اهداف ایجاد پایگاه داده
- **هدف اولیه:** ایجاد فضایی برای ذخیره‌سازی داده‌ها با
  - امنیت $(Security)$
  - جامعیت $(Integrity)$
  - کمترین میزان افزونگی
  - تحت یک کنترل متمرکز
- **هدف ثانویه:** پردازش داده‌ها $(Data\ Processing)$ و تولید اطلاعات $(Information)$

تکثیرسازی (Repilication): جنبه خوب افزونگی

روشهای ذخیره و بازیابی اطلاعات
- **روش غیربانکی** ≡ روش کلاسیک ≡ روش فایل‌های متعارف ≡ روش پرونده‌ای
- **روش بانکی** ≡ رویکرد بانکی ≡ DBMS (Database Management System)

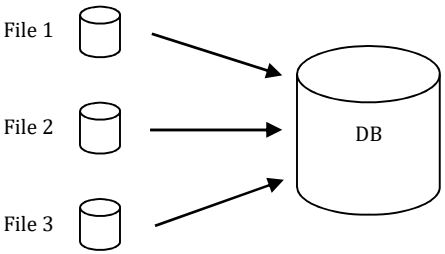
در روش غیربانکی
۱. هرکدام از بخش‌های محیط عملیاتی یک فایل جداگانه ایجاد می‌کند که این فایل‌ها با هم $Join$ نیستند.
۲. هرکدام از بخش‌های محیط عملیاتی با امکانات مجمود در زبان سطح بالا یک App ایجاد می کند.
۳. وظیفه ذخیره و بازیابی برعهده بخشی بنام $File\ System$ است.

روش غیربانکی
- **مزایا:** امنیت فیزیکی داده‌ها بالاست (البته در حال حاضر دیگر مزیت محسوب نمی شود)
- **معایب:**
  ۱. افزونگی داده‌ها بسیار بالا است.
  ۲. ناسازگاری داده‌ها $(Data\ Inconsistancy)$ بالا است.

ناسازگاری داده‌ها: زمانی ایجاد می شود که یک داده ی مشابه، در چند محل موجود است و آن محل ها به هم Join نیستند، اگر تغییر در یکی از آنها ایجاد شود این تغییر همزمان در مکان های دیگر اعمال نمی شود.

روش بانکی: اولین پیامد کاهش افزونگی است. (افزونگی هیچگاه کاملا حذف نمی شود.)

$$\overbrace{Field}^{\text{کوچکترین جز}} \rightarrow \overbrace{Record}^{\text{فیلدهای هم خانواده}} \rightarrow \overbrace{Table}^{\text{مجموعه رکوردها}} \xrightarrow{\text{ارتباط}} Database$$



گروهی از فایلها که به نوعی با هم ارتباط دارند

تعریف DBMS: یک **سامانه مدیریت پایگاه داده** به صورت نرم‌افزاری، شامل کلیه مولفه های لازم برای **مدیریت و راهبری** یک پایگاه داده است. در واقع همانند **قوه مجریه** عمل می کند یعنی هرگاه صحبت از اجرا شد وظیفه DBMS می باشد.

مدیریت و راهبری پایگاه داده توسط DBMS شامل
- برقراری امنیت
- اعمال محدودیت‌های جامعیتی
- انجام جستجوی سریع
- کنترل دسترسی های همروند $(Concurrent)$
- جلوگیری از بروز پدیده ناسازگاری داده‌ها

اهداف و مزایای یک پایگاه داده:
- کاهش افزونگی (هیچگاه حذف نمی شود) $(Reduce\ Redundancy)$
- امنیت اعمال $(Apply\ Security)$
- جامعیت حفظ $(Maintain\ Integrity)$
- کنترل همروندی $(Avoid\ Inconsistancy)$
- جلوگیری از بروز پدیده ناسازگاری داده‌ها $(Concurrent\ Control)$

جایگاه افراد در ارتباط با پایگاه داده:
- $DA\ (\boldsymbol{D}ata\ \boldsymbol{A}dministrator)$
- $DBA\ (\boldsymbol{D}ata\boldsymbol{b}ase\ \boldsymbol{A}dministrator)$
- $DBP\ (\boldsymbol{D}ata\boldsymbol{b}ase\ \boldsymbol{P}rogrammer)$

یا براساس تقسیم‌بندی دیگر
- $End\ User$
- DBA
- DBP

تعاریف:
- $End\ User$ کاربرانی که از خدمات سیستم استفاده می کنند ولی فنی نیستند.
- DA فاقد مهارت‌های تخصصی در پایگاه داده است و مسئولیت
  - مدیریت
  - تفسیر
  - طبقه بندی
  - و سازمانده

  داده‌ها، مستقل از کامپیوتر را دارد.
- DBA
  - **قبل از پیاده‌سازی:** ایجاد یک شمای مصنوعی (تصویر کلی) (Conceptual Schema) با ابزارهایی بنام ER / EER
  - **بعد از پیاده‌سازی:**
    - تعریف محدودیت‌های امنیتی
    - تعریف محدودیت‌های جامعیتی
    - ارزیابی بانک اطلاعاتی
    - ایجاد یک نقشه برای بک آپ گیری
- DPA اپلیکیشن طراحی و تولید می کند.

