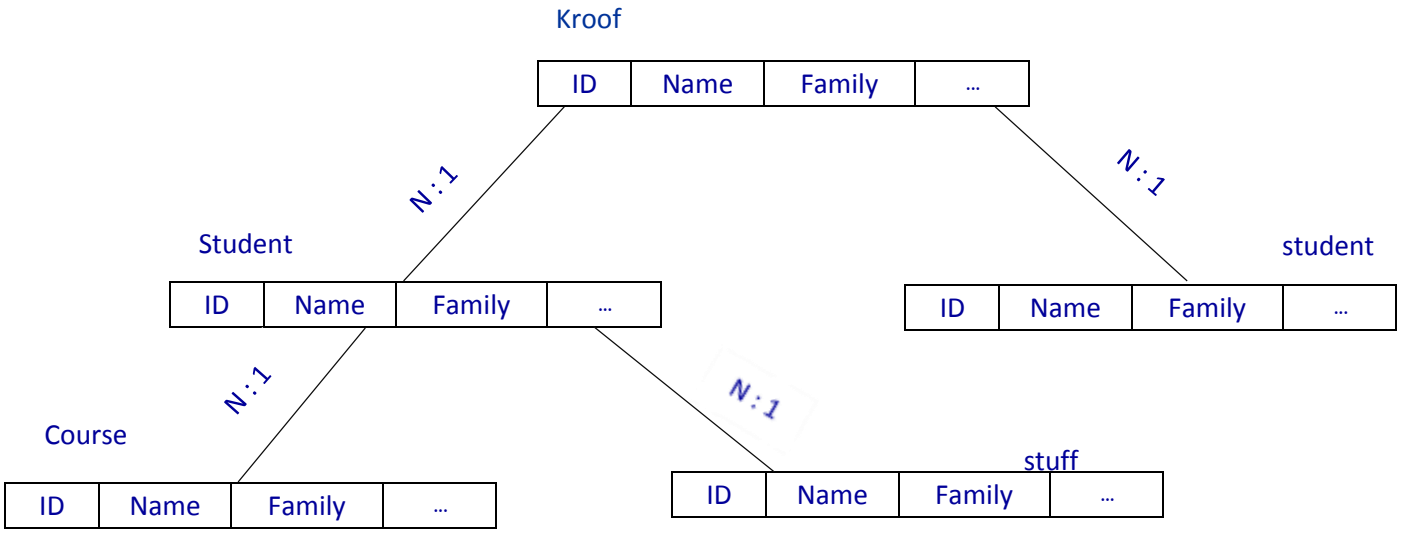
در **ویرایش** نیز مثلا با داشتن اطلاعات فرزندان یک parent برای اطلاعات دیگر درخت ها ناسازگاری پیش بیاید .(مثلا اگر واحد یکی از درس های یک دانشجو را تغییر دادیم باید در کل پایگاه بگیردیم و هر جا آن درس را داشتیم واحدش را عوض کنیم .)

9- **تقارن ساختار جدولی را ندارد.**

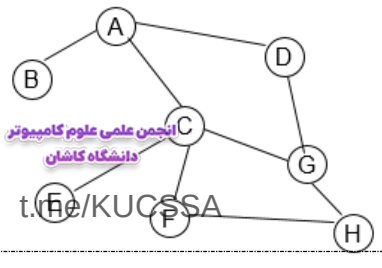
10- **برای بازیابی نیاز به عملگر ریشه یاب و وابسته یاب دارد.**

11- **در مواقعی که در ذخیره‌سازی نمونه‌های فرزند، افزونگی پدید آید، پایگاه در معرض ناسازگاری قرار می‌گیرد.**

12- **در این ساختار هر گره حداکثر یک parent دارد.**



ساختار داده‌ای شبکه‌ای



ساختار داده‌ای شبکه‌ای

عناصر ساختاری اساسی

1- نوع رکورد **2- نوع مجموعه**

نوع رکورد برای نمایش نوع موجودیت به کار می‌رود.

نوع مجموعه **برای نمایش ارتباط 1:N** بین دو (چند)
نوع موجودیت به کار می‌رود.

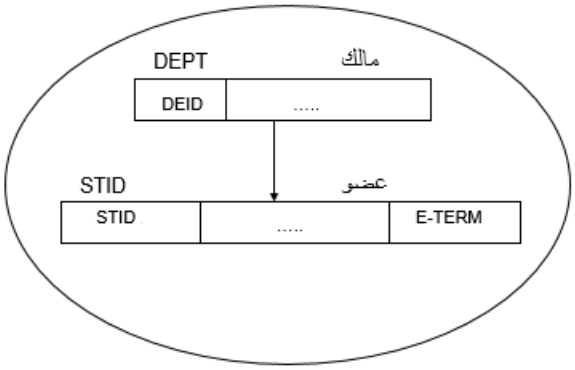
نوع مجموعه (مجموعه کوداسیل) از سه جزء تشکیل شده است:

1- نام مجموعه

2- یک نوع رکورد مالک (owner record type)

3- یک نوع رکورد عضو (member record type)

یک نوع مجموعه کوداسیل



- در این ساختار بین هر دو owner که یک member مشترک داشته باشند ، **ارتباط چند به چند** وجود دارد.

برخی ویژگیهای ساختار داده‌ای شبکه‌ای

1- سادگی ظاهری ساختار داده‌ای جدولی را ندارد.

2- مبنای ریاضی ندارد.

3- دو عنصر ساختاری اساسی دارد.

4- ماهیتا خاص نمایش ارتباطات "یک به چند" نیست.

به کمک آن می توان ارتباط چند به چند را هم نشان داد .

5- دستور بازیابی آن پیچیده‌تر از ساختارهای دیگر است

6- مثل ساختار داده‌ای جدولی تقارن دارد.

7- خطر بروز ناسازگاری داده‌ها نسبت به ساختار سلسله‌مراتبی، کمتر است.

8- اصل وحدت عملگر در عمل درج ندارد.

هم ریشه یاب و هم وابسته یاب را دارد .

9- مبانی تئوریک ریاضی ندارد

10- آنومالیهای مدل سلسله‌مراتبی در عملیات ذخیره‌سازی را ندارد.

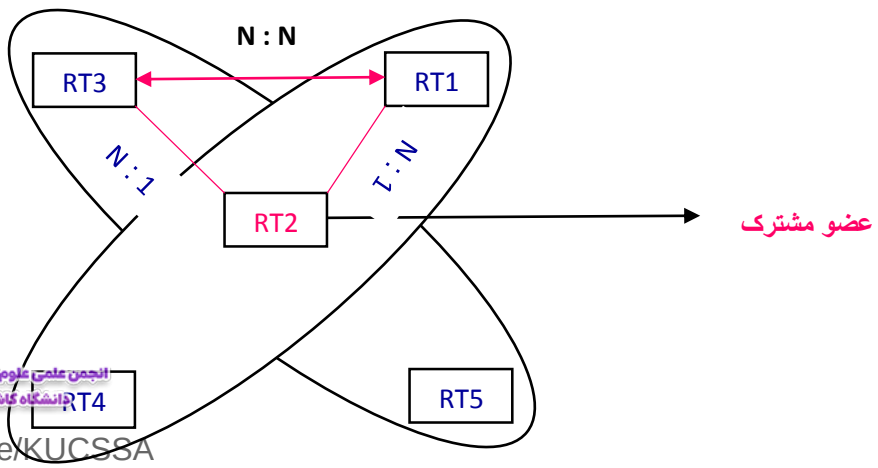
مثلا برای حذف یک نمره بدون آنکه دانشجو یا درس را حذف کنیم میتوان آن را از ساختار حذف کرد. پس آنومالی **حذف** نداریم .

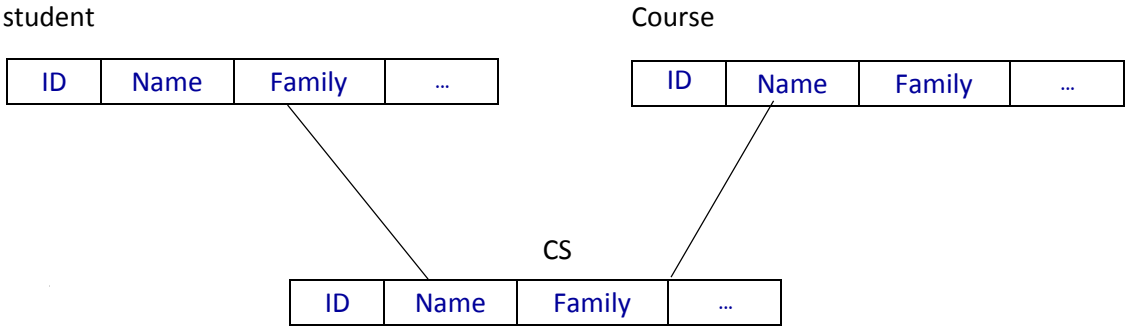
آنومالی **درج** و **ویرایش** هم نداریم.

11- افزونگی نسبت به سلسله مراتبی کمتر است .

12- برای پرس و جو های قرینه جواب یکسان دارد . اما پاسخ به آنها دشوار است .

نکته -- > مدل سلسله مراتبی جزئی از مدل شبکه ای است .





در اینجا

ارتباط بین درس و دانشجو : چند به چند

ارتباط بین درس و نمره : یک به چند

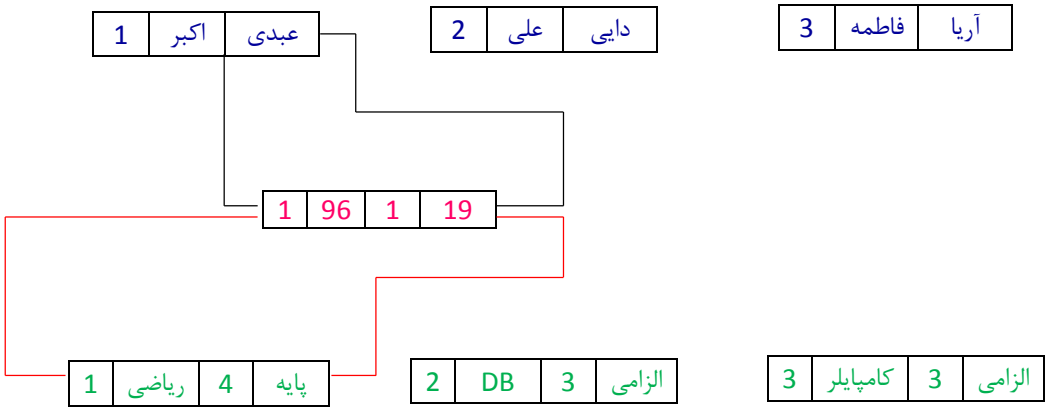
ارتباط بین دانشجو و نمره : یک به یک

الگوریتم درج یک گره در ساختار جدید به جز گره root only :

از آخرین member یک owner یک ارتباط به member جدید و سپس اتصال آن به owner

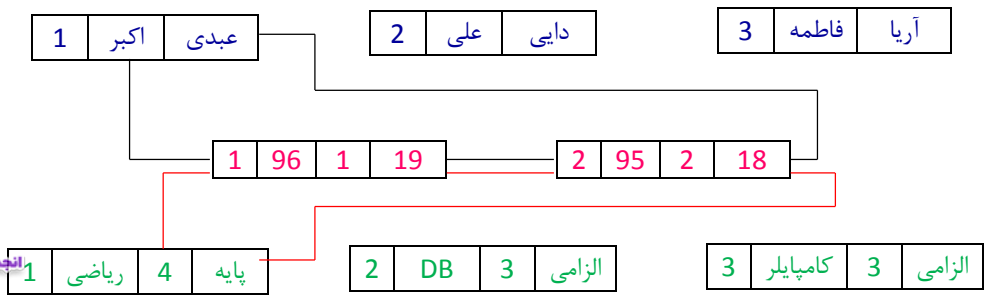
مرحله اول : میخواهیم گره | 1 | 96 | 1 | 19 | را جز فرزندان | عبدی | اکبر | 1 | و | پایه | 4 | ریاضی | 1 |

قرار دهیم . که چون هیچ کدام از این گره های راسی فرزندی ندارند تنها کافی است از member به owner ها وصل کنیم

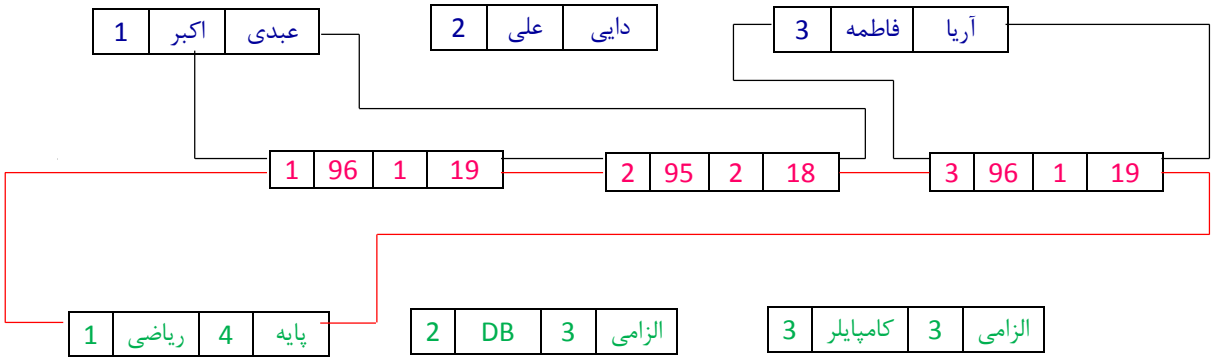


مرحله دوم : میخواهیم گره | 2 | 95 | 2 | 18 | را جزوفرزندان | عبدی | اکبر | 1 | و | پایه | 4 | ریاضی | 1 |

قرار دهیم . اما چون این دو گره راسی دارای فرزند دیگری هم هستند که آن را در مرحله قبل اضافه کردیم ، اکنون باید ابتدا بین این دو member ارتباط برقرار کنیم . که برای این کار از member قبلی به member جدید وصل کرده و سپس از member جدید به دو گره راسی وصل می کنیم .



مرحله سوم : می خواهیم گره | 3 | 96 | 1 | 19 | را جز فرزندان | 3 | فاطمه | آریا | و | 1 | ریاضی | 4 | پایه | قرار دهیم . اما چون یکی از این دو گره راسی دارای فرزندان دیگری هم هست که در مراحل قبل اضافه کردیم ، اکنون باید بین این چند member ارتباط برقرار کنیم پس برای این کار باید آخرین فرزند گره را به گره جدید متصل و از این گره به parentارتباط برقرار کنیم .



- **در مدل شبکه ای آنومالی وجود ندارد و نیز ساختار ریاضی هم ندارد.**
- **مدل شبکه ای برای نشان دادن ارتباطات چند به چند مناسب است.**
- **الان بیشتر از ساختار داده ای رابطه ای استفاده می شود و نوع دیگری از ساختار داده ، غیر رابطه ای است که برای big data ها استفاده می شود .**

(زمان یکه حجم خود اطلاعات انقدر زیاد (در حد ترابایت) باشد که اگر بخواهیم رابطه های بین آنها را به آن اضافه کنیم ، حجم خیلی بیشتر می شود بنابراین از مدل غیر رابطه ای استفاده میشود)

- **اما اگر حجم اطلاعات را در نظر نگیریم ، بهترین مدل ، مدل رابطه ای و مدل رابطه ای شی گرا است .**

جلسه هفتم

همانطور که در جلسات قبل گفتیم ، مدل داده ای شامل سه قسمت است :

1- **ساختار داده ای** (خیلی ها این و مدل داده ای را معادل هم میدانند)

2- **عملگر هایی که روی ساختار داده ای اعمال می شوند .**

در ابتدا باید ساختار داده ای تعریف شود و سپس عملگر های آن

مثلا پیش از آنکه عملگر های ریاضی را معرفی کنیم ، جبر ریاضی را تعریف می کنیم یا پیش از آنکه عملگر های بول یرا معرفی کنیم ، جبر بولی را تعریف می کنیم . در اینجا نیز پیش از آنکه عملگر ها را روی مدل رابطه ای ارائه بدهیم ، خودمان مدل رابطه ای را معرفی می کنیم که به صورت ریاضی وار به چه معنایی است .

3- **قواعد جامعیت**

مدل رابطه ای

مدل رابطه‌ای در یک مقاله تحصیلی توسط ادگار کاد (ریاضی‌دان) در سال ۱۹۷۰ ارائه شد

- **مدل رابطه ای مبنای ریاضی دارد .**
- **مدل رابطه ای و رابطه ای شی گرا برای حجم معمولی از اطلاعات (مثلا اطلاعاتی که در سیستم دانشگاهی استفاده می شوند و یا در شرکت های مختلف یا در بنگاه مسکن) استفاده می شود .**

**** تفاوت مدل رابطه ای شی گرا با شی گرایی چیست ؟**

در مدل شی گرایی به جای آنکه جدول در نظر بگیریم ، به هر موجودیت به عنوان یک شی نگاه می شود که این شی نه تنها صفت دارد بلکه یک سری رفتار هایی هم دارد که ما حتی می توانیم این رفتار ها را در DB ذخیره کنیم . یعنی نه تنها می توان اطلاعات را ذخیره کرد ، بلکه یک سری توابعی هم دارد که می شود آنها را ذخیره کرد (مثلا برای دانشجوی ثبت نام ، تابع ثبت نام سبب می شود که DB تغییر کند.) اما ارتباط بین اینها به عنوان رابطه در نظر گرفته نمی شود . بلکه ارتباط بین آنها به عنوان متد است (مثلا در مثال بالا ثبت نام ارتباط بین دانشجو و درس است)

پس در مدل شی گرایی ارتباط به عنوان متد اما در مدل شی گرایی رابطه ای ، ارتباط به عنوان رابطه در نظر گرفته می شود.

رابطه در مدل رابطه ای شامل بخش های زیر است :

1- **دامنه: مجموعه‌ای که مقادیر یک صفت خاصه از آن گرفته می‌شود.**

رابطه ها روی دامنه ها تعریف میشوند و به عبارت خلاصه تر دامنه مجموعه ای از مفادیر صفات است (یعینی هر کدام از صفت ها چه مقداری را می توانند به خود بگیرند ؟؟)

مثلا دامنه نام و نام خانوادگی ، مجموعه ای از کاراکتر ها است .

- هر کدام از صفات دامنه مشخصی دارند .
- هر نمونه از یک موجودیت باید صفاتش را از دامنه آن موجودیت بگیرد .

2- **رابطه: زیرمجموعه‌ای از ضرب دکارتی چند دامنه**

فرض کنید که صفات نام – نام خانوادگی – آدرس – تلفن و ..... را داریم که هر کدام از این صفات دامنه خاص خود را دارد . رابطه بین آنها مجموعه ای از چند تایی های مرتب است

$$\{(a,b,c,d,\dots.)|a \in \text{دامنه نام}, b \in \text{دامنه نام خانوادگی}, c \in \text{دامنه آدرس}, d \in \text{دامنه تلفن}, \dots\dots\dots\}$$

3- **تاپل: به اعضای رابطه (یا هر سطر از رابطه)تاپل گفته می‌شود.** (معادل همان دیتا های رابطه هستند)

4- **کاردینالیتی رابطه: تعداد تاپل‌های رابطه در یک لحظه از حیات آن**

5- **درجه رابطه: تعداد ستون‌های رابطه**

هر رابطه در ساختار رابطه ای شامل دو بخش زیر است :

1- مجموعه ای از داده ها که مجموعه ای از تاپل ها است .

2- مجموعه از عناوین یا header که همان schema پایگاه داده را برامون می سازد .

خصوصیات رابطه

- **رابطه شامل دو مجموعه عنوان (header) و پیکر (body) است.**
- **تاپل و عنوان تکراری نداریم**
- **تاپل‌ها در رابطه نظم ندارند**

ترتیب تاپل ها اهمیتی ندارد. مثلا اگر جای دو سطر یا جای دو ستون را با هم عوض کنیم تغییری در DB به وجود نمی آید .

نکته مهم -- > مقادیر تاپل ها را نمی توان جا به جا کرد اما خود تاپل ها را می توان جا به جا کرد .

- **صفات خاصه نظم ندارند**
- **همه مقادیر صفات خاصه تجزیه ناپذیرند(اتمیک). (تاپل نمی‌تواند شامل تاپل باشد)**

مثلا وقتی مجموعه ای از اعداد را داریم ، هر عضو مجموعه فقط میتواند عدد باشد و نمی تواند مجموعه ای دیگر از اعداد باشد .

پس در رابطه مجموعه ما باید چند تایی مرتب باشد و خود نمی تواند شامل یک چند تایی مرتب در بین آنها باشد .

- **همه مقادیر صفات خاصه تک مقداری هستند**
- **رابطه نرمال(استاندارد): همه مقادیر صفات خاصه اتمیک‌اند**

در این نوع از رابطه ، نظر DA ها مهم است . مثلا ممکن است آدرس دقیق دانشجو برای DA ها مهم نباشد ؛ پس صفت آدرس یک صفت اتمیک است و این رابطه یک رابطه نرمال است (مثلا نمیخواهیم دانشجو ها را بر اساس شهر یا کوچه جستجو کنیم که اگر بخواهیم این کار را انجام دهیم صفت آدرس باید تجزیه شود که اگر این اتفاق نیفتد رابطه نرمال نیست چون یک صفت تجزیه پذیر دارد)

صفت کلیدی :

صفاتی که مقدار آنها برای موجودیت ها منحصر به فرد باشد (تکراری نمی تواند باشد)

مثلا شماره دانشجویی یه صفت کلیدی است چون هر یک از نمونه ها (دانشجویان) شماره دانشجویی منحصر به خود را دارند.

انواع کلید

1- ابرکلید (Super Key) (S.K.)

زیر مجموعه ای (یعنی می تواند یک عضو یا چندین عضو داشته باشد) از صفات که دارای خاصیت کلیدی هستند .

زیر مجموعه سه عضوی از صفات -- > (شماره شناسنامه ، نام خانوادگی ، نام)

- نام به تنهایی ابر کلید نیست چون امکان تکراری بودن در آن هست .
- (کد ملی ، نام) برای هر فرد منحصر به فرد است پس ابر کلید است.

2- کلید کاندید (Candidate Key) (C.K.)

ابر کلیدی که دارای خاصیت کمینگی است .

یعنی زیر مجموعه ای از صفات که دارای خاصیت کلیدی باشد و هیچ زیر مجموعه ای از آن خاصیت کلیدی نداشته باشد .

یعنی نتوانیم هیچ عضوی از آن را حذف کنیم و همچنان خاصیت کلیدی باقی بماند .

- (کد ملی ، نام) ابر کلید است اما کلید کاندید نیست چون یک زیر مجموعه ای از آن (نام) وجود دارد که خاصیت کلیدی ندارد .

تعریف دیگر -- > مقادیر موجودیت ها تکراری نباشند و دارای خاصیت کمینگی باشند .

3- کلید اصلی (Primary Key) (P.K.)

یکی از کلید های کاندید به انتخاب DBA و DBP . (ممکن است روی یک رابطه کلید های کاندید زیادی داشته باشیم) یعنی DBA و DBP مشخص می کنند که این کلید اصلی باید چی باشد . (مثلا شماره دانشجویی یا کد ملی)

ویژگی های کلید اصلی :

الف) کاندیدی که کمترین صفات را دارد کلید اصلی است (یعنی تا کاندید یک عضوی داریم ، دو عضوی را انتخاب نمی کنیم)

ب) در طول حیات رابطه (تا زمانی که با این رابطه کار می کنیم) ، صفت باید کمترین تغییر را داشته باشد . (مثلا صفت شماره دانشجویی ممکن است در چند سال آینده رقمی شود پس یعنی این صفت چون امکان تغییر در آن است نمی تواند کلید اصلی باشد .)

ج) اهمیت صفات -- > صفاتی که در DB دو ویژگی بالا را داشته باشد و نیز دارای اهمیت بیشتر باشد (مثلا در سیستم دانشگاهی ، شماره دانشجویی و کدملی هر دو دو ویژگی بالا را دارد ولی شماره دانشجویی اهمیت بیشتری دارد چون هر دانشجو با شماره آن شناخته می شود .)

4- کلید فرعی (Alternative Key) (A.K.)

کلید کاندیدی که اصلی نباشد .