واژه ها و مفاهیم بنیادی:

تراکنش (Transaction): موفق $(commit)$ / ناموفق $(abort)$

هر کاری که توسط کاربر در محیط پایگاه داده انجام می شود یک **تراکنش** است و تراکنش **یک واحد منطقی** کار است.

$$Transacion \rightarrow DBMS \rightarrow OS$$
$$Task \rightarrow OS$$

یک تراکنش شامل چندین عملیات بانکی است که یا باید **همه دستورات** آن اجرا شود یا **هیچکدام** اجرا نشود.

جامعیت (Integrity): یعنی بی نقصی و اینکه تراکنش‌ها از قوانین تعریف شده در سیستم پیروی کنند.

کنترل های چهارگانه ACID: برخی از قوانین جامعیت، **قوانین عام (Meta Rule)** هستند که توسط DBMS اعمال می شوند، برای این منظور در حوزه پایگاه داده **کنترل های ACID** معرفی شدند.

ACID
- **یکپارچگی** $(Atomicity)$ کنترل همه یا هیچ یعنی همه دستورات اجرا شوند یا هیچکدام اجرا نشوند.
- همخوانی $(Consistancy)$ تراکنش‌ها باید از تمام قوانین تعریف شده در سیستم پیروی کنندو استثنا نداریم.
- انزوا $(Isolation)$
  - کنترل همروندی تراکنش‌ها بررسی می‌شود تا اثر تخریب کننده روی هم نداشته باشند.
  - یکی از روش‌های این کنترل $Lock$ می باشد.
    - مزیت آن افزایش سرعت و
    - مشکل آن ایجاد پدیده بن‌بست $(Dead\ Lock)$ است
- ماندگاری $(Durability)$ تراکنش‌هایی که انجام می شوند، اثرشان ماندنی است و هرگز بصورت تصادفی از بین نمی روند.

$$ACID \rightarrow DBMS \begin{cases} Recovery\ Management: A, D \\ Concurrency\ Control: I \end{cases}$$

معماری بانک اطلاعاتی (ANSI SPARC):

چگونگی ایجاد **ارتباط منطقی** بین کاربران (بیرونی ترین سطح) و **داده‌های ذخیره شده روی دیسک** (داخلی ترین سطح)

معماری ۳ لایه ANSI: لایه‌ها مستقل از یکدیگر هستند.

$ANSI\_SPARC$
1. $External\ Level\ (View\ Level)$
2. $Conceptual\ Level$
   - $Conceptual\ Design \equiv$ طراحی مفهومی (ادراکی) $\equiv$ ادراکی عام $\rightarrow$ ابزار (ER, EER)
   - $Logical\ Design \equiv$ طراحی منطقی $\equiv$ ادراکی خاص $\rightarrow$ مدل‌ها
3. $Internal\ Level\ (Physical\ Level) \rightarrow index, cluster$

لایه داخلی Internal (Physical) Level
- سطح انتزاع **پایین‌ترین**
- موضوع آن چگونگی **ذخیره‌سازی فیزیکی اطلاعات پایگاه داده‌ها در دیسک سخت**
- تعیین جزئیات عملکرد این لایه توسط DBMS و در برخی موارد با هدایت DBA

لایه ادراکی(سطح مفهومی) Conceptual Level
- سطح **میانی** انتزاع
- موضوع آن **نحوه سازماندهی و مدل‌سازی اطلاعات** در پایگاه داده
- **ساختار کلی طراحی شده** برای هر پایگاه داده را شامل می‌شود
- **فقط در دسترس DBA قرار دارد** و DBA جزئیات طراحی آن را مشخص می‌کند
- مهمترین لایه برای پایگاه داده تلقی می‌شود (ابزارها و مدل‌ها)

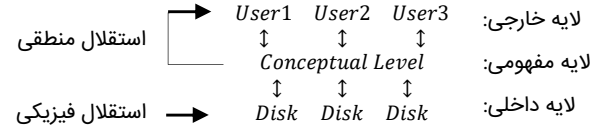
لایه ادراکی
- طراحی مفهومی (ادراکی عام) (***Conceptual Design***)
  - مستقل از پایگاه داده
  - ایجاد شمای مفهومی به کمک ابزار ER و ERR
- طراحی منطقی (ادراکی خاص) (***Logical Design***) انتخاب یک مدل برای بانک اطلاعاتی