5- کلید خارجی (Foreign Key) (F.K.)

تنها کلیدی است که کلید نیست یعنی خاصیت کلیدی ندارد و در واقع مجموعه صفاتی است که عضو تکراری هم میتواند داشته باشد

به این دلیل به آن کلید خارجی گفته می شود که ممکن است در یک رابطه عضو تکراری داشته باشد اما در یک رابطه دیگر کلید است

- مجموعه صفات A در رابطه R1 کلید خارجی است اگر رابطه R2 موجود باشد به طوری که A در رابطه R2 کلید کاندید(اصلی یا فرعی) باشد .
- عکس این رابطه برقرار نیست یعنی اگر A در R2 کلیدکاندید باشد به این معنی نیست که A در R1 کلید خارجی است .

6- کلید جزئی (Partial Key)

برای مشخص کردن **موجودیت های وابسته** استفاده می شود . (در اکثر موارد رابطه ما یک به چند است)

تعریف دیگر : اگر AB در یک رابطه کلید اصلی باشد به صورتیکه که A کلید خارجی باشد ، آنگاه B کلید جزئی است .

نکته -- > در همه موجودیت های وابسته ما کلید جزئی داریم .

ED (افراد تحت تکفل)

EID	PID	Relation	Name	Family
1	1	فرزند	رضا	شفیعی
1	2	فرزند	جواد	رضویان
2	1	پدر	رضا	کیانیان

کارمند

ID	name	family
1	اکبر	عبدی
2	.....	.....

- در این جا دو تا موجودیت داریم یکی کارمند و دیگری افراد تحت تکفل .
- میدانیم افراد تحت تکفل یک موجودیت وابسته است و وجود نمونه های آن وابسته به وجود نمونه های کارمندان است و اگر کارمندی نباشد ، افراد تحت تکفل آن نیز وجود ندارد و اگر یک کارمند اخراج شود کلیه اطلاعات مربوط به افراد تحت تکفل آن نیز پاک می شود .
- ED یک موجودیت وابسته(ضعیف) و کارمند موجودیت مستقل (قوی) است.
- ارتباط بین این دو موجودیت **یک به چند** است چون یک کارمند میتواند چندین نفر را تحت تکفل خود داشته باشد .

- EID کلید خارجی است چون میتواند تکراری باشد پس کلید کاندید – ابر کلید و کلید اصلی نیست.(یک نفر میتواند چندین نفر را تحت تکفل داشته باشد)
- PID کلید جزئی است چون میتواند تکرارير داشته باشد.
- **رابطه نرمال: همه مقادیر صفات خاصه اتمیک‌اند**

مثال : شماره دانشجویی هم ابر کلید است هم کلید کاندید و اگر DBA آن را انتخاب کند میتواند کلید اصلی نیز باشد .

نکته -- > هر ابر کلید تک عضوی حتما کلید کاندید است چون خاصیت کمینه بودن هم دارد

نکته -- > تهی دارای خاصیت کمینگی نیست چون میتواند بین نمونه ها تکرار شود .

نکته -- > هر رابطه حداقل یک ابر کلید و یک کلید اصلی و یک کلید کاندید را دارد.(در بدترین شرایط مجموعه تمامی صفات ابر کلید و کلید کاندید و کلید اصلی هستند) چون طبق تعریفی که گفتیم در مدل رابطه ای سطر تکراری نداریم پس در بدترین شرایط مجموعه تمامی صفات منحصر به فرد است .

کلید اصلی — دانشجو — خاصیت کلیدی ندارند — CS — درس

Std No	name	family	Avg

Std No	CID	grade
1	2	16
2	3	14

CID	name	Unit

هر دو با هم کلید کاندید هستند .

- شماره دانشجویی برای CS کلید خارجی است چون برای دانشجو کلید اصلی است .
- CID چون برای درس کلید اصلی است ولی برای CS کلید خارجی است .
- اگر یک کلید کاندید داشتیم آن کلید کاندید ، کلید اصلی است . (طریقه بدست آوردن کلید های فرعی -- > A.K = C.K - P.K)

قواعد جامعیت

تمامی قواعدی هستند که ما در سیستم پایگاهی در نظر میگیریم .

در حالت کلی به دو دسته تقسیم می شوند :

الف)قواعد جامعیتی که توسط خود کاربر تعیین می شوند (مثلا DA می گوید که من دوست دارم قاعده ای تنظیم کنم که هر دانشجو کمتر از سه روز برای خودش برنامه ای نچیند)

ب) قواعد جامعیتی که خود DBMS در نظر می گیرد و در همه DB ها این قواعد وجود دارند (مثلا کلید اصلی نباید عضو تکراری داشته باشد) **که به سه دسته زیر تقسیم می شود :**

1- **قواعد جامعیت درون رابطه‌ای:** هر رابطه به تنهایی باید صحیح تعریف شده باشد به عنوان مثال عضو تکراری نداشته باشد و کلیدهایش درست تعریف شده باشند .

2- **قواعد جامعیت موجودیتی:** هیچ یک از اجزای کلید اصلی نباید تهی باشد.
اگر ما مجموعه ای از صفاتمان کلید اصلی شدند هیچ جز از صفات نباید تهی باشد یعنی هیچ مقدار پذیر نیستند .

3- **قواعد جامعیت ارجاعی**: مقدار کلید خارجی یا NULL است (به شرطی که بخشی از کلید اصلی نباشد) یا در جدول اصلی (که در آنجا کلید اصلی به حساب می آید) مقدارش موجود است.

جلسه هشتم

جبرهای مختلف

- **جبر ریاضی:**
  - نوع داده‌ای: اعداد حقیقی
  - عملگرها: + و - و * و /
- **جبر منطقی: در واقع جبر بول دودیی است.**
  - نوع داده‌ای: مجموعه {True, False} و یا {1 ، 0 } <-- دامنه ی جبر منطقی
  - عملگرها: AND و OR و Not
- **جبر رابطه‌ای:**
  - نوع داده‌ای: فقط رابطه (خاصیت بسته بودن) : عملگر ها روی رابطه ها اعمال میشوند و نتیجه ی آن به ما یک رابطه جدید میدهد. بین هر دو کلید در رابطه اعمال میشود.

عملگرهای جبر رابطه‌ای

- **عملگرهای ساده :**

1 – گزینش Select ، 2- پرتو Project

- **عملگرهای مجموعه‌ای**

1- اجتماع

2- اشتراک

3- تفاضل

- **عملگرهای پیوند**
- **عملگرهای دیگر**

عملگرهای ساده: گزینش

- نشان: σ و select و restrict
- ورودی: یک رابطه
- شرط: عبارت شرطی
- خروجی: یک رابطه
- مثال:

$$\sigma_{name=\text{اکبر}}(S)$$

$$\sigma_{unit>2\,\hat{}\,type=\text{عمومی}}(C)$$

مثلا ما یک جدول برای دانشجویان داریم ، حال میخواهیم از بین این دانشجویان کسانی را که درس پایگاه داده را گرفته اند ، انتخاب کنیم در نتیجه عملگری که روی رابطه ی دانشجو اعمال میشود، Select است.

این عملگر روی سطر ها و داده های جدول کار میکند.

حاصل عملگر Select یک رابطه یا یک جدول دیگر است ولی تنها نسبت به مجموعه اصلی روی سطر ها فیلتر انجام شده است.

∂ (S) → Name = اکبر

S = (رابطه ی دانشجو (یعنی از بین همه ی دانشجویان

مفهوم : از بین دانشجویان آنهایی که اسمشان اکبر است را انتخاب کن.

∂ ($\prod$ (DB)) → جواب ندارد و خطا میدهد چون بعد از آنکه داخلی ترین رابطه اعمال شود دیگر ستونی برای معدل وجود ندارد که عملگر انتخاب روی سطر های آن انجام شود.

∂ ($\prod$ (DB) AVR > 17) → از بین دانشجویان درس دیتابیس ، کسانی را که معدلشان از 17 بیشتر است ، انتخاب کن

عملگرهای ساده: پرتو

- نشان: Π و project و project … over
- ورودی: یک رابطه
- انتخاب‌گر: نام ستون‌ها
- خروجی: یک رابطه (سطرهای تکراری حذف می‌شوند)
- مثال:

$$\Pi_{name}(S)$$
$$\Pi_{name,unit}(C)$$

روی ستون ها و یا همان Schema جدول پردازش انجام میدهد.

عملگر پرتو نه تنها ستون ها را فیلتر میکند بلکه ممکن است سطرها را نیز کاهش دهد. (سطر های تکراری حذف میشوند.)

$\prod$ name (S) → مفهوم : اسامی همه ی دانشجویان را به ما میدهد .

$\prod$ name , unit (C) → مفهوم : اسم و تعداد واحد درس ها را به ما میدهد.

$\prod$ city , phone (∂ (DB) AVG >= 17)) →

مفهوم : در ابتدا از بین دانشجویان پایگاه داده آنهایی که معدلشان بیشتر از 17 شده را پیدا میکند حال و حال روی این مجموعه ی جدید ، اسامی شهرها و تلفن های آنها حاصل نهایی این عبارت خواهد بود.

هر رابطه از مجوعه های body و header تشکیل شده و میتوان روی مجوعه ی body آن عملگرهای اجتماع و اشتراک و تفاضل را روی سطرهای جدول آن رابطه اعمال کنیم.

عملگرهای مجموعه‌ای: اجتماع

- نشان: U و Union
- ورودی: دو رابطه
- خروجی: یک رابطه
- مثال:

S -> مجموعه دانشجویان

P -> مجموعه اساتید

$$\Pi_{City}(S) \cup \Pi_{City}(P)$$

مفهوم مثال بالا : مجموعه ی شهر های اساتید و دانشجویان را به ما می دهد .

* بعد از آنکه اجتماع میشوند سطرهای تکراری حذف میشود.

عملگرهای مجموعه‌ای: اشتراک

- نشان: ∩ و Intersect
- ورودی: دو رابطه
- خروجی: یک رابطه
- مثال:

$$\Pi_{City}(S) \cap \Pi_{City}(P)$$

مفهوم مثال بالا : مجموعه ی شهر های مشترک اساتید و دانشجویان.

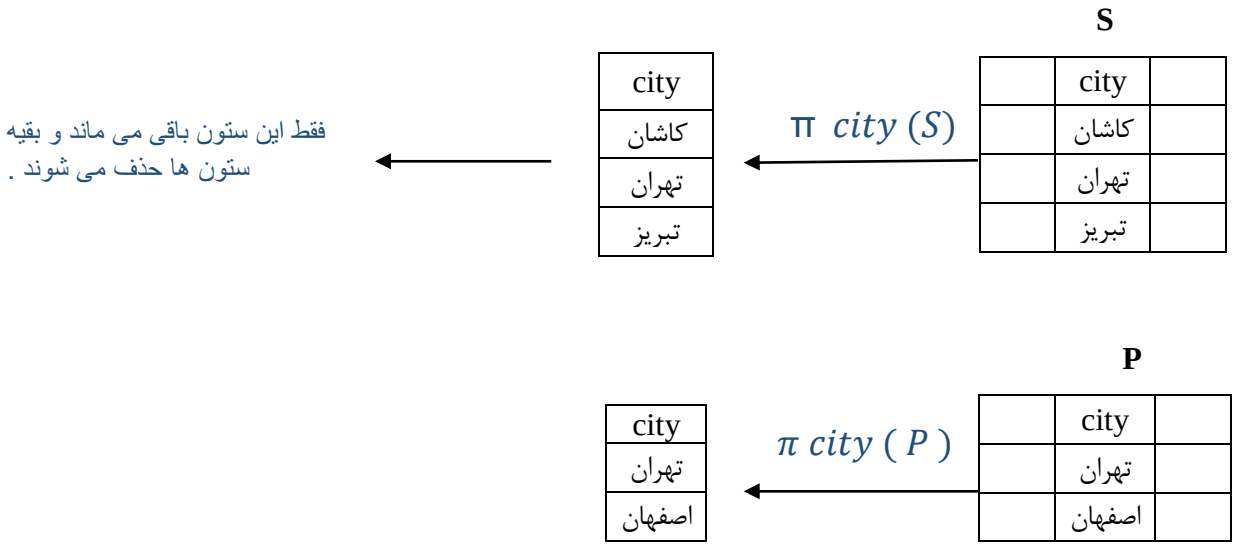
عملگرهای مجموعه‌ای: تفاضل

- نشان: − و Minus
- ورودی: دو رابطه
- خروجی: یک رابطه
- مثال:

$$\Pi_{City}(S) - \Pi_{City}(P)$$

مفهوم : مجموعه شهر دانشجویانی را به ما می دهد که استاد در آن نباشد .

مفهوم عملگر های − و ∩ و ∪ در مثال زیر :



$$\pi\ city\ (\ S\ ) - \pi\ city\ (\ P\ )\ \rightarrow$$

City
کاشان
تبریز

$$\pi\ city\ (\ P\ ) - \pi\ city\ (\ S\ )\ \rightarrow$$

City
اصفهان

$$\pi\ city\ (\ S\ ) \cap \pi\ city\ (\ P\ ) \rightarrow$$

city
تهران

$$\pi\ city\ (\ S\ ) \cup \pi\ city\ (\ P\ ) \rightarrow$$

City
کاشان
تهران
تبریز
اصفهان

ویژگی های عملگرهای مجموعه‌ای

- **ورودی‌ها باید همتا باشند یعنی هم تعداد ستون ها باید یکسان باشند و هم دامنه صفات**
- **اجتماع و اشتراک خاصیت جابجایی دارند**
- **اجتماع و اشتراک خاصیت شرکت‌پذیری دارند**
- **تفاضل خاصیت جابجایی ندارد**

** تمامی این عملگر هایی که تا اینجا گفتیم تمامی اطلاعات خود را از روی **یک جدول** بدست می آوردند.

مثلا برای پیدا کردن نام و نام خانوادگی دانشجویانی که در یک شهر زندگی میکنند ؛ میدانیم که نام و نام خانوادگی و شهر همگی جزو صفات دانشجو هستند و ما میتوانیم اطلاعات خود را از یک جدول یعنی جدول دانشجو بدست آوریم و نیازی نیست سراغ جدول های مختلف برویم .

عملگرهای پیوند

- **ضرب دکارتی**
- **پیوند طبیعی (Join)**
- **پیوند شرطی (Where)**
- **نیم پیوند (SimiJoin)**
- **فرا پیوند**

عملگرهای پیوند: ضرب دکارتی

- نشان: × و Times
- ورودی: دو رابطه
- خروجی: یک رابطه
- مثال:

$$S \times P$$

- حاصل این ضرب به ما رابطه ای را می دهد که تمام ستون های S و P را دارد.
- حتی اگر ستون مشترک هم داشته باشیم ، با توجه به اسم رابطه ستون ها را جدا میکنیم و مجموعه حاصل از ضرب دکارتی شامل همه ستون های دو رابطه است .
- به شرط آنکه تک تک رابطه ها سطر تکراری نداشته باشند ، در جدول حاصل ضرب دکارتی نیز سطر تکراری نداریم .
- در مجموعه حاصل از ضرب دکارتی هر سطر از مجموعه اول با تک تک سطر های مجموعه دوم ارتباط دارد یا به عبارت دیگر به ازای هر سطر رابطه اول تمام سطر های رابطه دوم در کنار آن قرار میگیرد .

برای درک بهتر به مثال زیر توجه کنید :

R1

A	B
A1	B1
A2	B2
A3	B3

R 2

C	D
C1	D1
C2	D2

R 1 × R 2

A	B	C	D
A1	B1	C1	D1
A1	B1	C2	D2
A2	B2	C1	D1
A2	B2	C2	D2
A3	B3	C1	D1
A3	B3	C2	D2

نکته مهم :

*** اگر R1 : n1 سطر و m1 ستون داشته باشد.**

*** اگر R2 : n2 سطر و m2 ستون داشته باشد.**

**** آنگاه R1 × R2 : m1 + m2 ستون و n1 × n2**

ویژگی های ضرب دکارتی :

- **زمان و فضای زیادی نیاز دارد**
- **تعداد ستون‌های حاصل: جمع تعداد ستون‌های ورودی‌ها**
- **تعداد سطرهای حاصل: ضرب تعداد سطرهای ورودی‌ها**
- **ستون‌های همنام با نام رابطه مشخص می‌شوند**
- **دارای خاصیت جابجایی و شرکت‌پذیری است! (چون هم سطر ها و هم ستون ها دارای خاصیت جابجایی و شرکت پذیری هستند)**

عملگرهای پیوند: پیوند طبیعی

- نشان: ∞ و Join و Join … over
- ورودی: دو رابطه
- خروجی: یک رابطه
- مثال:

$$S \infty SP$$

اگر اطلاعاتی میخواهیم که باید آنها را از چندین جدول بدست آوریم ، باید از این عملگر استفاده کنیم و یا به عبارت بهتر در دو صورت از عملگر join استفاده می کنیم :

1- **زمانیکه ستون هایی که میخواهیم از چندین جدول مختلف باشند .**

2- **ستون هایی که میخواهیم از یک جدول و شرطی که به آن نیاز داریم از جدول های دیگر باشد (مثلا اسامی دانشجویانی را میخواهیم که درس پایگاه داده 20 شده اند . میدانیم که اسامی دانشجویانی متعلق به جدول دانشجو و شرط 20 شدن درس مربوط به جدول CS است)**

$R1 \infty R2$ تمامی ستون های R1 و تمامی ستون های R2 را دارد ولی ستون های مشترک تنها یکبار نوشته میشوند و سطر هایی از R1 و R2 در کنار هم قرار می گیرند که در آن صفات یا ستون های مشترک ، مقدار یکسان داشته باشند .

سطر های $R1 \infty R2$:

بهترین حالت : $n1 \cap n2$

بدترین حالت : $n1 \times n2$

ستون های $R1 \infty R2$:

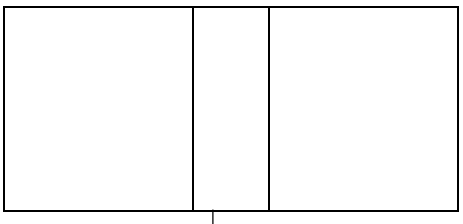
$$m1 \cup m2 \quad - \quad m1 \cap m2$$

- **90 درصد مواقع join روی کلید های خارجی انجام می شود . یعنی سطر هایی کنار هم قرار می گیرند که کلید خارجی آن مقدار یکسانی داشته باشند .**

مثال)

S join CS over sid

مفهوم : S را روی CS روی ستون sid ، join کن. (Sid همان ستون مشترک بین دو رابطه S و CS است .)



ستون های مشترک دو رابطه

معمولا رابطه ها را به صورت زیر نمایش می دهند .

نام رابطه (نام فیلد های رابطه)

S (SID , Sname , city)

ویژگی های عملگر پیوند طبیعی :

- **زمان و فضای کمتری نیاز دارد .**

چون بر خلاف ضرب دکارتی نیازی نیست که همه سطر های رابطه دوم جلوی سطر های رابطه اول قرار بگیرند پس زمان و فضای کمتری نیاز دارد . (در این عملگر تنها سطر هایی قرار می گیرند که در ستون های مشترک مقدار یکسان داشته باشند)

- **زمانیکه همه ستون های دو رابطه از اینن عملگر و اگر ستون های دو رابطه مشترک نباشند از ضرب دکارتی استفاده می شود .**

به عبارت خلاصه : اگر دو رابطه فیلد همنام نداشته باشند از ×

و اگر تمام فیلد ها مشترک باشند از ∩ استفاده می شود .

- **ستون های همنام یکبار نوشته می شوند .**
- **دارای خاصیت جابجایی و شرکت پذیری هستند**

چون هم سطر ها و هم ستون های آن میتوانند جا به جا شوند .

عملگرهای پیوند: پیوند شرطی

- نشان: X_θ و Times…where
- ورودی: دو رابطه
- شرط: عبارت شرطی
- خروجی: یک رابطه
- مثال:

$$S\ X_{Sage>Page}\ P$$

- **خروجی این عملگر رابطه جدیدی است که سطر هایی از R1 و R2 را به ما می دهد که شرط θ توی آنها برقرار باشد و همه ستون های R1 و R2 را به ما میدهد .**

چند نمونه مثال :

اساتید هم اتاقی را به ما میدهد

P1 × P2
P1.RoomNumber = P2.RoomNumber

چون هر استاد ممکن است هم در P1 و هم در P2 باشدچون هر شخص با خودش هم هم اتاقی است پس برای حذف این اسامی تکراری دو شرط دیگر میگذاریم

P1 × P2
P1 . RoomNumber = P2 . RoomNumber
و P1 . name <> P2
و P1 . family <> P2 . family

R1 × R2
B != C

همان جدول R1 × R2 است که در آنها تنها سطرهایی وجود دارد که در آن B != C است

R1

A	B
A1	2
A2	4
A3	1

R2

C	D
1	D1
5	D2

R1 × R2
B != C

A	B	C	D
A1	2	1	D1
A1	2	5	D1
A2	4	1	D1
A2	4	5	D2
A3	1	5	D2

عملگرهای پیوند: نیم پیوند

- نشان: ∝ و SemiJoin
- ورودی: دو رابطه
- خروجی: یک رابطه
- مثال:

$$P \propto SP$$

- خاصیت جابجایی ندارد (چون ستون های آن با هم فرق می کنند .)
- تعداد سطرها ممکن است از join کمتر باشد

- در این مثال S تولید کننده است و رابطه آن به صورت S (SID , Sname , City) است.
- P محصولات است و رابطه آن به صورت P (PID , Pname , Color , City) است.

SP رابطه بین محصول و تولید کننده است که یک رابطه ی جند به چند است (چون هر تولید کننده می تواند چندین محصول تولید کند و نیز هر محصول توسط چندین تولیید کننده میتواند تولید شود) و به صورت(SID , PID , QTY) SP تعریف می شود.

- این عملگر معادل است با π (R1 join R2)
- سطر های جدول حاصل از این عملگر شاید کمتر از سطر های حاصل از join باشد چون در اینجا π سطر های تکراری را حذف می کند .
- SP SemiJoin S ستون های SP را دور می ریزد و مشخصات و ستون های S را نگه می دارد .

جلسه نهم

رابطه زیر را در نظر بگیرید:

P

PID	Pname	Color	City
P1	پیچ	مشکی	-
P2	مس	نارنجی	کرمان
P3	زیتون	سیاه	گیلان

S

SID	Sname	City
S1	فن آوران	تهران
S2	تورین تن	تهران
S3	هوشمند گستر	اصفهان
S1	لنت	کاشان

SP

SID	PID	QTY
S1	P1	100
S1	P2	300
S2	P2	700
S3	P3	1000
S3	P1	1500

- SID در رابطه S کلید خارجی است چون میتواند تکرار شود .
- PID در رابطه P کلید خارجی است چون میتواند تکرار شود .
- (SID , PID) برای رابطه SP چون تکراری نمی شوند پس کلید اصلی است.
- مثلا در اینجا اگر بخواهیم تولید کنندگان شهر تهران یا محصولات قرمز رنگ را پیدا کنیم ، از یک جدول اطلاعات خود را بدست می آوریم .
- همچنین اگر بخواهیم تولید کنند هایی که محصولات بیشتر مساوی 1000 تولید می کنند را پیدا کنیم ، باید از اطلاعات چند جدول استفاده کنیم.که کوئری آن به صورت زیر است .

$$\pi_{SID}\ (\ \sigma_{QTY \geq 1000}\ )$$

- اگر *ID* تولید کننده هایی که بیشتر مساوی 1000 محصول را تولید میکنند را بخواهیم ، باید به صورت زیر عمل کنیم.

$$\pi_{Sname\ ,City}\ (\ \sigma_{QTY \geq 1000}\ \ (\ S \infty SP\ ))$$

- معمولا یک شکل منحصر به فرد برای نوشتن کوئری وجود ندارد و به راه های مختلف میتوان به یک جواب یکسان رسید به طور مثال برای حالت قبل میتوان کوئری دیگری به شکل زیر نوشت :

$$\pi_{Sname\ ,City}\ (\ S\ \propto\ \sigma_{QTY \geq 1000}\ (\ SP\ ))$$

- $\propto$: در اینجا join بین رابطه اول و دوم را انجام می دهد و سپس ستون های رابطه دوم را حذف می کند .

فرا پیوند outer join (join خارجی) :

- Left Outer Join :L-O-JOIN
- Right Outer Join :R-O-JOIN
- Full Outer Join :F-O-JOIN
- مثال:

$$S\ L\_O\_JOIN\ SP$$

- L.O.J از سمت راست join hk[hl ldnin
- Full outer join هم شامل L.O.J و هم شامل R.O.J است.
- همانطور که قبلا گفتیم در join طبیعی سطر هایی کنار هم قرار می گیرند که در ستون های مشترک مقدار یکسان داشته باشند پس هم ستون های مشترک رابطه اول و هم ستون های مشترک رابطه دوم باید مقدار داشته باشند و آن سطر هایی که در ستون های مشترک مقدار null داشته باشند ، در نظر گرفته نمی شوند.

مثال:

R1

A	B	C
A1	B1	C1
A2	Null	C2
A3	B3	C3

R2

D	B	E
D1	B1	E1
D2	B2	E2
D3	null	E3

$$R1 \infty R2$$

A	B	C	D	E
A1	B1	C1	D1	E1

- در پیوند طبیعی همه ستون ها را می نویسیم و ستون هایی که مشترک هستند را تنها یکبار مینویسیم.
- در پیوند طبیعی کلیه سطر هایی که دارای ستونی هستند که مقدار null دارند از بین می روند و در نظر گرفته نمی شوند .
- در دو رابطه R1 و R2 ستون B مشترک است پس طبق آنچه قبلا گفتیم در $R1 \infty R2$ سطر هایی کنار هم قرار می گیرند که در ستون مشترک B مقدار یکسان داشته باشند .
- پس در پیوند طبیعی دو نوع سطر داریم :

1 – سطر های پیوند شدنی - > سطر هایی که در ستون های مشترک مقدار دارند و مقدار آنها یکسان است.

2 – سطر های پیوند نشدنی - > سطر هایی که در ستون مشترک مقدار null دارند یا فاقد مقدار یکسان هستند .

Left Outer Join :

در این نوع پیوند نه تنها سطر های پیوند شدنی دو رابطه را در کنار هم قرار می دهیم بلکه سطر های پیوند نشدنی رابطه سمت چپ را هم در نظر می گیریم. (پس در این حالت خاصیت جا به جایی نداریم یعنی R1 ∞ R2 با R2 ∞ R1 تفاوت دارد)

R1 L.O.J R2

A	B	C	D	E
A1	B1	C1	D1	E1
A2	null	C2	Null	Null
A3	B3	C3	Null	Null

- سطر های پیوند نشدنی رابطه R1 چون با هیچ یک از سطر های رابطه R2 پیوند نمی خورد پس مقدار این سطر ها در ستون های R2 ، null در نظر گرفته می شود.
- پس همه سطر های R1 را در رابطه پیوند از چپ داریم و اگر معادل داشت ، معادل را مینویسیم در غیر این صورت null می گذاریم.

Right Outer Join :

در این نوع پیوند نه تنها سطر های پیوند شدنی دو رابطه را در کنار هم قرار می دهیم بلکه سطر های پیوند نشدنی رابطه سمت راست را هم در نظر می گیریم. (پس در این حالت خاصیت جا به جایی نداریم یعنی $R1 \infty R2$ با $R2 \infty R1$ تفاوت دارد)

R1 R.O.J R2

A	B	C	D	E
A1	B1	C1	D1	E1
Null	B2	Null	D2	E2
null	Null	null	D3	E3

- سطر های پیوند نشدنی رابطه R2 چون با هیچ یک از سطر های رابطه R1 پیوند نمی خورد پس مقدار این سطر ها در ستون های R1 ، null در نظر گرفته می شود.

Full Outer join :

$$F.O.J = L.O.J \cup R.O.J$$

مثال :

فرض کنید C جدول دروس و G جدول گروه آموزشی باشد .

جدول C یک ستون به نام GID دارد که نشان می دهد که درس توسط چه گروه آموزشی ارائه شده است .(مثلا درس پایگاه داده در گروه علوم کامپیوتر ارائه می شود)

ممکن است GID درس C ، null باشد به این معنا که این درس توسط هیچ گروهی ارائه نشده است.

اگر اسامی تمامی دروس به همراه گروه آموزشی که آن دروس را ارائه داده اند بخواهیم ، باید از Outer Join استفاده کنیم .

$$\pi_{Gname,\ Cname}$$

وقتی ما میگوییم لیست تمامی دروس را میخواهیم یعنی میخواهیم همه دروس را لیست کنیم چه آنهایی که توسط گروه آموزشی ارائه شده و چه ارائه نشده پس از پیوند خارجی باید استفاده کنیم .

اما اگر از پیوند طبیعی بین C و G بخواهیم استفاده کنیم ، تعدادی از اطلاعات یعنی سطر های پیوند نشدنی از بین می روند .(به عبارت دیگر لیست درس هایی که توسط گروه آموزشی ارائه نشده اند از بین می روند)

در اینجا برای دسترسی به لیست کامل دروس بهتر است از L . O . J استفاده کنیم و مطمئن هستیم که تمامی درس ها لیست می شود چون هم سطر های پیوند شدنی و هم درس های پیوند نشدنی را خواهیم داشت.

عملگر های دیگر :

- **عملگر جایگذاری :**

برای ذخیره موقت نتایج بکار می رود . مثلا ما میتوانیم نتایجی را که بدست آورده ایم در یک رابطه جدید با یک اسم جدید ذخیره کنیم و سپس روی این رابطه جدید باز هم پردازش انجام دهیم .

معمولا اگر تعداد پرس و جوها زیاد باشد ، یعنی احتیاج به چندین عملگر مختلف داشته باشیم تا به جواب برسیم ، بهتر است پرس و جوی بزرگ را به پرس و جو های کوچکتر بشکنیم و مرحله به مرحله پرس و جو ها را انجام و عملگر ها را اعمال و نتایج آن را در متغیر های موقت ذخیره کنیم .

مثلا در جدول تولید کننده و محصولات ، نام تولید کننده و نام محصول را داریم حال می خواهیم ببینیم چه تولید کنندگانی اهل تهران هستند و محصول قرمز رنگ تولید می کنند ؟؟

همانطور که میبینیم داریم چندین شرط را روی چندین جدول بررسی می کنیم و در آخر join نیز داریم .

$$\text{Red} <- \sigma_{color = \text{قرمز}} (P)$$

$$\text{Tehran} <- \sigma_{city = \text{تهران}} (S)$$

$$T <- \text{Tehran} \infty \text{Red}$$

$$\text{Answer} <- \pi_{\text{Sname , Pname}} (T)$$

- **عملگر Rename :**

این عملگر برای تغییر نام ستون است .

$$\pi_{\text{نام}} (\sigma_{\text{AVG} \geq 17} (S \quad \text{rename} \quad \text{Sname} \quad \text{to} \quad \text{نام}))$$

یعنی ابتدا در جدول S ستون Sname را به نام تغییر داده و سپس نام دانشجویانی که معدلشان بالاتر از 17 است را به ما می دهد .

- **عملگر ρ :**

برای تغییر نام رابطه یا جدول است .

این عملگر کاربرد های زیادی دارد مثلا بخواهیم از جدول خود یک کپی بگیریم - > $\rho_T (S)$ یعنی یک رابطه S و یک رابطه T داریم که هر دو اطلاعات دانشجو را به ما نشان می دهد .

- **عملگر SemiMinus :**

با Mins و ∝ پیاده سازی می شود .

$$A \quad \textbf{SemiMinus} \quad B = A - (A \propto B)$$

- **می دانیم – یک عملگر مجموعه ای است پس طبق ویژگی های آن ، A و B باید همتا باشند (یعنی هم تعداد ستون های آن با هم برابر باشند و هم ستون ها نظیر به نظیر هم دامنه باشند)**
- $A \propto B$ **از جنس** A **است پس** $A \quad SemiMinus \quad B$ **خود به خود همتا می شود .**
- $A \propto B$ **سطر های پیوند شدنی A را نگه می دارد .**
- $A \quad SemiMinus \quad B$ **سطر هایی از** A **که پیوند نشدنی با** B **هستند را نگه می دارند .**

- **عملگر Extend :**

از این عملگر زمانی استفاده می شود که بخواهیم یک رابطه را توسعه دهیم و در واقع برای اضافه کردن یک ستون به رابطه و نیز برای محاسبه صفات مشتق شده (صفاتی که از روی داده ها بدست می آیند و نیازی به ذخیره سازی ندارند) بکار می رود

مثلا در رابطه محصول و تولید کننده میخواهیم درآمد هر محصول که توسط هر تولید کننده بدست می آیند را بدست آوریم ؟

جدول مربوط به محصولات علاوه بر ستون هایی که قبلا گفتیم ، یک ستون Price برابر با قیمت آن محصول است ، نیز دارد.

در جدول مربوط به تولید کننده نیز ستون QTY را داریم که بیان میکند از هر نوع محصول چه تعدادی تولید شده است .

چون این دو ستون مربوط به دو جدول متفاوت است پس ابتدا باید این دو جدول را با هم join کنیم و سپس در آمد را از آن محاسبه کنیم.

$$All <- SP \infty P$$

رابطه *All* همه ستون های *SP* و *P* را دارد .

حال برای رسیدن به درآمد باید Price * QTY کنیم که برای اینکار کافی است ،

$$T <- Extend\ All\ Add\ QTY * Price\ As\ \text{در آمد}$$

و به این صورت ستون در آمد به جدول اضافه می شود .

حال اگر اسم تولید کنندگان را خواستیم باید به صورت زیر عمل کنیم . (اسم تولید کننده در رابطه S است)

$$\pi_{SID,\ Pname,\ \text{در آمد}}(T)$$

$$T2 <- \pi_{SID,\ Pname,\ \text{درآمد}}(T)$$

$$\pi_{SID,\ Pname,\ \text{درآمد}}(T2 \infty S)$$

جلسه دهم :

- **عملگر تقسیم ÷ :**

هر جا بحث **همه** پیش می آید از این عملگر استفاده می شود (هم عبارت **همه** در آن به کار رفته و هم چندین **شرط** دارد)

مثلا **همه** دانشجویانی که **همه** درس های عمومی را گذراندند .

سوال : می خواهیم نام و شهر همه تولید کنندگانی که همه محصولات قرمز رنگ را تولید می کنند ، پیدا کنیم .

SP (رابطه R1)

SID	PID	Qty
S1	P4	100
S2	P1	300
.	.	.
.	.	.
S1	P7	500
S2	P4	500
.	.	.
.	.	.
S1	P2	300

Red (رابطه R2)

PID
P4
P2
P7

S

SID	Sname	city

R1 ÷ R2 (رابطه R3)

SID	Qty
S1	100
S1	300
S1	500

- R3 ستون هایی از R1 را دارد که در R2 وجود ندارد .
- R3 سطر هایی از R1 را دارد که حتما **همه** مقادیر R2 در مقابل آن سطر از R1 وجود داشته باشد .

جواب سوال -- >

ابتدا باید ID تولید کنندگانی که این محصولات قرمز رنگ را تولید می کنند پیدا کنیم و سپس با داشتن آن طبق جدول S نام و شهر آنها را استخراج کنیم .

مراحل انجام کار به صورت زیر است :

$$Red <-- \pi(\partial_{color = "Red"}(P))$$

$$R1 <-- \pi_{SID,\ PID}(SP)$$

$$R3 <-- R1 \div Red$$

$$R4 <-- R3 \infty S$$

$$Answer <-- \pi_{Sname,city}(R4)$$

- **عملگر خلاصه (Summerize):**

این عملگر یک رابطه می گیرد و یک رابطه تحویل می دهد و در واقع عملگر **دسته بندی** است .یعنی بر اساس مقادیر یک یا چند ستون جدول را دسته بندی می کند .

کاری که این عملگر انجام میدهد :

1- دسته بندی انجام میدهد و هر دسته را به یک سطر تبدیل می کند

به طور مثال با توجه به مثال قبل میتوان دسته بندی بر اساس ستون ها را به شکل زیر نشان داد .

SP

SID	PID	Qty
S1	P2	100
S1	P4	300
S1	P7	500
S2	P1	300
S2	P4	500

2- ستون هایی را که توسط توابع جمعی بدست می آیند را اضافه میکنند .

(توابع جمعی در واقع همان توابعی هستند که در عملگر Extend بدست می آیند است که یک سری محاسبات روی ستون ها انجام می دهد و سپس ستون جدید را به ما می دهد)

(توابع جمعی شامل : max , min , count , sum)

SID	minQty	maxQty	sumQty	count
S1	100	500	900	3
S2	300	500	800	2

Summerize SP over SID (نام یک یا چند ستون)

Add sumQty as S

Add count as C

SID	minQty	maxQty	S	C
S1	100	500	900	3
S2	300	500	800	2

مثال) در رابطه دانشجو – درس – ارتباط بین درس و دانشجو :

داریم : CS (SID , CID , grade)

میخواهیم ببینیم هر دانشجو چه تعداد درس گرفته است . پس باید در جدول مربوط به CS ، دسته بندی بر اساس SID (شماره دانشجویی) یکسان انجام شود و سپس تابع جمعی count برای آن محاسبه شود .

SID	CID	grade
S1	C1	
S1	C2	
S1	C3	
S2	C2	
S3	C7	
S3	C4	

C

SID	Count(*)
S1	3
S2	1
S3	2

Summarize CS

Add count (*) as C

$\pi_{sSname,Sfamily}\ (R \infty S)$

اگر نام و فامیل دانشجو را بخواهیم

- برای ستون هایی که nullable نباشند ،

count (*) = count (ستون)

- در غیر این صورت این دو مورد با هم تفاوت دارند .
  Count (*) سطر هایی که nullable باشند را نیز نشان میدهد

عملگرهای کامل

- **مجموعه کاملی از عملگر های منطقی { AND , OR , NOT } است و سایر عملگر ها را می توان به کمک آنها پیاده سازی کرد مانند NAND و NOR**
- **البته { NAND } و { NOR } به تنهایی خود مجموعه کاملی از عملگر های منطقی هستند چون می توان به کمک آنها مدار طراحی کرد**
- **برای عملگر های مجموعه ای نیز به همین صورت است چون با داشتن مجموعه کاملی از عملگر ها می توان سایر عملگر ها را پیاده سازی کرد .**
- **این مجموعه کامل شامل {− و ← و ∪ و × و π و ∂ } هستند البته ما علاوه بر اینها از عملگر های دیگری مانند { ρ و X_θ و ∞ و ÷ و ∩ و ∝ } استفاده می کنیم تا راحت تر بتوانیم به سوالات پاسخ دهیم .**

مثال : $A \cap B = A - (A - B)$

خواص عملگرها

- **∞ و ∩ و ∪ و × خاصیت شرکت پذیری و جابه جایی دارند . یعنی :**

$$A \times B = B \times A$$
$$(A \times B) \times C = A \times (B \times C)$$

همچنین برای ∞ و ∩ و ∪ نیز به همین صورت است

- **خاصیت توضیع پذیری σ روی − و ∩ و ∪**

$$\sigma\,(A \cup B) = \sigma(A) \cup \sigma(B)$$

همچنین برای − و ∩ نیز به همین صورت است

- **خاصیت توزیع پذیری π روی ∞ و ∪**

$$\pi\,(A \cup B) = \pi(A) \cup \pi(B)$$

همچنین برای ∞ نیز به همین صورت است

- $\sigma(A) \infty B = \sigma(A \infty B)$
- $\sigma(\pi(A)) = \pi(\sigma(A))$

برای پاسخ دادن بهینه به سوالات بهتر است از عملگر ها طبق قواعد زیر استفاده کنیم :

- **گزینش هرچه زودتر**
- **عملگر پرتو زودتر**

عملگر project را تا جاییکه ممکن است عقب می آوریم و این باعث می شود که ستون هایی را که ممکن است بعد ها به آن احتیاج پیدا کنیم را دیر تر حذف کنیم و در نتیجه اینگونه اطلاعات ما دقیق تر خواهد شد .

- **تبدیل شرطهای ترکیبی به شرطهای متوالی**

مثال :

$\sigma(A) = \sigma(\sigma(A))$

$\theta \rightarrow X = x1$ و $Y = y$ $\quad X = x1 \quad Y = y1$

فرض کنید در یک رابطه 4000 سطر وجود داشته باشد و ما طبق طرف چپ میخواهیم یک شرط بزرگ را روی این 4000 سطر بررسی کنیم که این کار بسیار دشوار است اما در طرف راست بین 4000 سطر ، شرط کوچکی را بررسی و سپس از بین سطر های پیدا شده ، شرط دیگر را بررسی میکنیم که این کار آسان تر است .

- **خاصیت جابجایی و شرکت‌پذیری از لحاظ پیچیدگی فضایی و زمانی**

هر چقدر تعداد سطر ها کمتر شود ، کار ما راحت تر است پس مثلا اگر چند عملگر ∞ داشته باشیم باید نگاه کنیم و ببینیم که کدام ∞ زودتر انجام شود منجر به کمتر شدن تعداد سطر ها می شود .

یا مثلا در مثال زیر اگر A دارای 2 سطر و B دارای 1000 سطر و C دارای 2000 سطر باشد مشاهده میکنیم که انجام عملیان سمت راست آسان تر و پیچیدگی زمانی و فضایی کمتری دارد

$$A \times (B \times C) = (A \times B) \times C$$

کلید کاندید در عملگرها

- **در $Extend(A)$ و $\sigma(A)$ هر کلید رابطه A (K_A) ، یک کلید برای رابطه نتیجه است .**

به عبارت دیگر در این دو عملگر هر کلید برای رابطه A یک کلید برای رابطه نتیجه است .

- **در π کلید بستگی به ستون های انتخابی دارد .**

اگر کلید در π حذف شود رابطه جدید حتما یک کلید دارد و در بدترین حالت ، مجموعه تمامی ستون های انتخابی کلید می شود.

- **در $A \times B$ هر ترکیب K_B و K_A ، یک کلید است .**

فرض کنید { x1 , x2 ,x3 } کلید A و { y1 , y2 , y3 } کلید B باشند در این صورت کلید های $A \times B$ به صورت زیر خواهد بود

{ (x1 , y1) , (x2 , y2) , ... }

- **در ∪ در بدترین حالت ترکیب تمام صفات کلید کاندید است .**

اگر K_A در A منحصر به فرد و K_B در B منحصر به فرد باشد ، در $A \cup B$ لزوما اجتماع این کلید ها منحصر به فرد نیست .

- **در $A \cap B$ هر کلید رابطه A یا کلید رابطه B ، یک کلید است .**

- **در A – B هر کلید رابطه A ، یک کلید است .**

چون از A یک سری سطر ها را حذف کردیم اما کلید رابطه A هنوز منحصر به فرد بودن خود را از دست نداده پس میتواند کلید A-B نیز باشد .

- **در $A \infty B$ اگر صفت مشترک در A کلید باشد ، هر کلید رابطه B ، یک کلید است .**

همانطور که میدانیم این عملگر روی صفات مشترک انجام می شود پس مثلا در رابطه S و SP ، ستون SID مشترک است

$S \infty SP$

در اینجا SID کلید S و (PID , SID) کلید $S \infty SP$ خواهد بود .

Sname	SID	PID	Qty

- **در Summerize صفت جلو by یا over ، یک کلید است .**

یا به عبارت دیگر مجموعه ستون هایی که در جلوی over باشند ، کلید هستند.

امتحان میان ترم پایگاه داده

جلسه یازدهم :

آشنایی با یک زبان رابطه ای : SQL

زبانهای رابطه‌ای

- **SQUARE**
- **SEQUEL**
- **SQL**
- **QUEL**
- **QBE**
- **DATALOG**

امکانات مهم زبان SQL

- **دستورات تعریف داده‌ها**
- **دستورات مجازشماری**
- **دستورات پردازش داده‌ها**
- **دستورات پردازش داده‌ها به طور ادغام‌شدنی**
- **دستورات نوشتن ماژول و رویه**
- **دستورات کنترل جامعیت**
- **دستورات کنترل (جزو دستورات DCL) و تعریف تراکنشها**

یعنی تراکنش چه زمانی connect و چه زمانی rollback می شود را مشخص میکند .

انواع داده تعریف شده در SQL :

داده های عددی:

- **Tinyint** (عدد صحیح خیلی کوچک) : عدد صحیح **1 بایتی** مثبت (0 تا 255) – > فقط شامل اعداد مثبت
- **smallint** : عدد صحیح **2 بایتی** (32768 – تا 32767) - > فقط شامل اعداد مثبت نیست ، اعداد منفی را نیز شامل می شود.
- **int** : عدد صحیح **4 بایتی** (2147483648 – تا 2147483647) $\approx 2 \times 10^9$
- **bigint** : عدد صحیح **8 بایتی** - > هم مثبت و هم منفی می تواند باشد .
- **decimal** : اعداد **حقیقی** از $1 - 10^{38}$ – تا $1 - 10^{38}$ - > تعریف دسیمال برای اعداد اعشاری یعنی چند رقم عدد صحیح و چند رقم عدد اعشار دارد .

نمایش عدد اعشاری -> Decimal(5 , 2)

2 رقم اعشار و 5 رقم عدد صحیح دارد .

نمایش عدد صحیح -> Decimal(5 , 0)

0 رقم اعشار و 5 رقم عدد صحیح دارد.

- **real** : اعداد اعشار **4 بایتی** با دقت 7 رقم - > تنها برای نمایش اعداد اعشاری بکار می رود.
- **float : اعداد اعشار 8 بایتی** با دقت 15 رقم - > تنها برای نمایش اعداد اعشار بکار می رود.

داده های رشته ای :

- **char** : رشته تا حداکثر 8000 کاراکتر : ذخیره **با طول ثابت**

مثل char(50) رشته هایی با 50 کاراکتر را نگه می دارد که اگر رشته ما دارای کمتر از 50 کاراکتر باشد ، بقیه جایگاه های خالی ، space در نظر گرفته می شود . اما اگر رشته بیشتر از 50 کاراکتر داشته باشد ، فقط 50 کاراکتر آن نگهداری می شود و بقیه از بین می رود .

- **varchar** : رشته تا حداکثر 8000 کاراکتر : ذخیره با **طول متغیر**

مثلا varchar(100) رشته هایی با 100 کاراکتر را نگهداری می کند . که اگر رشته کمتر از این تعداد کاراکتر داشت ، به جای فضاهای خالی ، space قرار نمی دهد و اگر بیشتر از 100 کاراکتر داشت ، 100 کاراکتر آن نگهداری می شود و مابقی دور ریخته می شود .

** برای تعریف رشته هایی که به زبانهای دیگر هستند ، از دو نوع داده ا زیر استفاده می شود که هر کاراکتر را در 2 بایت ذخیره می کنند :

- **nchar** : رشته تا حداکثر 4000 کاراکتر : ذخیره با **طول ثابت و به صورت Unicode**
- **nvarchar** : رشته تا حداکثر 4000 کاراکتر : ذخیره با **طول متغیر و به صورت Unicode**

Char(max) - > یعنی در هر نسخه ای که هستی ، از حداکثر فضا برای ذخیره سازی استفاده کن .

انواع داده‌ای برای ذخیره سازی زمان:

- **TIME**: **hhmmss**

اطلاعات مربوط به ساعت و دقیقه و ثانیه را در خود نگهداری می کند .

- **DATE**: yyyymmdd

اطلاعات مربوط به سال و ماه و روز را در خود نگهداری می کند.

- **TIMESTAMP** : **yyyymmddhhmmssnnnnnn**

مهر زمانی است که کلیه اطلاعات مربوط به زمان دقیق را نگهداری می کند .(شامل: سال – ماه – روز – ساعت – دقیقه – ثانیه – میکروثانیه)

سایر انواع داده ای :

- نوعی برای نگهداری money
- نوعی برای ذخیره عکس (image) که به صورت binary خیره میکند .
- نوعی برای ذخیره متن (Text)
- نوعی برای ذخیره رشته ها و متن های خیلی بزرگ (nText)

** با کمک این متغر ها می توان ستون هاو سطر های جدول ها را مقدار دهی کرد .

دستورات DDL :

1- **create** : برای ساختن
2- **Alter** : برای ویرایش
3- **Drop** : برای حذف

دستور ایجاد جدول :

وقتی می خواهیم جدول بسازیم باید اول تعیین کنیم که چه اسمی دارد و چه ستون هایی دارد و در حالت کلی به صورت زیر عمل کنیم :

```
CREATE    TABLE    نام جدول (
نام ستون 1    نوع ستون 1    محدودیت های ستون 1 ,
نام ستون 2    نوع ستون 2    محدودیت های ستون 2
.....
);
```

مثال :

```
CREATE TABLE EMPLOYEE(
    EmpID       Integer       PRIMARY KEY,
    EmpName     Char(25)      NOT NULL
    );
```

- ستونها با نماد , (کاما) از هم جدا می شوند .
- در این مثال NOT NULL هیچ مقدار ناپذیر بودن ستون 2 را تضمین می کند .
- SQL حساس به حروف کوچک و بزرگ نیست پس دستورات به هر طریقی که نوشته شوند ، پذیرفته می شوند ولی اصولا کلمات کلیدی با حروف بزرگ نوشته می شوند .

(دقت کنید که این حساسیت فقط برای **دستورات** وجود ندارد.)

- SQL انعطاف پذیری بالایی دارد مثلا برای آن int یا integer و create یا CRT تفاوتی ندارد.

دستور تغیر (ویرایش) جدول :

ساختار کلی ویرایش به صورت زیر است :

```
Alter     TABLE     نام جدول (
Add    نام ستون    نوع ستون    محدودیت ستون
.
.
.
Drop    column    نام ستون
);
```

مثال:

```
Alter TABLE EMPLOYEE(
   Add      salary      Integer
   drop column EmpName
);
```

- برای اضافه کردن ستون ها از دستور Add استفاده می شود .
- در نظر گرفتن محدودیت برای ستون ها امری اختیاری است.
- اگر خواستیم یک ستون از جدول را به کلی تغیر دهیم از دستور زیر استفاده میکنیم (به شرط آنکه آن ستون حتما جزو ستون های جدول باشد . در غیر این صورت باید ستون اضافه شود)

Modify   نام ستون   نوع ستون   محدودیت ستون

دستور حذف جدول:

حذف جدول با Drop انجام میشود که جزو دستورات DDL است ولی برای حذف سطر از جدول از delete استفاده می شود که جزو دستورات DML است .

ساختار کلی :

DROP TABLE نام جدول

مثال:

DROP TABLE Employee

دستورات DML :

1- **Select : انتخاب یک سطر**
2- **Insert : اضافه کردن سطر به جدول**
3- **Update : ویرایش سط های جدول**
4- **Delete : حذف سطر از جدول**

دستور Select :

ساختار کلی :

Select نام ستون 1 و نام ستون 2 و.....

From نام جدول

Where شرط

- معادل همان عملگر project است با این تفاوت که project سطر های تکراری را نیز حذف میکرد اما اینجا اینگونه نیست و اگر بخواهیم در اینجا سطر های تکراری را نیز حذف کنیم باید از دستور زیر استفاده کنیم.

Select distinct نام ستون 1 و .....

From نام جدول

Where شرط

- شرط هایی که جلو where قرار میگیرد می توانند ترکیبی نیز باشند .

مثال :

```
SELECT [ALL | DISTINCT] item(s)-list
FROM table(s)-name
[WHERE condition(s)]
[GROUP BY column(s)]
[HAVING conditions(s)]
```

- در اینجا سه دستور اخر شرط های ترکیبی هستند

مثال:

Select Distinct name from S

Where (city = ‘ تهران ‘ OR AVG > 17) AND (.....)

- این مثال اسامی منحصر به فرد دانشجویانی را به ما می دهد که شرط ها برای آن برقرار باشد.

شرط ها :

1- =

2- <> یا =!

3- <

4- >

5- >=

6- Like / is null / is not null / All / any / In / .....

فصل SQL :

* SQL به دو صورت مستقل و ادغام شدنی به کار می رود .
* SQL در ابتدا یک زبان داده ای (DSL) بود اما در حال حاضر لازم نیست SQL را با زبان دیگری به نام زبان میزبان ترکیب کنیم تا برنامه کاربردی کاملی ایجاد کنیم .

تعدادی از زبان های رابطه ای عبارت است از : QBE - QUEL - SQL - SEQUEL - SQUARE

- دستورات تعریف داده ها ← امکان تعریف داده ها را فراهم می کند (جزو دستورات DDL است)
- دستورات مجازشماری ← قواعد جامعیت را روی داده ها کنترل می کند یا به عبارت بهتر دامنه صفات را کنترل می کند. (جزو دستورات DCL است)
- دستورات پردازش داده ها ← ماشه انتخاب و یا حذف سطرهایی از جدول .
- دستورات پردازش داده ها به طور ادغام شدنی
- دستورات نوشتن ماژول و رویه
- دستورات کنترل جامعیت ← دستوراتی برای اینکه قواعد جامعیت را روی کال داده ها اعمال کنیم (فراتر از مجازشماری است)
  - مثلاً هر دانشجو بیشتر از 20 واحد نمی تواند بردارد . (این دیگر مربوط به دامنه صفات نیست) یا مجموع واحدهایی که دانشجویی تواند بردارد.
- دستورات کنترل تراکنش ها ← اینکه تراکنش چه زمانی connect ، چه زمانی rollback می شود را مشخص می کند .

انواع داده تعریف شده در SQL :

tinyint : عدد صحیح 1 بایتی مثبت (0 تا 255) ← عدد صحیح کوچکتر از int است که فقط شامل اعداد مثبت می شود .

Smallint : عدد صحیح 2 بایتی (32768 - تا 32767) ← فقط شامل اعداد مثبت نیست ، اعداد منفی را نیز شامل می شود .

int : عدد صحیح 4 بایتی (2×10^9 - تا 2×10^9)

big int : عدد صحیح 8 بایتی ← هم مثبت و هم منفی می تواند باشد .

decimal : اعداد حقیقی از $-10^{38}+1$ تا $10^{38}-1$ ← برای اعداد اعشاری بکار می رود و در آن مشخص می کند که چند رقم عدد صحیح و چند رقم عدد اعشاری داریم . (برای اعداد دهدهی)

decimal (5 , 2) ⇐ عدد اعشاری
- 2 رقم اعشار داریم
- 5 رقم عدد صحیح داریم

decimal (5 , 0) ⇐ عدد صحیح
- 0 رقم اعشار دارد
- 5 رقم عدد صحیح داریم

(تمامی اعداد اعشاری پیش بکاری روند)

real : اعداد اعشار 4 بایتی با دقت 7 رقم .

float : اعداد اعشار 8 بایتی با دقت 15 رقم

char : رشته تا حداکثر 8000 کاراکتر : ذخیره با طول ثابت

- مثلاً char(50) رشته هایی با 50 کاراکتر را نگهداری می کند که اگر رشته ما دارای کمتر از 50 کاراکتر باشد ، بقیه ی جایگاه ها ی خالی ، space در نظر گرفته می شود . اما اگر تعداد کاراکترها بیشتر از 50 کاراکتر باشد ، فقط 50 تای آنها را نگه می دارد .

varchar : رشته تا حداکثر 8000 کاراکتر : ذخیره با طول متغیر

- مثلاً varchar(100) رشته هایی با 100 کاراکتر را نگه می دارد که اگر رشته کمتر از این تعداد کاراکتر داشت می گذارد . و اگر بیشتر از 100 کاراکتر داشت ، 100 کاراکتر آن مقداری گیرد و مابقی دور ریخته می شود .

nchar : رشته تا حداکثر 4000 کاراکتر : ذخیره با طول ثابت و به صورت unicode

nvarchar : رشته تا حداکثر 4000 کاراکتر : ذخیره با طول متغیر و به صورت unicode

char(max) = حداکثر فضای ذخیره سازی

برای تعریف رشته هایی که به زبان دیگر هستند بکار می روند.
مثلاً متغیر هایی که می توانند رشته های زبان فارسی را بگیرند.

انواع داده ای برای ذخیره سازی زمان :

Time : hh mm ss

اطلاعات مربوط به ساعت و دقیقه و ثانیه را در خود نگه می دارد.

Date : yyyy mm dd

اطلاعات مربوط به سال و ماه و روز را در خود نگه می دارد.

Time Stamp : yyyy mm dd hh mm ss nnnnnn

همه زمانی است که کلیه اطلاعات مربوط به زمان دقیق را دارد (شامل : سال - ماه - روز - ساعت - دقیقه - ثانیه - میکرو ثانیه)

سایر انواع داده ای :

- نوعی برای نگهداری money
- نوعی برای ذخیره عکس (Image) که به صورت binary ذخیره می کند
- نوعی برای ذخیره متن
- نوعی برای ذخیره رشته ها متن های بزرگ ntext

* با کمک این متغیر ها می توان سقف ها ستون های جدول را مشخص کرد.

دستورات DDL
- Creat ← برای ساختن
- Alter ← برای ویرایش
- Drop ← برای حذف

دستور ایجاد جدول : وقتی می خواهیم جدول بسازیم باید اول تعیین کنیم که چه اسمی دارد و چه ستون هایی دارد.

```
CREAT TABLE نام جدول (
    نام ستون ۱ نوع ستون ۱ محدودیت های ستون ۱ ,
    نام ستون ۲ نوع ستون ۲ محدودیت های ستون ۲ ,
    ⋮
);
```

ممکن است محدودیت نداشته باشیم

ستون ها با این علامت هم ② ← جدا می شوند.

هیچ مقدار ناپذیر بودن ستون ۲ را تضمین می کند ≡ می تواند NOT NULL باشد

نکته ← SQL حساس به حروف بزرگ و کوچک نیست پس دستورات به هر طریقی نوشته شوند، پذیرفته می شوند. ولی اصولاً کلمات کلیدی با حروف بزرگ نوشته می شوند.
(دقت کنید که این حساسیت فقط برای دستورات وجود ندارد.)

نکته ← SQL انعطاف پذیری بالایی دارد مثلاً برای آن int یا integer تفاوتی نمی کند، یا creat یا T

دستور تغییر (ویرایش) جدول :

ALTER TABLE نام جدول (
, محدودیت ستون نوع ستون نام ستون Add ← برای اضافه کردن ستون به جدول
, نام ستون column drop ← برای حذف ستون از جدول
, محدودیت ستون نوع ستون نام ستون modify
);

اگر خواستیم یک ستون از جدول را به کلی تغییر دهیم، از این دستور استفاده می‌کنیم.
(به شرط آنکه ستون حتماً جزو ستون های جدول باشد؛ در غیر این صورت اول باید آن را اضافه کنیم.)

دستور حذف جدول :

DROP TABLE نام جدول

* حذف جدول با Drop انجام می‌شود که جزو دستورات DDL است ولی برای حذف سطر از جدول از delete استفاده می‌شود که جزو دستورات DML است.

دستورات DML (داده ها را مورد پردازش قرار می‌دهد روی سطرها کار می‌کند.)
- Select ← برای انتخاب سطرها
- insert ← اضافه کردن سطر به جدول
- update ← ویرایش سطر های جدول
- Delete ← سطرها را از جدول حذف می‌کند.

دستور بازیابی (SELECT) :

Select نام ستون۱ , نام ستون۲ , ...
from نام جدول
where شرط

* معادل همان عملگر project است با این تفاوت که project سطر های تکراری را حذف می‌کرد اما اینجا اینگونه نیست. اگر نخواهیم سطر های تکراری حذف شود باید از دستور زیر استفاده کنیم.

اساستورها

Select [ALL | DISTINCT]
from نام جدول
where شرط

⇐ در کوئری نویسی هرجا [ ] گردشه بود نشانگر اختیاری بودن عبارت داخل گردشه و | ← بیانگر دستور OR است.

مثال

Select name, family from S
where city = 'تهران'

نام و نام خانوادگی دانشجویانی را به ما می‌دهد که ساکن در شهر تهران هستند (اسامی تکراری را هم به ما می‌دهد)

Select distinct name from S
where (city = 'تهران' OR Avg > 17)

اسامی نحصر به فرد دانشجویانی را به ما می‌دهد که شرط ها برای آن برقرار باشد.

شرط‌هایی که جلوی where می‌توانند قرار بگیرند:

۱- = ۲- <> یا != ۳- > ۴- < ۵- >=

۶- like / is null / is not null / All / any / In ....

* سمت چپ select یک دید خارجی است.

* با استفاده از دستور Select می‌توان عملگرهای π, ρ, Extend, rename را شبیه‌سازی کرد.

مثال ۱ Select name + ' ' + family As fullname => دو ستون family, name در جدول student هستند. حال در اینجا محتویات این دو ستون را با یک فاصله بین آنها کنار هم قرار می‌دهد و ستون جدیدی با عنوان fullname می‌سازد.
from student
↓
همان کار rename را انجام می‌دهد.

Select SID As studentID => ستون SID را به studentID تغییر نام می‌دهد
from student As S
همان کار ρ را انجام می‌دهد.

* Select نام ستون یا توابع جمعی و عملگرها [As نام جدید] کاربرد Extend

Select w * h As Area
from shape
} با این دستورات می‌توان مساحت هر شکل را با داشتن طول و ارتفاع بدست آورد.

در پاسکال ناحیه با این علامت نشان داده می‌شود.

Select " " from student => به ازای هر ستون student، یک spase چاپ می‌کند.

اگر بخواهیم به ازای هر سطری که داریم، یک عبارت ثابت در جلوی آن نوشته شود، مثلاً از دستور زیر استفاده می‌کنیم.

Select 'T' As fullname
from student

در اینجا ابتدا یک ستون fullname ساخته می‌شود و سپس در این ستون مقابل هر سطر عبارت ثابت T نوشته می‌شود.

* شرط‌های مقابل where در ساختار Select:

۱- کوچکتری و بزرگتری روی string ها از لحاظ الفبایی چک می‌شود. مثلاً 'تهران' <= city شهرهایی را به ما می‌دهد که با الفبای قبل از ا شروع شده باشند.

۲- شرط is null:

Select * from S
where city is null
} سطرهایی را به ما می‌دهد که در ستون مربوطه city آن null قرار داشته باشد (null با اینکه هیچی در جدول نوشته نشده باشد، کاراکتر دارد.)

به معنای در نظر گرفتن همه‌ی ستون‌ها است

3- شرط is not null :

```
Select * from student
where city is not null
```

→ سطرهایی را به ما می دهد که ستون city مربوط به آن null نباشد حتماً دارای مقدار باشد.

4- شرط ها می توانند ترکیبی باشند. یعنی زمانیکه چندین شرط داشتیم می توانیم با عبارت AND , OR این شرط ها را ساده سازی کرد.

5- برای مشخص کردن چند شرط می توانیم از کلمه‌ی In استفاده کنیم که این کلمه معادل OR است.

```
Select * from student
where city in ('تهران', 'کاشان', 'اصفهان')
```

=> اطلاعات دانشجویانی را به ما می دهد که ساکن یکی از این سه شهر باشد.

```
Select w*h As Area from shape
where w between 10 and 20
```

برای مقایسه اعداد، معادل in یا

6-

```
Select * from SP
where Qty > All (100, 150, 380, 300)
```

} در جدول SP سطرها را به ما می دهد که مقدار Qty آن از همه‌ی این مقادیر بزرگتر باشد.

Qty > All (100, 150, 380, 300) ≡ Qty > 380

به چون زمانیکه Qty از همه‌ی این مقادیر بزرگتر باشد، باید از max این اعداد نیز بزرگتر باشد.

* خیلی جاها این اعداد داخل پرانتز مشخص نیستند و خودشان کوئری هستند که به آنها کوئری تو در تو گوییم.
مثلاً لیست دانشجویانی که نمره آنها از همه نمرات الگوریتمی بیشتر است که باید در پرانتز نمرات الگوریتمی را بنویسیم.

* می توان به جای All از any استفاده کنیم که دارای مفهوم زیر است.

```
where Qty > any (100, 150, 380, 300)
```

→ سطرهایی را به ما می دهد که Qty آن از یکی از اینها بزرگتر باشد که ≡ Qty > 100

7- شرط like : برای شبیه بودن استفاده می شود (بطور مثال برای پیدا کردن سطرهایی که شهرشان مساوی یا شبیه به یک عبارتی است)

```
Select name from S
where city like 'تهران'
```

در اینجا ≡ city = 'تهران'

* فرق like و = بیشتر در پیدا کردن شهرهای مشابه است.
در این شرط از دو علامت %, ؟ نیز استفاده می شود.

```
where city like '%T%'
```

→ شهرهایی را به ما می دهد که حرف T بین حروف آن باشد

```
where city like 'T%'
```

→ شهرهایی مورد نظر ما است که با حرف T شروع شده باشد

where city like '%T' → شهرهایی مورد نظر ماست که با حرف T تمام شده باشد
where city like '?T%' → شهرهایی مورد نظر ماست که حرف T حرف دوم آن باشد

* بعد از آنکه شرایط را اعمال کردیم و سطر ها را بدست آوردیم، می توان براساس ستون های این سطرها را مرتب سازی کرد که اینکار را با دستور زیر می توان انجام داد.

order by نام ستون ← نام ستون هایی می توانند قرار بگیرند که حتماً در جدول باشند.

مثال
```
Select * from S
where ---
order by family, name, SID
```
→ براساس family مرتب می کند، کسانیکه فامیل یکسان دارند، براساس نام و اگر نام یکسان داشتند براساس شماره دانشجویی مرتب می کند.

* لزومی ندارد ستون های جلوی order by، جلوی Select باشد ولی باید در ستون های جدول باشد.

مثال
```
order by family Asc , name desc
```
مرتب سازی صعودی ← family
مرتب سازی نزولی ← name

ستون family را به ترتیب صعودی مرتب می کند و ستون name را به صورت نزولی مرتب می کند

* می توان از توابع جمعی استفاده کرد و ستون های مشتق شده دیگری را از ستون ها بدست آوردیم.

```
Select sum(Qty) As S , Avg(Qty) As A
from SP
```

در اینجا مجموع Qty را در ستون جدیدی با اسم S قرار می دهد و میانگین Qty را در ستونی به نام A قرار می دهد. پس این کوئری دارای دو ستون است (A, S)

اگر خواستیم میانگین و مجموع را برای S1 حساب کنیم می نویسیم:

```
where SID = 'S1'
```

حتی روی ستون های مشتق شده می توانیم شرط بگذاریم.

```
Select w*h As Area
from shap
where w*h > 20
```
} ⇒ اشکالی را به ما می دهد که مساحت آن از 20 بزرگتر باشد.

در اینجا می توانیم بنویسیم Area > 20 چون هنوز چیزی به اسم Area را نمی شناسد

بین دو کوئری می توان اجتماع، اشتراک و تفاضل گرفت.
union ← این دستور دو کوئری را با هم اجتماع می کند.
Intersect ← این دستور اشتراک بین دو کوئری را می گیرد.
Except ← کوئری اول را - کوئری دوم می کند.

```
Select name from S
UNION
Select name from P
```
⇒ حاصل این دستور، ستونی است که حاوی اسامی دانشجویان و کارمندان است

union [All] ← همه‌ی اسامی حتی تکراری‌ها را به ما می‌دهد.
union [distinct] ← بعد از اجتماع اسامی، اسامی تکراری را حذف می‌کند.

```
Select [distinct] name from S
UNION
Select [distinct] name from P
```
⇒ اسامی منحصر به فرد اساتید و اسامی منحصر به فرد دانشجویان را به ما می‌دهد ولی لزوماً اجتماع آنها منحصر به فرد نیست.

* می‌توانیم Pi name و family از S را با هم اجتماع کنیم فقط باید دقت کنیم که مجموعه‌ها همتا باشند. (هم تعداد ستون‌ها، هم دامنه صفات یکسان باشند)

```
Select min (qty)
from SP
where SID = 'S1'
```
⇒ کمترین تولید S1 را به ما می‌دهد. (اگر شرط where بزرگترین تولید را به ما می‌دهد.)

اول S1 را پیدا می‌کند سپس کمترین تولید آن را پیدا می‌کند.

* اگر کمترین تولید هر تولیدکننده را بخواهیم، ابتدا جدول تولیدکننده را براساس SID دسته‌بندی می‌کنیم و سپس کمترین تولید هر دسته را برمی‌گردانیم.

```
Select SID, min (qty)
from SP
group by SID
```
⇒ این دستور اقدام به دسته‌بندی براساس SID می‌کند.

⇒ با group by می‌توان سطرهای جدول را براساس تعادل یک ستون ساده گروه‌بندی می‌کند به طوری که در هر گروه، مقدار آن ستون یکسان است.

در اینجا اول جدول SP را براساس SID دسته‌بندی می‌کند. همه‌ی SID های یکسان در یک دسته و هر دسته در یک سطر قرار می‌گیرد.
در summerize به جز ستون انتخابی و توابع جمعی نمی‌توانیم ستون دیگری بنویسیم پس اینجا هم جلوی group by یا باید ستون‌های انتخابی را بنویسیم و یا از توابع جمعی استفاده کنیم.

مثال) در دانشجویان از هر اسم چه تعدادی است؟

```
Select name, count (*)
from student
group by name
```
⇒ تعداد هم دسته را به ما می‌دهد.

شرط گذاری روی دسته بندی به دو صورت زیر انجام می شود :

۱ - اعمال شرط بعد از دسته بندی ⇐ توسط دستور زیر انجام می شود :

having شرط

۲ - اعمال شرط قبل از دسته بندی ⇐ قبل از دسته بندی ابتدا سطر های جدول براساس شرایط خاصی فیلتر می کند و سپس دسته بندی روی این نتایج جدید اعمال می شود و نتیجه نهایی حاصل از select در اختیار ما قرار می گیرد .

where شرط

مثال

```
select Pid , sum(Qty) As total from SP
were Pid in ('P1' , 'P2' , 'P6')
group by Pid
having sum(Qty) > 500
```

این ستون از هر دسته یک نماینده انتخاب می کند.

شرط قبل از دسته بندی

شرط گذاری بعد از دسته بندی

لزومی ندارد که شرطی که جلوی having می آید بین ستون ها باشد بلکه می تواند تابع جمعی از ستون های دیگر نیز باشد

مثال

```
Select name , count(*)
from Student
where city = 'تهران'
group by name
having count(*) > 10
```

اگر نام دانشجویان تهرانی را بخواهیم که تعدادشان بیشتر از ۱۰ نفر باشند به صورت زیر عمل می کنیم ⇐

* در اینجا ابتدا دسته بندی بعد از جدا کردن دانشجویان تهرانی از سایر دانشجویان براساس اسم انجام می شود .
شرط having مشخص می کند که چه اسم هایی بیشتر از ۱۰ مرتبه تکرار شده اند .

* در group by ترتیب ستون هایی که در جلوی آن می آید مهم نیست اما در order by ترتیب ستون ها مهم است چون براساس آن مرتب سازی انجام می شود .

فرض کنید cs (SID, city, cID, grad) حال مدل (دانشجویان) شهرهای پرجمعیت را بیابید.

باید در اینجا دو مفهوم را تعریف کنیم:

مدل شهر ⇐ معدل دانشجویانی که از آن شهر هستند.

شهر پرجمعیت ⇐ شهری که بیشتر از 100 دانشجو داشته باشد.

```
Select   city, Avg(grad) As A
  from   cs
  group  by  city
  having   count( SID) > 100
```

⇐ وقتی دسته‌بندی براساس یک ستون انجام می‌شود، از توابع حمجی بقیه‌ی ستون‌ها می‌توانیم کوئری‌های مختلف را جواب دهیم. که می‌توانند (این توابع) جلوی Select و یا در شرط having بیایند.

می‌خواهیم معدل دانشجوهای تهرانی را حساب کنیم.

```
Select   city,  Avg( grad)
  from   cs
  where  city = 'تهران'
  group by  city
```

⇐ اول فیلتر می‌کند دقیقا شهر تهران را برمی‌دارد و سپس معدل آنها را حساب می‌کند. :: برنامه‌ی اشتباه

درست ⇐

```
Select   SID, city, Avg (grad)
  from   cs.
  where  city = 'تهران'
  group by SID
```

مثال

```
Select   SId,  count( * )  As  c
  from   cs
  where    grad > 17
  group by   SID
```

⇐ در این کوئری ابتدا رکوردهای زیر 17 را حذف می‌کند سپس سطرهای جدول را براساس شماره دانشجویی دسته‌بندی می‌کنیم و در نهایت تعداد درس‌هایی که دانشجوها گرفته، در آن عددهای بالای 17 دارد را به ما می‌دهد.

* ما تا اینجا هی اطلاعات خود را از یک جدول بدست می‌آوردیم، اما اگر بخواهیم اطلاعات را از چندین جدول بدست آوریم، دو راه داریم:

1 ـ جدول‌ها را join کنیم.

2 ـ از کوئری‌های تو در تو استفاده کنیم.

* کوئری داخلی (subQuary) می‌تواند به‌عنوان ستون جلوی Select یا به‌عنوان جدول و یا به‌عنوان شرط جلوی where حساب شود.

* اگر بخواهیم تعداد درس‌های دانشجویان تهرانی را پیدا کنیم، دو راه وجود دارد:

1 ـ اگر به شماره دانشجویی دانشجویان تهرانی دسترسی داشته باشیم:

```
where  SID  in ( S1 , S2 , S3 )
```

2 ـ به شماره دانشجویی دانشجویان تهرانی دسترسی نداشته باشیم. (پس ابتدا باید این شماره دانشجویی را با یک کوئری دیگر بدست آوریم)

```
where  SID  in ( Select   SId   from  S
                 where  city = 'تهران' )
```

* اگر شماره دانشجویی دانشجویانی را بخواهیم که همه نمرات آن از S_1 بیشتر باشد، به صورت زیر عمل می‌کنیم:

۱- یک کوئری برای پیدا کردن نمرات S_1 می‌خواهیم (کوئری داخلی)

۲- کوئری دیگر دانشجویانی را به ما می‌دهد که نمرات آن از همه نمرات S_1 بیشتر باشد.

```
Select SId from CS
    where grad > All ( Select grad from CS
                       where SId = 'S1' )
```

اگر به نمرات S_1 دسترسی داشته باشیم، احتیاج به نوشتن این کوئری داخلی نیست.

مثلاً `where grad > ALL (14, 15, 17)`

(grad باید عدد باشد)

* اگر نام و نام خانوادگی دانشجویانی را بخواهیم که نمرات آن از همه‌ی نمرات S_1 بیشتر باشد را بدست آوریم، به صورت زیر عمل می‌کنیم.

```
Select name from S
    where SId in ( Select distinct SId
                   from CS
                   where grad > ALL ( Select grad
                                      from CS
                                      where SId = 'S1' ))
```

* می‌خواهیم نام و نام خانوادگی دانشجویانی که نمره‌ای بالاتر از کلیه نمرات عبدی را دارند را پیدا کنیم، پس به صورت زیر عمل می‌کنیم.

(چون از دو جدول S و CS استفاده می‌کنیم، احتیاج به کوئری های تودرتو داریم.)

```
Select name from S
  where SId in ( Select distinct SId
                 from CS
                 where grad > ALL ( Select grad
                                    from CS
                                    where SId = ( Select SId
                                                  from S
                                                  where name = 'کبری'
                                                  And family = 'عبدی' )))
```

* ابتدا داخلی ترین کوئری عمل می‌کند و سپس به سمت بیرون حرکت می‌کنیم.

درحقیقت با کوئری های تودرتو عملگر ← پیاده‌سازی می‌کنیم.

* فرض کنید نام تولیدکننده ها، بیشترین تولید، تولیدکنندگان تهرانی را بخواهیم. پس به صورت زیر عمل می‌کنیم.

```
Select   T.name , ( Select  Max(qty)  from  SP
                          where  sp.sid = T.sid )  As  MX
   from ( Select  *  from  S
            where  city = 'تهران' )  As  T
   where  exists ( Select  *  from  SP
                     where  sp.sid = T.sid )
```

* می‌خواهیم نام دانشجویانی که درس ها 4 واحدی گرفته‌اند را بیابیم پس:
در ابتدا باید درس های 4 واحدی را از جدول درس انتخاب کنیم:

```
Select    sid    from  cs
   where   cid  in ( Select  *  from  c  where  unit = 4 )
                      !cid
```

حال نام این دانشجویان را پیدا می‌کنیم:

```
Select   name   from   s
   where   sid  in ( Select  sid  from  cs
                       where  cid  in ( Select  cid  from  c
                                          where  unit = 4 ))
```

* اگر به جای name بنویسیم * همه‌ی مشخصات دانشجویان که درس 4 واحدی گرفته‌اند را به ما می‌داد.

* می‌خواهیم ببینیم از هر شهر چند تا دانشجو درس های 4 واحدی گرفته‌اند:
در اینجا باید براساس شهر دسته‌بندی انجام دهیم.

```
Select   city , count(*)
   from ( Select  *    from  S
            where  sid  in ( Select  sid  from  cs
                               where  cid in ( Select cid  from  c
                                                 where  unit = 4 )))
   As  T
   group by  T.city
```

* پیاده‌سازی پیوند شرطی:

```
Select   نام‌جدول۱ . نام‌ستون۱ , ... , نام‌جدول۲ . نام‌ستون۲ , ...
   from   نام‌جدول۱ , نام‌جدول۲ , ...
   where  شرط
```

ستون انتخابی از جدول اول

Exists ← چک می کند که آیا کوئری داخلی دارای سطر هست یا نه ؟ که اگر حداقل یک سطر داشت کوئری چاپ می شود و در غیر این صورت هیچ چیز چاپ نمی شود .

مثال) Select name from (S) ← تولید کننده
where exists (select * from SP
where SP.SId = S.SId)

⇐ به ازای هر سطر از جدول S ، کوئری داخلی را چک می کند که آیا سطری دارد یا نه و اگر داشت نام آن را چاپ ولی اگر نداشت سراغ سطر بعدی می رود .

SP

SId	CId	Qty
S_1		

S

SId	name	...
S_1	فن آوران	
S_2	توربین تن	
S_5	لنت	

خروجی ← فن آوران

* کوئری داخلی هم ستون های جدول خود و هم ستون های کوئری خارجی را می بیند . پس در کوئری داخلی می توان از شرط های کوئری خارجی استفاده کرد .
* اگر اسم و بیشترین تولید هر تولیدکننده را بخواهیم به صورت زیر عمل می کنیم :

```
Select  SId , name ,  ( Select  max ( Qty )
                        from  SP
                        where  SP.sid = S.Sid )  As  c
from  S
```

* اینجا کوئری داخلی به عنوان ستون در نظر گرفته شده است .
* کوئری داخلی سراغ جدول SP می رود و بیشترین تولید کسی را پیدا می کند که SId = S.SId است .
* کوئری داخلی ممکن است هیچ سطری پیدا نکند پس null برمی گرداند .
* کوئری های جلوی Select حتماً باید اسم داشته باشند و اگر نداشته باشند ، خودش اسم می دهد .
* نتیجه کوئری بالا جدولی است که ستون اول آن شماره Id ، ستون دوم نام و ستون سوم ماکزیمم تولید ، تولیدکننده است .
* اگر در جدول S ، S_{10} داشته باشیم اما در جدول SP آن را نداشته باشیم ، اسم S_{10} در جدول نتیجه می آید و ستون c آن null می شود چون در جدول SP ، S_{10} پیدا نمی کند .

حال فرض کنید نام تولیدکنندگانی را بخواهیم که تولید دارند ؛ یعنی آنهایی که تولیدشان برابر null است را نداریم ، پس باید به صورت زیر عمل کنیم :

```
Select  name , ( Select  max (Qty)  from  SP
                 where  SP.Sid = S.Sid )  As  MX
from  S
where  exists ( Select  sid  from  SP
                where  sp.sid = S.sid )
```

* شرط جلوی where هم می‌تواند روی جدول اول باشد، هم می‌تواند روی جدول دوم باشد و ...
* اثر where را تئوریسیم ضرب دکارتی را پیاده‌سازی کرده‌ایم.
* اگر بنویسیم * . نام جدول ← همه‌ی ستون‌های جدول i منظورمان است

* اسم دانشجوهایی که درس‌های 4 واحدی گرفته‌اند و نمره‌ی آن‌ها و اسم آن درس و اسم درس را می‌خواهیم
اطلاعات از سه جدول S , cs , c است.

```
Select  S.name  As  Sname  ,  c.name  As  cname  ,  grade
from  S, c, cs
```

دو ستون name داریم که باید اسم‌ها متفاوت باشند که یا خودمان اسم آن‌ها را عوض می‌کنیم یا SQL در خروجی name1 , name چاپ می‌کند.
چون در بالا where را نوشتیم منظورمان ضرب دکارتی این سه جدول است، سپس روی این سه ستون تیلتا انجام داده‌ایم.
یعنی:

$$\Pi_{S.name,\ C.name,\ grade}\ (S \times CS \times C)$$

چون دانشجویانی که درس‌های 4 واحدی گرفته‌اند را می‌خواهیم پس شرطمان باید با توجه به این سمت تعیین شود.

```
where  c.unit = 4  and  S.id = cs.sid  and  c.id = cs.cid
```

و این عبارت معادل است با

$$\Pi\ (\ \sigma_{unit=4}\ (\ S \underset{S.sid=cs.sid}{\times} CS \underset{C.id=cs.cid}{\times} C\ )\ )$$

مثال) نام و نام خانوادگی استادان و دانشجویان هم‌شهری را می‌خواهیم:

```
Select  S.name  As  Sname  ,  S.family  As  Sfamily
      , P.name  As  Pname  ,  P.family  As  Pfamily
from  S, P
where  S.city = P.city
```

} ≡ $S \times P$, $P.city = S.city$

* اگر در پیوند شرطی شرطمان تساوی روی ستون‌های مشترک باشد به ∞ می‌رسیم.

برای پیوند طبیعی در کوئری نویسی دو روش داریم:

1- inner join
2- پیوند شرطی با شرط تساوی روی ستون‌های مشترک

} ⇐ از لحاظ زمانی، فضایی این دو روش یکسان‌اند.

عملگر Rename ≡ نام‌ستون‌را تغییر می‌دهیم

مثال: Select S1.name As Sname , S2.name As Sname2
from S AS S1 , S As S2
where S1.city = S2.city and S1.id <> S2.id

عملگر ρ ≡ نام‌جدول‌را تغییر می‌دهیم

S1.S2 هردو اشاره‌گر به جدول S‌اند و چون خود دانشجو با خودش هم‌شهری است این شرط را برای حلوگیری از تکراری گذاشتیم.

* پیوند طبیعی را می‌توان روی یک جدول نیز انجام داد که چون همه‌ی ستون‌ها مشترک‌اند، ∞ معادل اشتراک است.

$$A \infty \rho_B(A)$$

B یک جدول A است.

ساختار کلی inner join:

Select نام‌جدول۱.* , نام‌جدول۲.*
from نام‌جدول۱ [inner] join نام‌جدول۲ on شرط‌پیوند
[where .....]
[group by ....]
[order by ....]

در اینجا ستون‌های مشترک را مشخص می‌کنیم.

ستون‌های مشترک ⇐ نام‌ستون x . نام‌جدول۱ = نام‌ستون y . نام‌جدول۲

مثال:
Select S.name , C.name , CS.grad
from S inner join CS on S.id = CS.id
inner join C on C.id = CS.cid
where S.city = 'تهران'

* الان همه‌ی جدول‌ها را داریم و می‌توانیم روی همه‌ی جدول‌ها شرط بگذاریم.
* درست است که join خاصیت شرکت‌پذیری دارد اما در اینجا نمی‌توانیم از S ∞ C استفاده کنیم چون جواب به دست نمی‌آید در حقیقت ما در آن ستون مشترک نداریم.

مثال: با توجه به جدول C، CS می‌خواهیم ببینیم چند تا دانشجو درس‌های 4 واحدی گرفته‌اند و از هر واحد چند تا هست: کامل است
روی unit دسته‌بندی کنیم و سپس count، city می‌گیریم.

* اگر به جای inner join از outer join استفاده کنیم، تراپندی می‌شود.
پس می‌توانیم L.O.J - F.O.J - R.O.J داشته باشیم.

در مثال قبل چون cid , sid هیچ کدام نمی‌توانستند مقدار null بگیرند پس join , outer join آن فرقی ندارد .

* در حالت کلی اگر در یک رابطه صفتی وجود نداشت که مقدار null بگیرد یعنی در پیوند سطر پیوند نشدنی در آن وجود ندارد پس در آن پیوند طبیعی و شرایط پیوند تفاوتی نمی کند .
* اما مثلاً اگر در رابطه B (... , x , ...) ستونی مانند x وجود داشت که بتواند مقدار null بگیرد یعنی سطر پیوند نشدنی دارد در حتماً باید از شرایط پیوند استفاده کنیم .

فرض کنید رابطه‌ی A , B به دو شکل زیر باشند :

A (.... , x, ...)

B(---- , x, ---)

null

a

b

c

می دانیم که در پیوند طبیعی (∞) سطرهایی با هم پیوند می شوند که در ستون مشترک دارای مقدار یکسان باشند پس در اینجا اگر رابطه B مهم نباشد و ما در رابطه‌ی A ، a , b , c را نداشته باشیم ، ∞ می گیریم چون مقادیر a , b , c جلوی هیچ سطری از رابطه قرار نمی گیرند .

اگر B مهم باشد از شرایط پیوند استفاده می کنیم که اگر B سمت چپ باشد ن.L.O ، اگر راست باشد ن.R.O است .

مثال)

```
Select  A.* , B.*
from    B   Left. outer. join   A   on   A.x = B.x
```

اگر چیزی نفهمیدن توقع من نیست نبینه
چون منم از روی جزوه یه بنده خدای دیگه این صفحه رو نوشتم ☺

در رابطه‌ی دانشکده و استاد رابه‌صورت زیر درنظر بگیرید. می‌خواهیم نام و نام خانوادگی استاد و نام دانشکده را پیدا کنیم.

Department (Id , name , Location) prof (Id , name , family, DeptId)

→ Select P.name , P. family , D.name As Dept
from Prof As P , Depatment As D
where P. DeptID = D. Id → در اینجا ستون مشترک id دانشکده است پس
با جداز join دو جدول می‌توانیم اسم دانشکده را نیز پیدا کنیم.

پس با join دو جدول کوئری بالا معادل کوئری زیر است.

Select ....
from Prof As P inner join Department As D
on P. DeptId = D. id

در اینجا احتیاج به نوشتن where نیست چون خودش به صورت پیش‌فرض شرط را روی ستون مشترک بررسی می‌کنیم.
* این دو کوئری از لحاظ محاسبات و نتیجه یکی هستند و حاصل آن به ما جدولی شامل سه ستون name , family , Dept می‌دهد.
به مثال زیر توجه کنید:

ID	name	Location
1	علوم ریاضی	null
2	فیزیک	null
3	شیمی	علوم

id	name	family	depthId
1	اکبر	عبدی	1
2	رضا	صادقی	2
3	شاهرخ	استخری	null

=>

name	family	Dept
اکبر	عبدی	علوم ریاضی
رضا	صادقی	فیزیک

در اینجا می‌بینیم که نام یک استاد وجود ندارد و در واقع استادی که دانشکده ندارد وجود ندارد و این نقض است. پس اگر بخواهیم همه‌ی اساتید را داشته باشیم، باید از فرا پیوند استفاده کنیم.

استفاده از فرا پیوند (L.O.J)

Select ....
from prof As P Left outer join
Department As D on p. DeptID = D. Id

name	family	Dept
اکبر	عبدی	علوم ریاضی
رضا	صادقی	فیزیک
شاهرخ	استخری	null

می خواهیم اطلاعات دانشجویان مشروطی را پیدا کنیم و این اطلاعات را در قالب یک دید خارجی به EU نمایش دهیم پس به صورت زیر عمل می کنیم :

```
Great   view   Mashrotiha
As
Select  *   from  student
   where   Avg < 12
```

بنابراین دستور کلی تشکیل یک دید به صورت زیر است :

```
Creat   view    نام دلخواه
AS
Select   Quary
```

* این نام دلخواه که برای یک دید در نظر می گیریم در هیچ جا ذخیره نمی شود و به عبارت دیگر در قالب جدول نیست و در سطح انتزاعی است .
* view در DBMS ذخیره می شود ولی وابسته به محل ذخیره سازی نیست .
* بنابراین از روی جدول های پایگاه داده یک view برای نمایش به کاربر می سازیم .

مثال

```
Creat  view   تولید [ ( نام تولیدکننده , نام محصول , تعداد) ]
   As
   Select   S.name , P.name , SP.Qty
        from  S  innerjoin  SP  on  S.id = sp.sid
              innerjoin  P  on  SP
```

* بعد از آنکه view ساخته شد، به جای S.name نام تولیدکننده و به جای P.name نام محصول و به جای SP.Qty تعداد تولید دیده می شود . پس view دارای سه ستون نام تولیدکننده و نام محصول و تعداد است .

انگار را برای این انجام می دهیم که ممکن است کاربر بداند که ستون هایی که ما برای جدول های خود در نظر گرفته ایم دقیقاً چیست ؟ برای همین خودمان نام ستون ها را در view تغییر می دهیم .

* برای حذف یک view از drop view استفاده می کنیم .

دستور Insert :

هرگاه بخواهیم ستون هایی را به جدول اضافه کنیم ، از آن استفاده می شود . که ساختار کلی آن به صورت زیر است .

```
insert  into   نام جدول [ ( نام ستون1 , نام ستون2 , ... ) ]
value ( مقدار1 , مقدار2 , ... )
```

اگر نخواستیم همه ی ستون ها را مقدار دهیم ، نوشتن این قسمت ضروری است .

انتظار ما از این کوئری به این صورت است که به جای نام ستون 1 ، مقدار 1 و به جای نام ستون 2 ، مقدار 2 و ... قرار دهیم.
نوشتن نام ستون ها در این کوئری اختیاری است و اگر آن را ننویسیم باید کلیه‌ی ستون های جدول را به ترتیب مقدار دهی کنیم.

مثلاً : S (id , name , city)

کوئری →
```
insert into S
    value ( 'S10' , 'چهارگون' , 'تهران' )
```
این مقادیر به ترتیب ستون های S را مقداردهی می کند.

اگر خواستیم که همه‌ی ستون ها را مقدار ندهیم و یا ترتیب مقدار دهی به ستون ها را تغییر دهیم باید نام ستون هایی را که می خواهیم در جلوی نام جدول قرار دهیم.

```
insert into S ( name, id )
    value ( 'چهارگون' , 'S10' )
```

* اگر یک دستور insert را دوبار اجرا کنیم، به ما خطای دهد چون مثلاً در این مثال یکبار به id که کلید اصلی است مقدار S_{10} داده شده و اگر برای بار دوم بخواهیم به آن S_{10} را بدهیم چون تکرار پیش می آید، خطا می دهد. چون کلید اصلی مقدار تکراری نمی پذیرد.

* اگر خواستیم به ستون ها چندین مقدار بدهیم می توانیم از چندین پرانتز استفاده کنیم.

```
insert into S
    value ( S10 , 'چهارگون' , 'تهران' )
          ( S11 , 'Test' , 'کاشان' )
          ( S12 , 'هوشمند' , 'اصفهان' )
```

نکته =>
دستور insert تنها روی جدول ها انجام می‌شود. روی view اجرای این دستور امکان پذیر نیست.

اگر خواستیم یک سری اطلاعات به جدول اضافه کنیم باید به صورت زیر عمل کنیم که با انجام آن کوئری Select یک سری سطر بدست می آید که می توان آن را به جدول اضافه کرد.

```
insert into نام جدول [ ( نام ستون ها ) ]
    Select Query
```

مثال)
```
insert into mashrotiha ( Id, name, family)
    Select id , name , family
        from student
        where Avg 12
```

نکته ۱ : جدول mashrotiha باید قبلاً ساخته شده باشد .

نکته 2 : ترتیب ستون هایی که جلوی Select قرار می گیرد باید با ستون های جلوی نام جدول، یکی باشد .

کوئری معادل قبل

```
Select  id, name, family
from    student
into    mashrotiha
where   ---
```

← در اینجا اطلاعات را از یک جدول بدست می آوریم و در جدول دیگر می ریزیم

* برای حذف جدول از دستور drop و برای حذف داده های آن از دستور delete استفاده می شود .

دستور حذف :

```
Delete  from    نام جدول
        where   شرط
```

مثال →

```
Delete  from    sp
        where   qty <10
```

در اینجا سطر هایی که برای آن شرط برقرار است را حذف می کند که در مثال مقصولاتی که کمتر از ۱۰ تا تولید شده اند حذف می شوند .

```
Alter   table   s
remove  city
Add     x       varchar (10)
modify  name    nvarchar (100)
```

← جدول s را ویرایش کرده و ستون city را حذف می کند و
← اضافه کردن ستون x از نوع varchar(10)
← برای ویرایش ستون name استفاده می شود

دستور update :

برای ویرایش سطر ها از این دستور استفاده می کنیم .

```
update      نام جدول
set
نام ستون 1  =  مقدار 1  ,
نام ستون 2  =  مقدار 2  ,
نام ستون 3  =  مقدار 3
        [ where   شرط ]
```

← این مقادیر را می توانند توسط select Quary نیز بدست آورد

← اگر این دستور نباشد همه ی سطر ها ی جدول را ویرایش می کند .

مثال ۱

```
update  P
set     color = null
```

← همه ی سطر های P را ویرایش می کند و در آن هر سطر در ستون color مقدار null می گیرد پس در نهایت ستون color در جدول دارای مقدار null خواهد بود .

مثال ۱

```
update  sp
set     qty = 1400
where   sid = s1
```

← تمامی محصولاتی که s1 تولید می کند را تعداد محصولات آن را به 1400 تغییر می داد .

مثال
```
update   student
set      Avg = 19
         city = 'تهران'
where    id = 'S1'
```
این دستور معدل و شهر دانشجوی S_1 را تغییر می دهد.

مثال
```
update   SP
set      Qty=( Select   max (Qty)   from   SP
where    Sid = 'S1'   and   Pid = 'P1'
```
تولید P_1 و S_1 را به بیشتر تعداد تولید تغییر می دهد.

مثال
```
update   S
set      name = 'پارسا'  ,  family = 'پیروزفر'
where    id in (1, 2, 3)
   and   city = 'تهران'
```
→ اگر این شرط نبود همه ی دانشجویان را به پارسا پیروزفر ویرایش می کرد.

دستور Delete from نام جدول [where شرط] سطرهایی از جدول که دارای این شرط باشند را حذف می کند و در صورتیکه شرط where نباشد کلیه ی سطرهای جدول را حذف می کند.

دستور Truncate table نام جدول ≡ Drop برای حذف کل جدول بکار می رود.

تفاوت (Truncate , Delete):
Delete ← درون دونه سطرها را پاک می کند
Truncate ← جدول را دوری انداخته و دوباره می سازد که در این حالت برای جدول هایی با رکورد های بالا سرعت بیشتر است.
هر دو جزء دستورات DDL هستند.

اگر خواستیم به کاربران مجوزهایی برای اعمال تغییرات در جدول ها بدهیم برای آن ها دسترسی مشخص کنیم باید از دستور زیر استفاده کنیم.

```
Grant   < priviteges >
on      < tables >
to      < users >
[with grant option]
```

مثال →

```
Grant   Insert , update
on      S,P
to      Ali, Akbar
with grant option
```

به دو کاربر Ali , Akbar اجازه می دهیم روی جدول عملیات insert , update انجام دهد.

* مجوزهایی که به کاربر داده می شود شامل موارد زیر است:
update(x) - Delete - update - insert

علی راکبر با همین دستور، دستورات را به کاربرانی که در DBMS تعریف شده اند هم می دهد.
با یک دستور Select همه‌ی کاربران را انتخاب می کند و دسترسی روی جدول ها را به آن ها می دهد.

پس گیری یا لغو مجوز:

vevoke grant option
on <tables>
from <users>
→ حق دسترسی و اعطا را از کاربر می گیرد.

vevoke <priviteges>
on <tables>
from <users>
→ همه‌ی مجوزها را می گیرد

حذف مجوز از کاربر:

Deny <priviteges>
on ---
from ---
→ دسترسی که از اول کاربر داشته را حذف می کند

اضافه کردن محدودیت به ستون ها

با توجه به نوع ستون ها ما می توانیم روی آن یک سری محدودیت هایی به صورت زیر قرار دهیم:

not null	primary key	foreign key	unique	check / default
ستونی که نمی تواند null باشد.	x null تکراری x	null باشد یا در جدول دیگر مقدار داشته باشد	مقادیر متفاوت منفرد	پیش فرض مقدار دهی می کند

(اگر ستون کلید اصلی باشد / اگر ستون کلید خارجی باشد)

```
Creat table نام جدول (
    Cobum - name   type   constraint ,
    name   nvarchar(50)   defult 'Akbar'
    ---
)
```

مثال → در این ستون در هر سطر به صورت پیش فرض نام اکبر را می نویسد.

مثال)
```
Alter table R (
    Add column c integer not null defult 0 )
```

مثال)
```
Creat table sp(
    SID   nvarchar(5)   foreign key   refrence (ارجاع)   S(Id)
    PId   nvarchar(5)   foreign key   refrence   p(Id
    Qty   integer   check   Qty >0 and Qty <100
```

ستون SID جدول SP معادل ستون Id جدول S است

Qty تا روی ستون Qty محدودیت می گذارد

شرطی که باید چک شود.

```
Alter table S
    Add constrant chek_Avg
     check ( Avg >= 0 and Avg < 20 )
```

```
Alter table Exam(
    Add constrant Date
    chek ( startTime ( End Time))
```

دوستان معذرت این چندتا سوال
رو نفهمیدم لطفا هنگام خوندن به خودم مراجعه کنید تا [illegible]

* صفت های چند مقداری را می توان در یک جدول نشان داد چون در این صورت پیمایش جدول برای شخص دشوار می شود پس از یک جدول دیگر استفاده می کنیم.

ID	مدرک تحصیلی
4	لیسانس و دکتری و ارشد و ...

X =>

ID فرد	ID مدرک

✓

⇐ Assertion

فرم کلی: Create Assertion ali ⟶ مثال: Create Assertion Asr

check (شرط)

check (

not Exist (Select name, family, city
from S
group by name, family, city
having count(*) > 1
)

)

← در اینجا چک می کنیم که آیا این شرط وجود دارد یا نه

← این دستور نام و نام خانوادگی و شهر دانشجویانی را که به ما می دهد که تعداد آنها بیشتر از یکی باشد.

* اصولاً از دستور Select در اینجا استفاده می شود.

مثال دیگری از Select دستور مثال قبل: Select S_1.name , S_2.name
from S AS S_1 , S AS S_2
where S_1.name = S_2.name
and S_1.id <> S_2.id
and S_1.family = S_2.family
and S_1.city = S_2.city

← اطلاعاتی از دانشجویانی را به ما می دهد که دارای نام، نام خانوادگی و شهر یکسانی دارند اما خود افراد متفاوت اند.

* هم چنین ما می توانیم روی چندین جدول نیز محدودیت بگذاریم. مثلاً ما می خواهیم مجموع تولید هر شهر از 10000 بیشتر نشود که برای این کار طبق مثال قبل از not Exist استفاده می شود در شرط Select آن ابتدا دسته بندی براساس شهر دسته بندی انجام می شود و سپس sum مربوط به تولید محصولات محاسبه می شود و حال چک می کنیم که بیشتر از 10000 نشود.

فصل 5 : وابستگی داده ای

* از وابستگی داده ای برای نرمال سازی استفاده می شود.

اگر A و B زیرمجموعه ای از صفات رابطه R باشند. می گوئیم B به A وابستگی تابعی دارد (A ⟶ B) اگر در طول حیات رابطه R به ازای هر مقدار از A تنها یک مقدار متناظر از B موجود باشد.

به ازای Date های در رابطه R در هر زمان

id	name	family
1	رضا	صادقی
2	علی	دایی

* اگر به ازای هر مقدار از name یک مقدار متناظر در family وجود داشته باشد، گوئیم که family به name وابستگی تابعی دارد. (یعنی هر جا اسم رضا دیدیم باید فامیلی آن صادقی باشد.)

با توجه به داده های فعلی جدول این وابستگی تابعی برقرار است اما در جدول زیر این رابطه برقرار نیست.

کلید

(id)	name	family
1	رضا	صادقی
2	علی	دایی
1	رضا	عطاران

name ⟶ family
id ⟶ name
id ⟶ family
} => داریم

نکته: اگر A دارای خاصیت کلیدی باشد، آنگاه به ازای هر B داریم A ⟶ B

R

id	name	family	city	code
1	رضا	صادقی	تهران	021
2	علی	دایی	کاشان	031
3	رضا	عطاران	تهران	021
4	بیژن	بنفشه‌خواه	اصفهان	031

* در طول حیات رابطه‌ی R به ازای هر مقدار از city تنها یک مقدار متناظر برای code وجود دارد.
* مهم نیست که این مقدار متناظر تکرار شود.

city ⟶ code

(id, name) ⟶ city

لزومی ندارد حتماً صفات تک مقداری باشند
می توانند چند مقداری باشند.

نکته کامل برعکس: اگر A شامل صفتی باشد که خاصیت کلیدی داشته باشد، آنگاه به ازای هر B داریم $A \longrightarrow B$.

(name, family) ⟶ family => وابستگی تابعی بیهوده

اگر $B \subseteq A$ باشد به $A \longrightarrow B$ وابستگی تابعی بیهوده گوییم.

تعریف: B به A وابستگی تابعی کامل دارد ($A \Rightarrow B$) اگر B به A وابستگی تابعی داشته باشد، و B هیچ زیرمجموعه‌ای از A وابستگی تابعی نداشته باشد. (حالت خاص: اگر $A \longrightarrow B$، و A یک مجموعه یک عضوی باشد، آنگاه $A \Rightarrow B$)

id ⟶ city

(id, name) ⟶ city => وابستگی تابعی کامل نیست

چون یک زیرمجموعه از آن وجود دارد که وابستگی تابعی دارد.

نکته: وابستگی تابعی چه کامل باشد و چه نباشد دارای خاصیت جابه جایی نیست.

* اگر رابطه‌ی R دارای صفات A، B، و C باشد به طوریکه:

$$\left.\begin{array}{l} A \longrightarrow B \\ A \longrightarrow C \end{array}\right\} \Rightarrow A \longrightarrow BC$$

بستار وابستگی تابعی:

اگر F یک مجموعه از وابستگی های تابعی باشد، F^* بستار F است و شامل F، تمام وابستگی های تابعی دیگر است که از F نتیجه می شود.

$F = \{ A \longrightarrow B , A \longrightarrow C \}$

$F^* = \{ \underbrace{A \longrightarrow B , A \longrightarrow C}_{F} , A \longrightarrow BC , A \longrightarrow A , B \longrightarrow B , \dots \}$

* F_1 ، F_2 معادل هستند اگر $F_1^* = F_2^*$ (بستار آنها یکسان باشد)

قواعد استنتاج آرمسترانگ:

ده تا قانون است که ما می توانیم با یک آنها با داشتن مجموعه‌ای از وابستگی های تابعی به بستار آنها برسیم.

۱- بازتاب ⇐ اگر $B \subseteq A$ باشد آنگاه $A \longrightarrow B$

۲- افزایش (بسط) ⇐ اگر $A \longrightarrow B$، و C یک صفت باشد آنگاه $AC \longrightarrow BC$ (به هر دو طرف اضافه می شود C)

3- تعدی ⇐ اگر $A \to B$ ، $B \to C$ آنگاه $A \to C$ ⇐ این 3 قاعده‌ی اول کامل اند و بقیه قواعد از آنها استنتاج می شود .

4- اجتماع ⇐ اگر $A \to B$ ، $A \to C$ آنگاه $A \to BC$

5- تجزیه ⇐ اگر $A \to BC$ آنگاه $A \to B$ ، $A \to C$

6- اگر $AB \to C$ آنگاه $A \to C$ ، $B \to C$ ⇐ لزوماً این قانون برقرار نیست .

6- ترکیب ⇐ اگر $A \to B$ ، $C \to D$ آنگاه $AC \to BD$

7- خودتعیینی ⇐ ازروی بازتاب بدست می آید $A \to A$

8- شبه تعدی ⇐ اگر $A \to B$ و $BC \to D$ آنگاه $AC \to D$

$A \to B$: به ازای یک مقدار از B یک مقدار برای A وجود دارد

$BC \to D$: به ازای یک مقدار از D یک مقدار برای BC وجود دارد .

$AC \to D$: پس ⇐ به ازای یک مقدار از D یک مقدار از AC وجود دارد .

9- اگر $A \to B$ و $AB \to C$ آنگاه $A \to C$

$A \to B$: به ازای هر مقدار A یک مقدار متناظر برای B وجود دارد .

$AB \to C$: ...

$A \to C$: به ازای هر مقدار A یک مقدار متناظر برای C وجود دارد .

10- اتحاد طی ⇐ اگر $A \to B$ ، $C \to D$ آنگاه $AU(C-B) \to BD$

⇐ از جنس C است .

مثال نقض :

A	B	C
x	y	z
x	w	t
x	y	z

* اگر F متناهی باشد و مجموعه صفات نیز متناهی باشد ، آنگاه F^* نیز متناهی است .

F_{opt} (مجموعه وابستگی تابعی بهینه) ← مجموعه وابستگی تابعی است که از یک مجموعه وابستگی تابعی به همراه قواعد بدست می آید .

F_{min} (مجموعه وابستگی تابعی مینیم) ← ازروی F_{opt} بدست می آید به صورتیکه قواعدی که قابل استنتاج ازروی قواعد دیگر هستند ، حذف می شوند .

⇐ اگر F یک مجموعه وابستگی تابعی باشد آنگاه برای محاسبه F_{opt} مراحل زیر را انجام می دهیم .

1- سمت راست وابستگی های تابعی را تک عضوی می کنیم . یعنی به صورت زیر عمل می کنیم :

نکته : اگر یک قانون تکراری شد ، تکراری را حذف می کنیم .

$A \to BCD$ ⇒ $A \to B$ ، $A \to C$ ، $A \to D$

2- صفاتی که با حذف آنها تغییری در بستار F ایجاد نکند را از سمت چپ حذف می کنیم .

$AB \to C$ ، $A \to B$ ⟶ در رابطه‌ی اول می توان B را حذف کرد ⇒ $A \to C$

یا

$AB \to C$ ⟶ $B \to C$

$AA \to AB$ ، $A \to AA$ ⟶ $A \to AB$ ، $AB \to C$ ⟶ $A \to C$

3 - قواعد تکراری را حذف می کنیم .

مثال ۱ $R(A, B, C, D, E, F)$

$$F = \{ AB \to CD \;,\; C \to DE \;,\; A \to BE \;,\; EC \to BDF \}$$

$AB \to C$ ① $AB \to D$ ②

$C \to D$ ③ $C \to E$ ④

$A \to B$ ⑤ $A \to E$ ⑥

$EC \to B$ ⑦ $EC \to D$ ⑧ $EC \to F$ ⑨

①,⑤ $\to A \to C$

④,⑧ $\to C \to D$

④,⑤ $\to A \to D$

④,⑨ $\to E \to F$

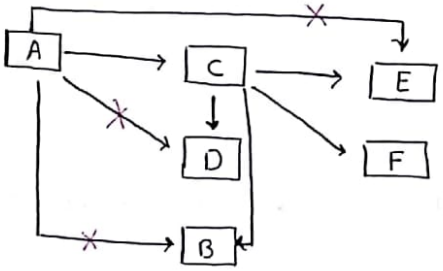
④,⑦ $\to C \to B$

③,⑤ $\to A \to D$

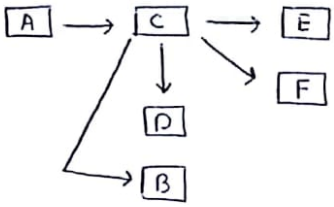
$$F_{opt} = \{ A \to C, \; A \to D, \; C \to D, \; C \to E, \; A \to B, \; A \to E, \; C \to B, \; C \to F \}$$

طبق تعدی

برای پیدا کردن راحت تر خاصیت های تعدی بهتر است نمودار وابستگی تابعی را برای F_{opt} رسم کرده تا به F_{min} برسیم :

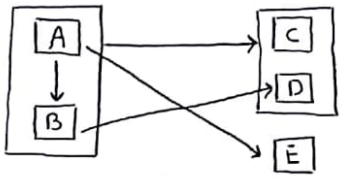


=>



$F_{min} \cong F_{opt} \cong F$

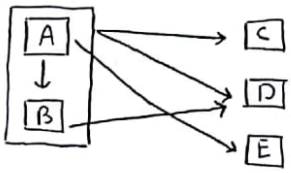
مثال



=>

$AB \to CD$
$A \to E$
$B \to D$
$A \to B$

مرحله‌ی اول) قواعد سمت راست را تک عضوی می کنیم .



=>

$AB \to C$
$AB \to D$
$A \to E$
$B \to D$
$A \to B$

مرحله‌ی دوم) صفت B را از box خارج می کنیم چون با داشتن A می توان به B رسید .

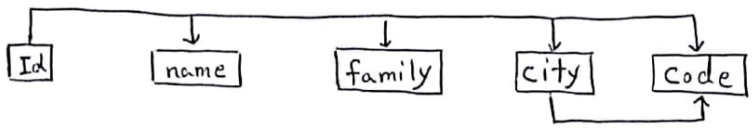
=>

$AB \to C$ } $\to A \to C$
$A \to B$ }
$AB \to D$ $\to A \to D$

$A \to E$
$B \to D$

$\Rightarrow F_{min} = F_{opt}$

مثال) رابطه‌ی R(Id, name, family, city, code) را در نظر بگیرید همانطور که می‌بینید این رابطه بهینه است .



=>

Id → name
Id → family
Id → city
Id → code
city → code

* وابستگی تابعی این رابطه بهینه است (F_{opt}) اما کمینه نیست (F_{min}) .
* وابستگی city → code را می‌توان حذف کرد چون می‌توان آن را از روابط دیگر نتیجه گرفت اما اگر وابستگی code → Id را حذف کنیم رابطه به f_{min} تبدیل می‌شود .

نکته : معمولاً نمودار وابستگی تابعی به صورت یک گراف جهت دار همبند است چون یک کلید داریم که از آن می‌توانیم به همه برسیم .

بستار مجموعه صفات A :

اگر A یک زیرمجموعه از تمامی صفات R باشد بستار A مجموعه تمام صفاتی است که به A وابستگی تابعی دارند و با A^+ نشان می‌دهیم .

$A = \{name, family\}$

$A^+ = \{name, family\}$

بدست آوردن A^+ از روی A :

همه‌ی صفات A را در مجموعه می‌گذاریم چون می‌دانیم که یک وابستگی تابعی بدیهی است . به ازای تمامی وابستگی های تابعی که در F است . ($x \to y$) اگر x زیرمجموعه‌ای از A^+ است . y را به آن اضافه می‌کنیم و این کار را تا زمانی انجام می‌دهیم که A^+ دیگر تغییر نکند .

```
A+ = A
do { for each x → y in F
        if x ⊆ A+ then A+ = A+ ∪ y
} while A+ did not change ;
```

مثال) فرض کنید $R(S, T, U, V, W)$ ، $F = \{S \to T, V \to SW, T \to U\}$.

$A = \{S, V\}$ کلید S, V .

$A^+ = \{S, V, T, W, U\}$

$S \to T$: چون S را در مجموعه داریم تنها T را اضافه می‌کنیم .

$SW \to V$: چون S را در مجموعه داریم و نیز می‌توان در مجموعه عضو تکراری باشد فقط W را اضافه می‌کند .

$B = \{ \}$

$B^+ = \{ \}$

* صفات A و B یک بستار دارند .
* چون از روی ... می توان به همه‌ی صفات رسید پس ... کلید است .

نکته : اگر بستار یک صفت شامل تمامی صفت های رابطه شود ، آن صفت کلید است .

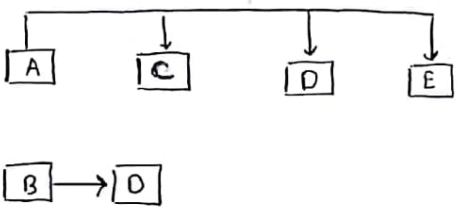
نکته : A کلید رابطه R است اگر و تنها اگر $A \longrightarrow B \quad : \forall B \in \underset{R}{\text{صفات}}$.

این کلید ابر کلید است .

* از بستار برای محاسبه‌ی کلید یک رابطه استفاده می کنیم .

* اگر هیچ کدام از صفات ، بستارش همه‌ی صفات را انداد نگاه می کنیم که اجتماع کدام دو تا همه‌ی صفات را می دهد .

مثال



$A^+ = \{A, C, D, E\}$

$\Rightarrow$

$B^+ = \{B, D\}$

→ کلید : $\{A, B\}^+ = \{A, B, C, D, E\}$

لزوماً کلید منحصر به فرد نیست و ممکن است چندین کلید داشته باشیم . ولی هر رابطه حتماً کلید دارد .

فصل نرمال سازی :

اگر بخواهیم تعداد جداول را مناسب در نظر بگیریم که مشکلاتی از قبیل آنومالی و افزونگی پیش نیاید، از نرمال سازی استفاده می کنیم.

فرض کنید تمام صفات موجودیت و رابطه را در یک جدول در نظر بگیریم.

DB → جدول

ALL

Pname	Pfamily	Sname	S family	SID	Cname	grade	ccode	...
null	null	اکبر	عبدی	3	null	null	null	
ناصر	نوروزی	null	null	null	null	null	null	
		اکبر	عبدی	3	DB	17	3	
		اکبر	عبدی	1	ریاضی	14	2	

معایب این روش :

1- جدول پرشده از مقادیر null و این یعنی برای پیاده سازی این کار به حجم زیادی احتیاج داریم.

2- دارای افزونگی زیاد است.

3- چون افزونگی دارد باعث بروز آنومالی و ناسازگاری می شود (یعنی مثلاً اگر اطلاعات یک نفر در یک جا عوض کردیم باید اطلاعات آن در سایر قسمت ها نیز تغییر پیدا کنیم.

4- ایجاد آنومالی.

* برای برطرف شدن این مشکلات باید جدول بالا به چندین جدول شکسته شود که تعداد این جدول ها می مراحل نرمال سازی بدست می آید.

حالت دیگر این است که تمامی این ستون ها را در جداول جداگانه در نظر بگیریم.

حالت آکادمیک

که در آن جداول را به جدول های کوچکتر تجزیه می کنند در هر مرحله نرمال سازی انجام می شود.

n_1		n_2		n_3		
SId	Sname	SId	S family	SId	city	...

معایب این روش :

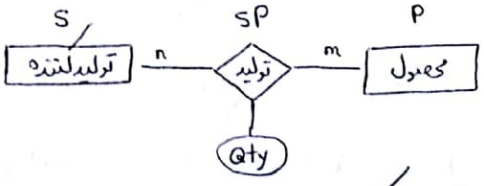
1- پیچیدگی زمانی و حافظه ای زیادی دارد (فضای زیاد برای ذخیره سازی و زمان زیاد برای انجام محاسبات و پاسخ به پرس وجو)

مزایای این روش نسبت به روش قبل :

چون افزونگی ندارد پس ناسازگاری نیز ندارد.

نه روش اول و نه روش دوم کاملاً مناسب نیست و ما چیزی بین این دو می خواهیم.

که برای این کار کافی است به ازای هر موجودیت یک جدول در نظر بگیریم و نیز برای هر رابطه ی n به m نیز یک جدول در نظر می گیریم.



در نرمال سازی ابتدا رابطه را چک می کنیم و آن را به روابط کوچکتر می شکنیم تا بفهمیم که به چند جدول نیاز داریم.

انواع سطوح نرمال سازی :

1- نرمال سطح 1 (1NF) ← first normal form

2- نرمال سطح 2 (2NF) ← Second normal form

3- نرمال سطح 3 (3NF) ← third normal form

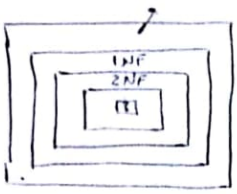
4- نرمال سطح BCNF ← Boyce-codd normal form

5- نرمال سطح 4 ← fourth normal form

6- نرمال سطح 5 ← fifth normal form

7- نرمال سطح BKNF ← Domain-key normal form

(روابطی که 1NF نباشند اصلا رابطه نیستند)



* تاکنون نه ثابت شده که نرمال سطح دیگری داریم و نه اینکه ثابت شده که سطح دیگری نداریم بنابراین در آخرین سطح ؟ داریم .

* تمامی این سطوح زیرمجموعه‌ی هم هستند اما هیچ‌گاه مساوی نیستند و حتماً در یک سطح رابطه‌ای وجود دارد که در دیگر سطوح نیست .

نرمال سطح 1 :

** رابطه R در سطح نرمال 1 قرار دارد اگر تمام صفات آن روی دامنه‌های اتومیک تعریف شده باشند (یعنی تمام صفات آن اتمیک باشد)

اگر و تنها اگر ≡

منظور از صفت اتمیک دو چیز است : 1- تجزیه پذیر نباشد (مثل آدرس) 2- صفت چندمقداری نباشد . (مثل تلفن و مدرک تحصیلی نباشد .)

S

id	name	family	phone
1	اکبر	عبدی	0914-1214 0912-3118

→ این اصلاً رابطه نیست

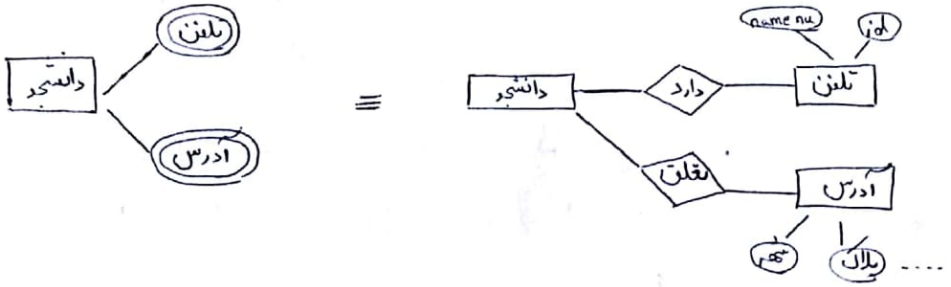
چون ما در رابطه می گفتیم که یک سطر نباید شامل تاپل دیگر باشد .

در اینجا اسم و فامیل و id اتمیک است چون دو ویژگی بالا را دارد اما تلفن این گونه نیست .

البته باز هم به نظر DA بستگی دارد چون ممکن است پرس و جوی خود را روی بخش اول یک اسم انجام شود (مثلاً افرادی را بخواهد که اول اسم آنها محمد باشد) بنابراین در اینجا اسم قابل تجزیه خواهد شد و دیگر اتمیک نیست .

همچنین در مورد آدرس اگر DA بخواهد روی بخش های آن پرس و جو انجام دهد ، آدرس قابل تجزیه می شود ولی در غیر این صورت اتمیک است . در مورد تاریخ نیز به همین صورت است .

** البته همه‌ی اینها در حین پیاده سازی مشخص می شود اما در حالت آکادمیک تمامی این صفات قابل تجزیه است .



برای رفع مشکل چند مقداری بودن و تجزیه پذیر بودن این کار را انجام می دهیم .

برای نرمال سازی در این سطح جدول را با صفات اتمیک آن در یک جا نگه می داریم و صفات غیر اتمیک را در جدول دیگری قرار می دهیم .

(دقت کنید به ازای هر رابطه‌ی اتمیک کلید رابطه‌ی اصلی را با کلید صفت غیر اتمیک در جدول دیگر در نظر می گیریم .)

S

id	name	family

SP

id	phone

SA

sid	city	Ave	Aly	—

اگر رابطه R، 1NF نباشد:

1- رابطه R را به همراه تمام صفات اتمیک آن در رابطه R_1 قرار می دهیم.
2- به ازای هر صفت غیر اتمیک رابطه R مانند A:
الف) صفت A را به همراه کلید رابطه R_1 در رابطه جدید قرار می دهیم.
ب) صفت A را در صورت نیاز به بخش های تشکیل دهنده تجزیه می کنیم.

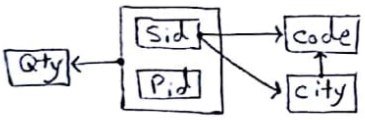
⇐ طریقه ی تبدیل رابطه ی غیر نرمال به رابطه ی نرمال سطح ۱

* از الگوریتم ساده سازی می توانستیم به جای سه جدول در مثال قبل، دو جدول داشته باشیم اما در این حالت مشکل null value پیش می آید چون ممکن است یک دانشجو تلفن داشته باشد ولی آدرس نداشته باشد و برعکس. پس بهتر است همان الگوریتم پیش برویم. در این حالت درست است که تعداد جدول ها بیشتر می شود ولی دیگر مشکل null value را نخواهیم داشت.

* این الگوریتم را تا جایی تکرار می کنیم که رابطه دیگر فیلد غیر اتمیک نداشته باشد.

* فرض کنید رابطه ی SP به صورت SP(sid, Pid, code, city, Qty) باشد، وابستگی آن به صورت زیر باشد. آیا رابطه نرمال سطح ۱ است؟

در حالت کلی رابطه نرمال سطح ۱ است چون تمامی فیلد های آن اتمیک است.
حال جدول زیر را در نظر بگیرید:



Sid	Pid	code	city	Qty	Sname
S_1	P_1	021	تهران	100	فن آوران
S_1	P_2	021	تهران	200	فن آوران
S_1	P_3	021	تهران	300	فن آوران
S_2	P_1	031	اصفهان	150	هوشمند
S_2	P_2	031	اصفهان	200	هوشمند
S_3	P_2	031	کاشان	100	باری

مشاهده می کنیم که درست است که رابطه در سطح ۱ نرمال قرار دارد اما همچنان مشکلات افزونگی و ناسازگاری، آنومالی را داریم که این نشان می دهد که ما به سطح بالاتر از نرمال سازی احتیاج داریم.

* در حقیقت ما چون در اینجا صفت هایی داریم که به کلید وابستگی تابعی کاملی ندارد، افزونگی پیش می آید.
(یعنی می دانیم که (Pid, Sid) کلید اصلی این رابطه است و مثلاً چون city به Sid وابستگی تابعی دارد ($Sid \rightarrow city$) که Sid زیرمجموعه ای از کلید است طبق تعریف وابستگی کامل، city به (Sid, Pid) وابستگی کامل ندارد. $(X \subseteq (Sid), Pid) \Rightarrow city$)

* همانطور که در جدول می بینیم در ستون Qty، افزونگی اطلاعات داریم اما آن را افزونگی در نظر نمی گیریم چون هیچ دو سطر مشابه دارای اطلاعات یکسانی نیستند یعنی مثلاً دو جا 200 داریم اما در یک جا کلید (S_1 و P_2) و در جای دیگر (S_2 و P_2) است که این دو متفاوت است.
پس Qty به کلید وابستگی کامل دارد.

(Sid, Pid) ⇒ Qty ✓
Sid ⟶ Qty ✗
Pid ⟶ Qty ✗ ⟶ هرجا P_2 داریم لزوماً Qty=200 نداریم چون می بینیم که در جایی P_2 داریم درحالیکه Qty = 100 است. ⇐ وابستگی ندارند.

*

در نرمال سطح 2 مشخص می شود که چگونه باید این فیلد های اضافه را پیدا کنیم .

✳ نرمال سطح 2 (2NF) :

↵ رابطه R نرمال سطح 2 است اگر و تنها اگر

1 - رابطه R نرمال سطح 1 باشد (یعنی نرمال سطح 1 زیرمجموعه‌ی نرمال سطح 2 است)

2 - تمامی صفات R به کلید رابطه وابستگی تابعی کامل داشته باشند .

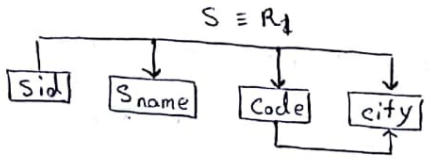
* مثالی که قبلاً آوردیم نرمال سطح 2 بود چون در آن صفاتی وجود دارند که به کلید وابستگی تابعی کامل ندارند .

* اگر یک صفت به کلید وابستگی داشته باشد اما وابستگی کامل نداشته باشد ، رابطه نرمال سطح 2 نیست . ← $A \longrightarrow B$ ✓ , $A \longrightarrow B$ ✗

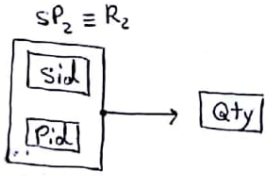
* اگر R نرمال سطح 2 نباشد و بخواهیم آن را نرمال کنیم ، به صورت زیر عمل می کنیم :

1 - آن بخش از کلید که وابسته دارد به همراه تمام وابسته هایش به رابطه جدید R_1 منتقل می شود

به طور مثال ، در مثال قبل



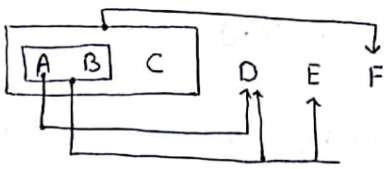
2 - کلید رابطه R را به همراه صفات باقیمانده در رابطه R_2 قرار می دهیم .



3 - در صورت نیاز تکرار الگوریتم برای R_1 , R_2 . (البته مطمئنیم که رابطه‌ی R_2 نرمال است پس برای R_2 نمی‌خواهیم چون ما در مرحله‌ی اول تمامی صفاتی که به بخشی از کلید وابستگی داشتند را از آن خارج کردیم پس تمامی صفات R_2 به کلید وابستگی کامل دارند .)

* اگر یک رابطه داشته باشیم که کلید آن تک صفتی باشد حتماً نرمال سطح 2 است ، دیگر نیازی به تجزیه‌ی آن نیست .

مثال

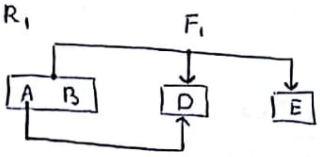


فرض کنید که تمامی صفات اتمیک ، رابطه نرمال سطح 1 باشد

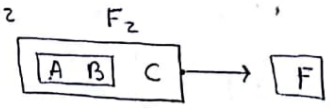
و کلید رابطه ABC است .

می بینیم که رابطه نرمال سطح 2 نیست چون در آن صفات D , E به کلید وابستگی تابعی کامل ندارند .

مرحله‌ی اول →



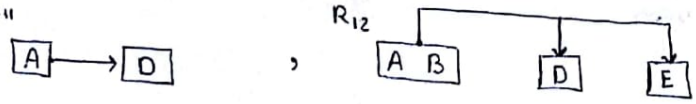
مرحله‌ی دوم →



$F^+ = F_1^+ \cup F_2^+$

↵ در هر سطح از نرمال سازی بستار رابطه باید حفظ شود .

مرحله‌ی سوم →



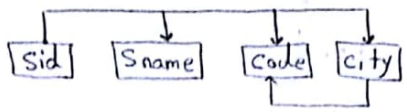
در اینجا کل رابطه‌ی R به سه رابطه‌ی R_2 , R_{11} , R_{12} تجزیه شد که تمامی این روابط نرمال سطح 2 هستند .

وابستگی تابعی با واسطه

فرض کنید یک رابطه‌ی کلی مانند $A \to B \to C$ می دانیم نتیجه این رابطه $A \to C$ است و در اینجا وابستگی تابعی A با C واسطه‌ی B است اما گرچه به ازای تمامی صفات B ، A و C وابستگی تابعی برقرار است و مستقیماً از A به C برسیم وابستگی تابعی مستقیم است.

اگر به ازای هر مقدار متفاوت از A یک مقدار متفاوت از B و خود داشته باشد آنگاه B ، A در رابطه یک همبستگی وابستگی C ، A مستقیم است

مثال ۱



$\Rightarrow$ Sid → city
city → code
$\Rightarrow$ sid → code

وابستگی تابعی با واسطگی به city وجود دارد.

sid	Sname	city	code
S_1	فن آوران	تهران	۰۲۱
S_2	توین تن	تهران	۰۲۱
S_3	هرشمند	اصفهان	۰۳۱
S_4	مینا	کاشان	۰۳۱
S_5	چهارگون	اصفهان	۰۳۱

در جدول مربوط به رابطه‌ی بالا همچنان مشکلات افزونگی و ناسازگاری و آنومالی را داریم.
اگر می توانستیم که به جای اینکه در هر جا که تهران داریم کد ۰۲۱ را تکرار بدهیم در یکجا آن را بگذاریم و در مکان های دیگر به جای آن الله بگذاریم بهتر می شد.

* در واقع مشکل در اینجا وجود وابستگی تابعی بین صفات غیرکلیدی است.
صفاتی که نه کلید هستند و نه جزئی از کلید

رابطه‌ی نرمال سطح 3 (3NF):

رابطه R نرمال سطح 3 است اگر دو شرط زیر

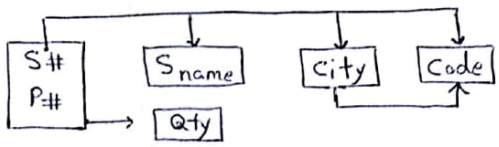
1- رابطه R نرمال سطح 2 باشد.

2- بین صفات غیرکلیدی وابستگی تابعی وجود نداشته باشد - (به صورت غیرمستقیم به کلید وابسته نباشد)
(انتقالی)

اگر رابطه R ، 3NF نباشد آن را طبق الگوریتم زیر تجزیه می کنیم:

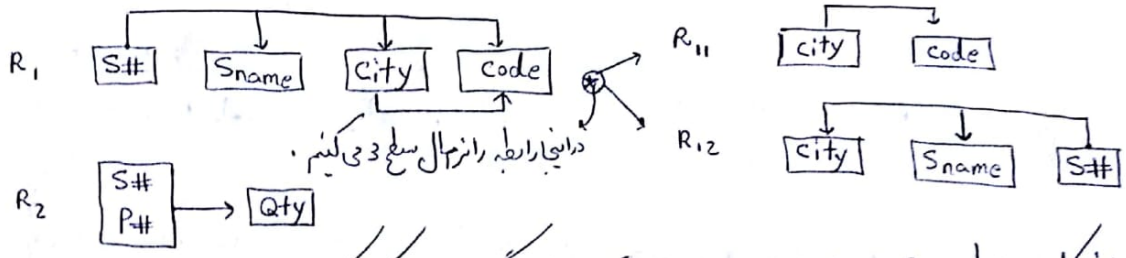
1- آن صفات غیرکلیدی که به یکدیگر وابستگی تابعی دارند را در رابطه‌ی R_1 قرار می دهیم.
2- کلید رابطه R_1 به همراه صفات باقیمانده در رابطه R_2 قرار می گیرد.
3- در صورت نیاز تکرار الگوریتم برای R_1 و R_2

مثال ۱



این مثال در سطح 1NF قرار دارد چون دامنه‌ی تمام $\Rightarrow$ صفات اتمیک است ولی نرمال سطح دوم نیست چون ما گفتیم که در در نرمال سطح دوم تمامی صفات باید به کلید وابستگی داشته باشند درحالیکه در اینجا صفاتی وجود دارد که به بخشی از کلید وابستگی دارند

اول باید رابطه‌ی R_1 را نرمال کنیم 2NF



در اینجا رابطه را نرمال سطح 3 می کنیم.

در اینجا کل سه رابطه‌ی R_{11} و R_{12} و R_2 در 3NF قرار دارند و دیگر نیازی به تکرار الگوریتم نیست.

مشکلات رابطه‌ی اصلی (R): 1- آنومالی درج (چون مثلاً گد شهر را به شهایی نمی توان درج کرد) 2- آنومالی ویرا
4- افزونگی اطلاعات (به ازاء همه‌ی تولید کنندگان گد شهر تکرار می شود)

* در رابطه‌ی نهایی این مشکلات برطرف شد

رابطه های بانک ری ← رابطه هایی هستند که دو صفت بیشتر ندارند که یکی کلید و دیگری صفت است یا در اصطلاح (key-value) هستند.

* تا اینجای کار ما روابطی که نرمال نبودند را از طریق تجزیه به نرمال فرم های مختلف می رسیم اما همیشه این تجزیه جواب نمیدهد و ممکن است جواب های نامرتبط برای ما ایجاد کند.

مثال: R

تولیدکننده (S#)	محصول (P#)	پروژه (J#)
S_1	P_1	J_2
S_1	P_2	J_1
S_2	P_1	J_1
S_1	P_1	J_1

تجزیه جدول →

R_1

S#	P#
S_1	P_1
S_1	P_2
S_2	P_1

,

R_2

P#	J#
P_1	J_2
P_2	J_1
P_1	J_1

تا اینجای کار مشکلی پیش نیامده است. مشکل زمانی پیش می آید که بخواهیم یک سری اطلاعات را از R_1 و R_2 بدست آوریم. مثلاً اطلاعاتی بخواهیم که نیاز باشد دو جدول R_1 و R_2 را با هم پیوند دهیم.

$R_1 \infty R_2$

S#	P#	J#
S_1	P_1	J_2
S_1	P_1	J_1
S_1	P_2	J_1
S_2	P_1	J_1
S_2	P_1	J_2
S_2	P_1	J_1

→ همین طور که می بینیم در اینجا برای ما اطلاعات درست آمده که درست نیست، چون در این جدول 5 سطر داریم در حالیکه در جدول اصلی 4 سطر داشتیم پس داده ی اضافی تولید شده است.
و همچنین در این تجزیه وابستگی J# → S# از بین رفته است.

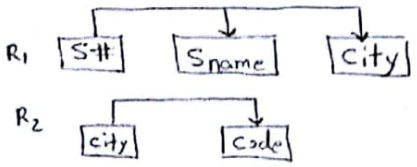
⇐ تجزیه خوب ⇐ تجزیه رابطه R به R_1 و R_2 خوب است اگر

۱- وابستگی تابعی از بین نرود ($F^+ = (F_1 \cup F_2)^+$) ← مثلاً در بالا وابستگی J# به S# از بین رفته بود.

۲- پیوند رابطه R_1 و R_2 سطر اضافی ایجاد نکند.

ضوابط ریسانن: تجزیه خوب است که

۱- ستون مشترک در یکی از رابطه ها کلید باشد. ← یعنی اگر رابطه ی R را به R_1 و R_2 شکستیم و ستون مشترک در یکی از رابطه کلید بود در این صورت تجزیه خوب است: مثلاً



} → اینجا ستون مشترک city است که در R_2 کلید نیز هست → تجزیه ی خوبی داریم.

۲- وابستگی های تابعی حفظ شوند.

قضیه ی هیث Heath:

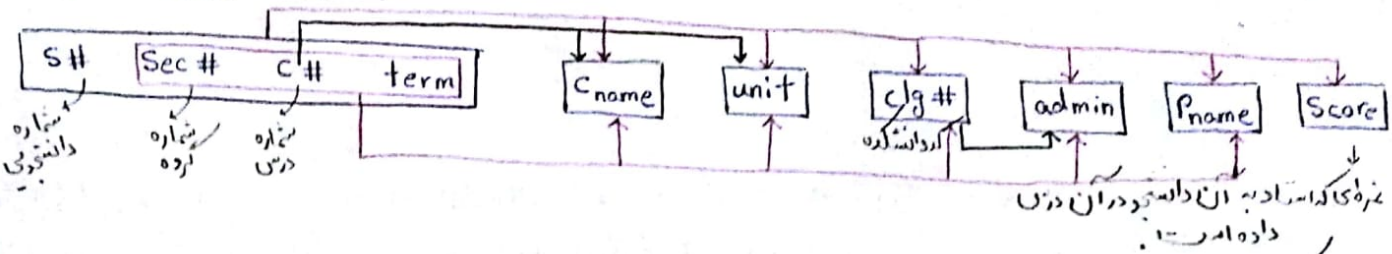
اگر در رابطه $R(A, B, C)$ داشته باشیم $A \to B$ آنگاه تجزیه R به $R_1(A,B)$ و $R_2(A,C)$ تجزیه ای خوب است.

به بیان دیگر:

اگر رابطه $R(A, B, C)$ باشد و داشته باشیم $A \to B$ و $B \to C$ آنگاه تجزیه ی R به $R_1(A,B)$ و $R_2(B,C)$ خوب است.

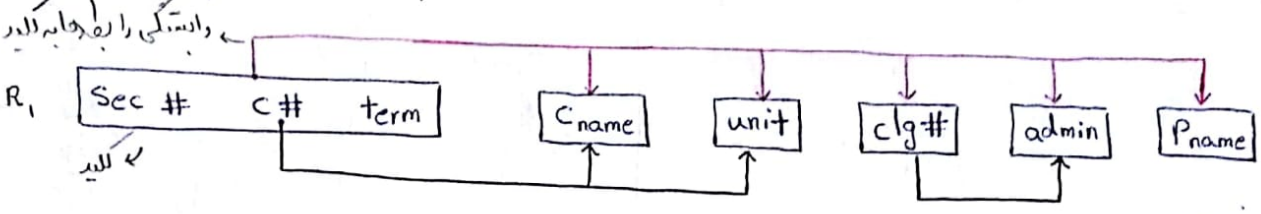
← در اینجا وابستگی تابعی از بین نرفته چون طبق قدری داریم $A \to B$ و $B \to C$ ← $A \to C$ و نیز ستون مشترک B در یکی از رابطه ها کلید است پس تجزیه خوب است.

رابطه‌ی زیر را در نظر بگیرید و نرمال کنید (مثال

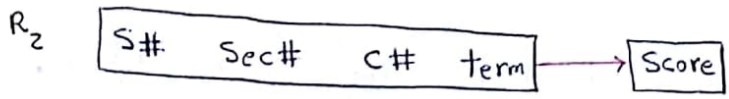


* وابستگی‌های تابعی با نظر DA مشخص می‌شود .
* من کد دانشکده و رئیس دانشکده وابستگی تابعی وجود دارد چون هر دانشکده تنها یک رئیس دارد . (عکس آن برقرار نیست چون ممکن است یک نفر رئیس چند دانشکده باشد .
* این رابطه در نرمال سطح 1 قرار دارد چون تمام صفات آن اتمیک هستند . پس به سراغ نرمال سطح 2 می رویم که باید آن بخش از کلید که وابسته دارد را به همراه تمامی وابسته ها از رابطه‌ی اصلی خارج کنیم (دقت کنید که تمامی وابستگی ها حفظ شود .

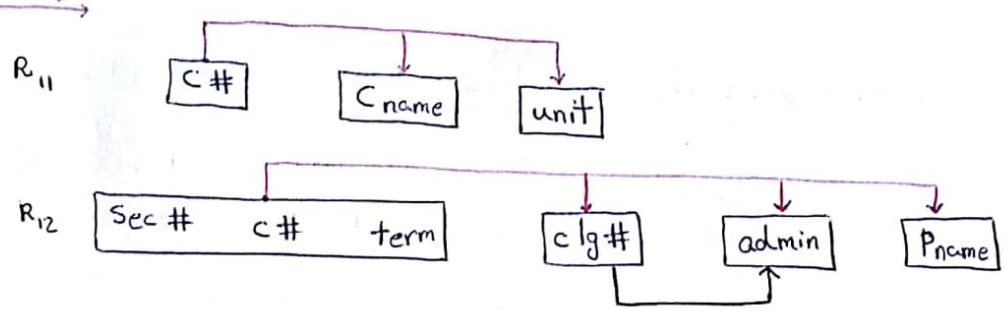
وابستگی را با خط به کلید



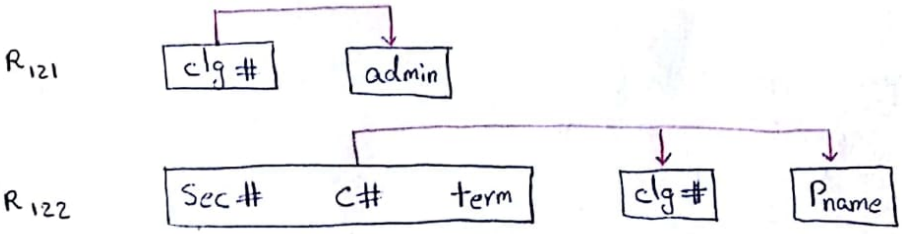
نرمال سطح 2 نیست پس الگوریتم را مجددا تکرار می‌کنیم .



مرحله‌ی دوم



* مشاهده می‌کنیم که سه رابطه‌ی R_2 و R_{11} و R_{12} همگی نرمال سطح 2 هستند چون صفات به وابستگی تابعی ندارند حال باید نرمال سطح 3 را بررسی کنیم که باید صفات غیرکلیدی که به هم وابسته هستند را باید از رابطه خارج کنیم و صفات باقیمانده را به همراه کلید رابطه‌ی قبلی در رابطه‌ی دوم قرار می‌گیرد .



حال همه‌ی این روابط نرمال سطح 3 هستند .

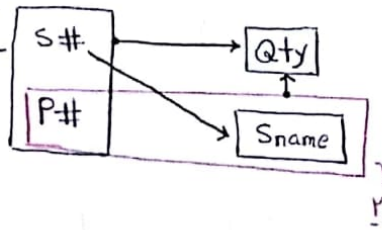
عیب تجزیه کردن روابط در این است که برای بدست آوردن اطلاعات باید جدول ها را با هم join کنیم که این کار هزینه زمان است .
معمولاً تا نرمال سطح 3 اطلاعات تا حدامکان گرفته می‌شود دیگر سراغ نرمال سازی های دیگر نمی رویم

شرایط لازم برای BCNF بودن:

۱- چند کلید کاندید داشته باشد.
۲- کلید کاندید ترکیبی باشد. (کلید کاندید تک صفتی نباشد)
۳- دارای صفت مشترک باشد.

* اگر رابطه‌ای وجود داشته باشد که یکی از شرایط بالا را نداشته باشد ولی 3NF باشد آنگاه BCNF است.
* اگر یک رابطه هر سه شرط بالا را نداشته باشد لزوماً نمی‌توان گفت که BCNF نیست.
* این شرایط، شرایط لازم برای BCNF بودن است اما شرایط کافی نیست.

مثال:



کلید ۱ ← S# P#
کلید ۲ ← Sname P#

چون یک صفت غیرکلیدی بیشتر نداریم.

⇐ این رابطه نرمال سطح 3 است، دارای ۲ کلید است اما دارای افزونگی اطلاعات است مانند

S_1	تورین‌تن	P_1	100
S_1	تورین‌تن	P_2	200
S_2	فن‌آوران	P_1	300
S_2	فن‌آوران	P_2	150

می‌توان ازروی کد به تعداد آن و اسم آن پرسیم و نیازبه نوشتن دوباره آن نیست

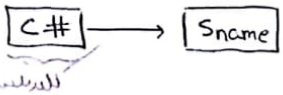
* همانطورکه مشاهده می‌کنیم تمامی مشکلاتی که قبلاً داشتیم یعنی افزونگی و ناسازگاری اطلاعات و آنومالی درج و حذف و ویرایش همچنان باقی است.

ترمینان ← اگر وابستگی تابعی $A \rightarrow B$ کامل باشد به A ترمینان گفته می‌شود.

رابطه‌ی R، BCNF است اگر

در تمام وابستگی‌های تابعی $A \rightarrow B$ در رابطه R، A ترمینان باشد و کلید باشد. (به عبارت خلاصه‌تر اگر هر ترمینان کلید باشد)

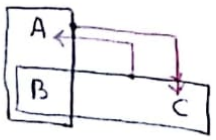
در مثال بالا می‌بینیم که وابستگی تابعی داریم که سمت راست آن کلید نیست بلکه بخشی از کلید است پس رابطه BCNF نیست.



کلید نیست.

نکته ⇐ هر رابطه‌ای که BCNF باشد پس 3NF نیز حتماً هست.

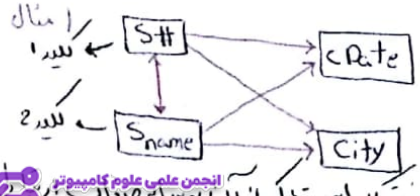
مثال:



→ این یک رابطه BCNF است چون وابستگی‌های تابعی دارند که سمت راست آن کلید است.

⇐ در این مثال می‌توان گفت بدون به چشم بینیم که در اینجا هر سه شرط برقرار است اما با این وجود رابطه BCNF است.

مثال:



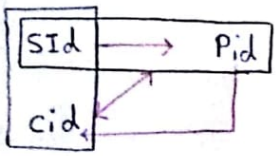
کلید ۱ ← S# cRate
کلید ۲ ← Sname cRate

→ این یک رابطه BCNF است ① چون وابستگی‌های تابعی دارند که سمت راست آن کلید است

② هیچ یک از سه شرط BCNF بودن را ندارد.

★ صفتی کلید است که از آن به سایر صفات وجود داشته باشد

مثال



این رابطه BCNF نیست
→ Pid → cid چون وابستگی تابعی
در آن وجود دارد که سمت چپ آن کلید نیست.

تجزیه

$R_1(Sid, Pid)$
$R_2(sid, cid)$
→ BCNF

* در اینجا R_1 و R_2 باینری اند پس رابطه BCNF می‌شود اما وابستگی تابعی $(cid, sid) \rightarrow Pid$ از بین رفته پس این تجزیه یک تجزیه خوبی نیست.

* هر رابطه تمام کلید یا باینری، BCNF است.

↳ رابطه تمام کلید، رابطه‌ای است که تنها یک کلید دارد و آن کلید مجموعه‌ی همه‌ی صفات است. هم‌چنین در این رابطه وابستگی تابعی غیر بدیهی وجود ندارد پس BCNF است.

↳ رابطه باینری، رابطه‌ای است که دو صفت دارد حال اگر دو صفتش کلید باشند، تمام کلید است پس BCNF است.

وابستگی تابعی بدیهی:

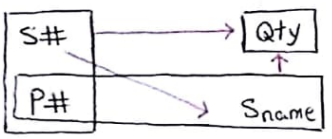
$A \rightarrow B$ وابستگی تابعی بدیهی است اگر $B \subseteq A$ باشد.

صفات عمده:

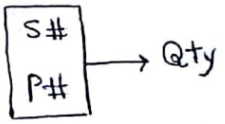
صفاتی که بخشی از کلید باشد.

* در صفات عمده اگر صفتی وجود داشته باشد که وابستگی تابعی کامل به کلید نداشته باشد را به هنگام BCNF کردن یک رابطه، از آن خارج می‌کنیم.

↳ به طور مثال در مثال قبل داریم:



=> $S\# \rightarrow Sname$ وابستگی را خارج می‌کنیم →



سوال: اگر یک رابطه قابل تجزیه به دو رابطه R_1 و R_2 بود. آیا همیشه تجزیه کردن باعث نرمال سازی می‌شود؟

خیر. اگر باعث کاهش سطح نرمال سازی شود، تجزیه می‌کنیم در غیر این صورت تجزیه کردن آن رابطه یک هزینه اضافی برای پیوندجدول‌ها برای بدست آوردن سؤالات ما خواهند داشت. همچنین باید دقت کنیم که با تجزیه کردن وابستگی تابعی از بین نروند.

نرمال سازی 4NF:

P	C	B
آسیدی	DB	رانلوفی
اسعدی	DB	معتمی
نوروزی	DB	رانکوهی
نوروزی	DB	معتمی
قدیری	میانجا	D&D
	میانجا	جعفرنژاد
	میانجا	جعفری

مثال رو به رو مشخص می‌شود که چه استادی چه درسی را با چه کتابی درس می‌دهد.

این رابطه یک رابطه‌ی تمام کلید است چون هیچ کدام از صفات به تنهایی کلید نیستند همچنین هیچ دو صفتی نیز با هم کلید نیستند. اما مجموعه‌ی همه‌ی صفات با هم می‌توانند کلید باشند و سطر تکراری در جدول تولید نمی‌کنند.

* این رابطه، یک رابطه تمام کلید است پس BCNF است.

* این رابطه 3NF نیز هست .

* مشکلات این رابطه :

۱- افزونگی اطلاعات دارد ← اسم استاد و اسم درس تکرار می شود .

2- آنومالی داریم ← یعنی نمی توان اسم یک درس را به تنها اضافه کرد و حتماً باید یک استاد آن درس را تدریس کند . برای کتاب نیز همینطور است .

یعنی نمی توانیم سطر سیلور شات OS null را اضافه کنیم یا نمی توانیم مشخص کنیم منبعی که میاد لیر تدریس می کند چون حتماً باید منبع داشته باشیم . ← آنومالی درج

آنومالی حذف و ویرایش نیز داریم .

مثلاً اگر منبعی مبانی DBD را حذف کنیم ممکن است اینکه DBD یک منبع برای مبانی است نیز از بین برود و دیگر از آن استفاده نکرده ایم .

وابستگی تابعی چند مقداری : (MVFD)

اگر A و B دو مجموعه صفات از رابطه R باشد ، B به A وابستگی تابعی چند مقداری دارد اگر در طول حیات رابطه R به ازای هر مقدار از صفت A تنها یک مجموعه متناهی مقدار متناظر از صفت B موجود باشد . $A \twoheadrightarrow B$

مثلاً در مثال قبل $B \twoheadrightarrow C$ به ازای هر درس چندین منبع وجود دارد .

$C \twoheadrightarrow P$ به ازای هر درس چندین استاد می تواند وجود داشته باشد .

* اگر مجموعه ی B تک عضوی باشد . $A \twoheadrightarrow B$ معادل $A \rightarrow B$ (وابستگی تابعی) است .

پس هر وابستگی تابعی ، یک وابستگی تابعی چند مقداری است .

در مثال قبل وابستگی تابعی نداریم چون رابطه تمام کلید است ! اما وابستگی تابعی چند مقداری دارد .

4NF :

رابطه R ، 4NF است اگر به ازای هر وابستگی تابعی چند مقداری غیر بدیهی $A \twoheadrightarrow B$ صفت A ابرکلید باشد .

مثلاً پس رابطه ی مثال قبل 4NF نیست چون وابستگی های تابعی چند مقداری $B \twoheadrightarrow C$ و $C \twoheadrightarrow P$ را داریم که در آن C ابرکلید نیست .

* اگر $R(A,B,C)$ را داشته باشیم و وابستگی تابعی چند مقداری $A \twoheadrightarrow B$ را داشته باشیم ، نتیجه می گیریم $A \twoheadrightarrow B$ و معمولاً نتیجه می گیریم $A \twoheadrightarrow BC$

* هیچ رابطه ای نداریم که یک وابستگی تابعی چند مقداری داشته باشد . معمولاً دو تایی اند یکی خود وابستگی تابعی و یکی مکمل آن

مثال ← $ABCDE$ و $A \twoheadrightarrow B$ داریم ← نتیجه $A \twoheadrightarrow CDE$

رابطه ای که 4NF نیست وابستگی تابعی چند مقداری $A \twoheadrightarrow B$ را دارد که A کلید نیست .

برای 4NF کردن A,B را جدا کرده و در یک رابطه و A و بقیه صفات را در رابطه ی دیگر قرار می دهیم .

$R_1(A,B)$ $R_2(A, R - \{A,B\})$

* رابطه PCB تجزیه می‌شود به $C \to\to P$ و $C \to\to B$ که 4NF اند.

نکته ← در روابط باینری وابستگی تابعی چندمقداری نداریم. (اگر بخواهیم غیربدیهی‌ها را در نظر بگیریم)

مثال) اگر $R(A,B,C,D,E,F)$ که $A \to\to B$ ، $C \to\to D$ ، R ، 4NF باشد.

$R_1(A,B)$

$R_2(A,C,D,E,F)$ ↗ $R_{21}(C,D)$

$R_2(A,C,D,E,F)$ ↘ $R_{22}(C,A,E,F)$

✳ قضیه‌ی نالین : Fagin

رابطه $R(A,B,C)$ می‌تواند به دو صورت $R_1(A,B)$ و $R_2(A,C)$ تجزیه شود اگر و تنها اگر $A \to\to B$

← این قضیه فراتر از هیث است و آن را در بر می‌گیرد. البته با فرض $A \to B$ برابر $A \to\to B$.

بنابراین ← در رابطه R اگر داشته باشیم $A \to\to B$ ، $B \to C$ رابطه R را به صورت $R_1(A,B)$ ، $R_2(B,C)$ تجزیه می‌کنیم.

طبق این تعریف هر رابطه 4NF ، BCNF نیز هست اما عکس آن برقرار نیست.

✳ وابستگی تابعی چندمقداری بدیهی :

اگر داشته باشیم BCA داریم $A \to\to B$

خواص وابستگی تابعی چندمقداری :

- $AC \to\to BC$: $A \to B$
- $A \to\to C$: $B \to C$ ، $A \to\to B$
- $A \to\to R-A-B$: $A \to\to B$
- $A \to\to B$: $A \to B$
- $A \to\to BC$: $A \to\to B$ ، $A \to\to C$

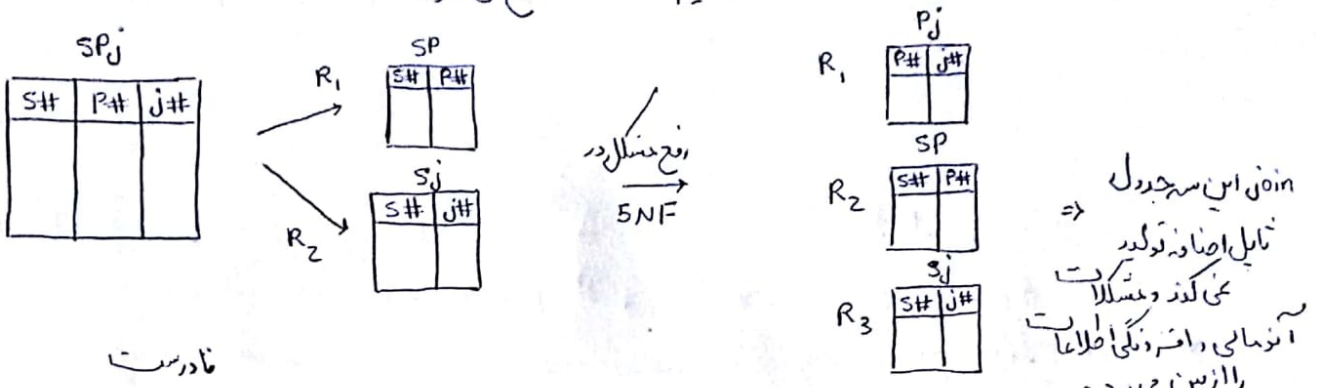
* برای اینکه یک رابطه را نرمال کنیم ابتدا باید F_{opt} آن را بدست آوریم.

* در این روابط همین که $A \to\to B$ را داشته باشیم که سمت چپش کلید نباشد، باید از رابطه خارجش کنیم و نرمال کنیم.

نکته ← تا جایی که ممکن است سعی می‌کنیم زیاد جدول‌ها را نشکنیم چون هزینه join بالا می‌رود.

5NF : شامل رابطه هایی هستند که نمی توان آن ها را به دو رابطه تجزیه کنیم و همچنان مشکلات قبل (آنومالی و ...) وجود داشته باشد.
در واقع رابطه R را باید به بیش از دو رابطه تجزیه کرد. ($n > 2$) باید آنقدر تجزیه کنیم که دیگر آن مشکلات در هیچ یک از n رابطه وجود نداشته باشد.

مثال : رابطه ی زیر را در نظر بگیرید که قبلاً دیدیم که اگر آن را به دو جدول SP و Sj تجزیه کنیم، تاپل اضافه تولید می کند که این مشکل ایجاد می کند از طرف دیگر می توان آن را به هیچ دو رابطه ی دیگری تجزیه کرد که مشکل در آن وجود نداشته باشد پس یعنی این رابطه یک 5NF است، و اگر بیاییم آن را به بیش از دو رابطه تجزیه کنیم مشکلات رفع می شود.



⇐ تعریف کلی 5NF :

اگر رابطه R به n رابطه R_n و ... و R_2 و R_1 قابل تجزیه باشد و به هیچ $n-1$ رابطه ای قابل تجزیه نباشد می گوییم رابطه R وابستگی پیوندی دارد به R_n و ... و R_2 و R_1.

$R = *(R_1, R_2, \dots, R_n)$

به R_n و ... و R_2 و R_1 پرتوهای R گفته می شود.

* در مثال بالا طراحی اطلاعات به صورت جدول SPJ نادرست است چون از طرفی افزونگی اطلاعات داریم و از طرف دیگر مشکل آنومالی در درج و حذف در آن وجود دارد در حالیکه اگر آن را به سه جدول تجزیه کنیم همه ی این مشکلات برطرف می شود.

* رابطه های BCNF که تمامی صفت های آن ساده باشند را می توان تا سطح 5NF نرمال سازی کرد.

* هر رابطه ای که وابستگی پیوندی داشته باشد را حتماً باید تجزیه کرد تا مشکلات آن برطرف شود.

* رابطه های تمام کلیدی که وابستگی چند مقداری (MVFD) در آن وجود نداشته باشد 4NF است.

سؤال) آیا هر رابطه ای که وابستگی پیوندی داشت را حتماً باید به بیش از دو رابطه تجزیه کرد؟؟

برای مثال رابطه ی زیر را در نظر بگیرید :

R(S# Sname city Date)

می بینیم که درست است که می توان رابطه ی R را به سه رابطه تجزیه کرد اما در اینجا تجزیه مناسب نیست چون سطح نرمال سازی را افزایش نمی دهد و فقط یک هزینه ی اضافه برای join جداول فراهم می کند.

R_1(S# city)
R_2(S# Sname)
R_3(S# Date)

تجزیه R به این سه رابطه طبیعی هیس انجام می شود.

بنابراین اگر در رابطه‌ی R وابستگی پیوندی بین $R_1, \dots, R_n$ وجود داشته باشد ولی کلید در هیچ یک از $R_1, \dots, R_n$ وجود نداشته باشد پس باید رابطه‌ی R را به بیش از 2 رابطه تجزیه کرد چون دیگر 5NF نیست .

- رابطه R در سطح 5NF است اگر :
درصورتی که وابستگی پیوندی به پرتوهایی داشت کلید رابطه R در تمام پرتوها موجود باشد .

* هر رابطه باینری تا سطح 5NF نرمال است چون در آن وابستگی پیوندی در آن موجود نیست .

* در مثال رابطه SP_1 کلید رابطه ($S\#, P\#, \#_1$) است که در هیچ یک از پرتوهای آن موجود نیست و این یعنی SP_1 ، 5NF نیست و باید آن را نرمال سازی کنیم .

خلاصه :

BCNF : رابطه R در سطح BCNF است اگر هر FD در آن توسط کلید کاندید مطرح شود . ($A \Rightarrow B$ ، A کلید)

4NF : رابطه R در سطح 4NF است اگر هر MVFD در آن توسط کلید کاندید مطرح شود . ($A \not\rightarrow\!\!\!\rightarrow B$ ، A کلید)

5NF : رابطه R در سطح 5NF است اگر هر JD در آن توسط کلید کاندید مطرح شود ($R = *(R_1, \dots, R_n)$ ، $\forall i\ R_i$ کلید R در آن باشد)
وابستگی پیوندی

* در 5NF الگوریتمی برای تشخیص آن نداریم و باید به حالت Broat force تمامی حالات ممکن را در آن چک کنیم .

- مزایای نرمال سازی :
1 - کاهش آنومالی
2 - کاهش افزونگی
3 - ساده‌سازی اعمال تواعد جامعیت چون روابط را به رابطه های کوچکتر تجزیه می کنیم .
4 - طراحی استاندارد

مثلاً اگر یک DB را بسازیم و آن را نرمال سازی کنیم در نهایت به یک شکل استاندارد و یکسانی می رسیم . این سبب می شود که اگر برای DB مشکلی پیش آمد و خواستیم آن را به شرکت دیگری بدهیم که آن را رفع کند ، آن شرکت با DB پیچیده‌ای روبه رو نباشد چون ما DB را مطابق با اصول نرمال و استاندارد آن را طراحی کرده ایم .

- معایب نرمال سازی :
1 - سربار اضافی بابت پیوند جدول ها (چون نیاز داریم برای انجام محاسبات جدول ها را پیوند دهیم که اینکار یک هزینه اضافی است)
2 - مقداری افزونگی (افزونگی تکنیکی) ← افزونگی حاصل از وجود کلید خارجی
* افزونگی در این حالت کمتر از حالتی است که نرمال نکرده ایم .
چون گفتیم که در صورتی نرمال می کنیم که یک وابستگی تابعی حفظ شود ، دوم اینکه طبق هدیس ، مست مشترک در آن کلید باشد و به همین دلیل اگر اطلاعات در رابطه‌ی R یکبار تکرار شود در R_1 و R_2 دوبار تکرار می شود .
3 - پیچیدگی نرمال سازی در رابطه های بزرگ .
4 - نرمال سازی وابسته به مجموعه وابستگی های تابعی بهینه است .
* فقط سطح اول است که وابستگی تابعی در آن ندارد .

۴- (۴) نرمال سازی وابسته به مجموعه وابستگی های تابعی بهینه است.
نرمال های سطح ۲ به بعد همگی به وابستگی های تابعی نیاز دارند.
ممکن است برای رابطه های بزرگ متخصص بیرون باشد رابطه ها متخصص بیرون نیست.
هرچند بعضی شرکت ها خودشان اقدام به تشکیل اطلاعات می دهند که خاص باشد (متخصص بیرون)
برای اینکه داده ها در اختیار خود شرکت یا سازمان دیگری نباشد. DB را کد می کنند
و برای ... دی که برگردن پول می گیرن ...
هرچند که DB باید در اختیار سازمان باشد و این کار غیر احتمالی است.

مباحث تکمیلی SQL: table value functions خروجی آن ها جدول است
scaler " " خروجی آن ها اسکالر (عدد و کاراکتر...)
در حقیقت یک جدول 1x1 است اما به دلیل کاربرد زیاد جدا در نظر گرفته می شود

قسمت های پیروژه:
۱- Function
۲- View
۳- Procedure
۴- ConStrain

در SQL می توانیم function تعریف کنیم مثل برنامه نویسی.

(۳): مجموعه دستوراتی که یک کار را برای ما انجام می دهند شباهت زیادی با function دارند.
در اوراکل ... pro ... اما در SQL بیشتر store
در ... pro ... می توانیم در جدول insert کنیم اما در function نمی توانیم.
منظور از دستوراتی بزرگ ... store pro ... هایت.

function: یک سری ورودی دارد که روی آن ها پردازش می کند و یک خروجی می دهد.

تعریف Schema function:

create function نام (@ نام پارامتر1 نوع 1 ، @ نام پارامتر2 نوع 2 ...)
returns table ← نوع خروجی
ادامه صفحه بعد

در SQL متغیری که با @ مشخص شده باشد به این معنی است که پارامتر است.
بعد از آن یک نوع قرار می گیرد و نوع خروجی را نشان می دهد حتماً باید returns داشته باشد.

[As]

[begin] اختیاری

return (select_query) کوئری که جدول مدنظر را برمی گرداند

[End] اختیاری

return چون کلمه کلیدی returns وقتی می خواهیم نوع خروجی را مشخص کنیم حتماً بنویسیم
هم داریم اما کار دیگری برایمان انجام می دهد.

در table value fu- ها begin, End کلمه کلیدی داریم اما در
Scaler ها ~ ~ نداریم.
کلمه کلیدی As در هر دو اختیاری است.

```
Create function student From (@city nvarchar(50))
    returns table
     As
    return(
        select * from Student
          where city = @city
    )
```

این تابع یک شهر می گیرد و تمام ستون های جدول student که شهرشان city است را برمی گرداند.

```
select name
from student
```

```
select name from
student From('تهران')
```

تابع

این ستون حتماً باید در خروجی تابع باشد.

پس بنابراین در کوئری می توانیم جلوی from هم می توان view هم sub qurey هم table fu- برگرداند

Scaler value :

Create function نام (@نام پارامتر و نوع , ...)
returns نوع خروجی
[AS]
Begin
Body
End

هرچند تا متغیر که بخواهیم می توانیم تعریف کنیم

هرجا که نیاز به متغیر داشته باشیم با استفاده از @ تعریف می کنیم.

ورودی تابع می تواند یک کوئری نیز باشد. ← اما جواب آن باید مثلاً اینجا ورودی تابع یک String است اگر بنویسیم Select City from prof یک جدول است ← خطا

اما اگر بنویسیم select City from prof where id=1 اسامی دانشجویان همشهری با استاد 1 را می دهد.

Begin , End ← بدنه را مشخص می کند.

در table value ما میان ( ) بدنه را مشخص می کند.

Body : ممکن است در Body نیاز به تعریف متغیر داشته باشیم برای تعریف متغیر در Body متغیر از هر نوعی باشد از declare استفاده می کنیم یعنی داریم یک متغیر تعریف می کنیم.

declare @نام نوع = مقدار

و نداریم هم خطا می دهد ← اختلاف بزرگ

یا حتی اگر کلمات کلیدی را کامل بنویسیم می فهمد.

مقدار دهی اولیه متغیر دلخواه است می توانیم مقدار دهی اولیه نکنیم. مقدار اولیه ی دهیم مجهول است این

تابعی بنویسیم که بیشترین نمره دانشجو را برگرداند ← یک مقدار را برمی گرداند ← خروجی یک مقدار

decimal(4,2) یعنی 4 رقم می گیرد و رقم آن اعشار است

```
create function maxscore (@Id int)
returns decimal (4,2)
Begin
declare @x decimal(4,2);
select Max(score) into @x
from cs where sid = @Id
return @x;
End
```

متغیر از همان نوع

مقدار این را در x بریز

مثال:

```
insert into table2
(select * from table1)
```

تمام اطلاعات جدول 1 را در جدول 2 قرار می‌دهد.

کار تابع: یک Id از ورودی می‌گیرد. یک متغیر x تعریف می‌کند بیشترین نمره Id ورودی را در x می‌ریزد و x را برمی‌گرداند.

```
declare @x int
set @x = 4
set @ = (select ——)
```

نوع Select و x یکی باشد

```
select id, name, family, maxscore(id)
from (student)
```

جدولی که 4 ستون دارد. نام و نام خانوادگی و id بیشترین نمره هر دانشجو.

بدون رجوع به cs بیشترین نمره را داریم. در حقیقت این تابع بیشترین نمره را از cs برایمان می‌آورد.

معمولاً کاربرد زیادی در بیشترین‌ها، کمترین‌ها، مجموع‌ها دارد.

sclare value — فقط جلوی select می‌توانند بیایند جلوی from نمی‌آیند.

در اوراکل برای Begin می‌توانیم متغیر تعریف کنیم بدون declare و در بدنه نیز قابل دستیابی است

Arman

```
create function std count (@city nvarchar(50))
returns int
Begin
declare @x int;
select count(*) into @x
from student where city = @city
    return @x;
End
```

این تابع می‌گیرد و تعداد دانشجویان آن شهر را برمی‌گرداند.

```
Select city , std count(city)
    from student
```

مثلاً بر روی city group by می‌کنیم.

```
create Procedure fi
@fi نوع ,
@fi نوع
;
As
[Begin]
    Body
[End]
```

: procedure

پارامترهای ورودی function مثل ( ) برمی‌گرداند

ترتیب ستون‌های جلوی select مهم نیست اما وقتی insert می‌کنیم مهم است.

```
insert into table1 (name , family)
    Select name ,family
        fromtable 2
```

```
Create procedure Insert_cs
  @sid int,
  @cid int,
  @score decimal(4,2)
  As
  insert into cs(sid,cid,score)
        values(@sid,@cid,@score)
```

این سطر یا جدول cs اضافه می‌کند.

مهم اینجا و مهم function ترتیب پارامترها و ورودی‌ها مهم است.

یک procedure ممکن است چندین Insert داشته باشد.
مثلاً دانشجو ← یک سری مشخصات ۲. person و یک سری مشخصات ۲. S.

procedure ویرایش نمره:

```
Create procedure update_cs
  @sid int,
  @cid int,
  @score decimal(4,2)
  As
  update cs
   Set score=@score
   where sid=@sid and cid=@cid
```

آن جایی که sid و cid آن را گرفته نمره آن را ویرایش می‌کند.

نحوه اجرای procedure:

Exec

Execute اسم پروسیجر ورودی 1 و ورودی 2 و ..... ترتیب ورودی‌ها مهم است

Exec update_cs 1,2,18 برای دانشجوی 1 درس 2 نمره 18 میزاره

Arman

Subject:
Year Month Day page: ( )

می خواهیم یک procedure بنویسیم که این اطلاعات را به جدول ها اضافه کند.

S

S1	تهران	آبکین

SP

S1	P1	300

```
Create procedure Add_S
@sid varchar(5), @name nvarchar(50), @city nvarchar(20)
,@pid varchar(5), @qty int
AS
insert into S(id, name, city)
    values(@sid,@name,@city)
insert into SP( sid,pid,qty)
    values(@sid,@pid,@qty)

exec Add_S 'S10', 'آبکین', 'تهران', 'P1', 300
```