لایه خارجی External (View) Level
- سطح انتزاع **بالاترین**
- موضوع آن **مدیریت کاربران (تعیین حوزه دید)**
- حوزه‌های دید کاربران توسط DBA و به DBMS معرفی می‌شود

حوزه دید کاربر (View):
- بخشی از پایگاه داده که مطابق قواعد، کاربر اجازه و امکان کار با آن بخش را دارد.
- تاثیرات
  - **محدود سازی سطح دسترسی کاربران**
  - **تامین امنیت پایگاه داده**
  - **موجب تسهیل کار کاربران**

استقلال داده ای (Data Independence): یعنی عدم وابستگی لایه خارجی به دیگر لایه‌ها در معماری ANSI یا به بیان دیگر مصونیت برنامه‌های کاربردی و دید کاربران در سطح خارجی در برابر تغییرات اعمال شده در سطح پایین‌تر.

استقلال داده‌ای
- استقلال فیزیکی: لایه فیزیکی بطور **کامل** از لایه ادراکی مستقل است. مثال تعویض هارد در سخت افزار.
- استقلال منطقی: لایه ادراکی بطور **نسبی** از لایه خارجی مستقل است. مثال اضافه کردن فیلد در جدول.



اطلاعاتی که در یک DB ذخیره می‌شوند
- ۱. اطلاعات اصلی $(Data)$
- ۲. اطلاعات کنترلی (مدیریتی) $(Meta\ Data)$

چند اصطلاح مهم در پایگاه داده:

- **کاتالوگ سیستم (System Catalog):** محلی برای ذخیره‌سازی **فراداده‌ها** (Meta Data) است. **مدیریت کاتالوگ** پایگاه داده **یکی از اصلی‌ترین وظایف DBMS** محسوب می‌شود.

- **لغت‌نامه داده‌ها (فرهنگ داده‌ها) (Data Dictionary):** زیرمجموعه کاتالوگ سیستم است که شامل فهرست اسامی نظیر اسامی جدول‌ها و ایندکس‌ها، صفات و قوانین امنیتی، قوانین جامعیتی و ... می‌باشد.
  نکته: با توجه به اینکه لغت‌نامه زیرمجموعه کاتالوگ سیستم است، لذا ایجاد **viewها** و **indexها** باعث **تغییر** در کاتالوگ سیستم در پایگاه داده می‌شود.

- **پرس‌وجو (Query):** هرگونه درخواستی که یک کاربر به DBMS ارسال می‌کند تا بتواند به این صورت با داده‌های موجود در پایگاه داده کار کند.

- **تقسیم بندی قدیمی**
  - زبان تعریف داده‌ها $DDL\ (\boldsymbol{D}ata\ \boldsymbol{D}efinition\ \boldsymbol{L}anguage)$
  - زبان دستکاری داده‌ها $DML\ (\boldsymbol{D}ata\ \boldsymbol{M}anipulation\ \boldsymbol{L}anguage)$
  - زبان کنترل داده‌ها $DCL\ (\boldsymbol{D}ata\ \boldsymbol{C}ontrol\ \boldsymbol{L}anguage)$

- **تقسیم بندی جدید (فصل ۶)**
  - DDL
  - DML
  - $Embedded\ SQL\ \&\ Dynamic\ SQL$
  - دستورات $Integrity$
  - دستورات $Authorization$
  - دستورات کنترل تراکنش‌ها $(Transaction\ Control)$
  - دستورات تعریف حوزه دید کاربران $(View)$

تعاریف
- DDL مجموعه‌ای از دستورالعمل‌ها که در صورت اجرا **ساختار** پایگاه داده را تحت تاثیر قرار می دهند.
- DML در صورت اجرا **محتوای** پایگاه داده را تحت تاثیر قرار می دهند اما اثری بر ساختار پایگاه داده ندارند.
- DCL **اجرای عملیات** در پایگاه داده را کنترل و راهبری می‌کنند و اثری بر ساختار یا محتوای پایگاه داده ندارند.
- DSL **اجتماع مجموعه دستورات** $\boldsymbol{DDL, DML, DCL}$ را گویند، که زبان پرس‌وجو $(Query\ Language)$ هم خوانده می‌شود.

پایگاه داده شناخت ≡ پایگاه بصیرت ≡ پایگاه داینامیک ≡ پایگاه پویا (Knowledge Base):
پایگاه داده‌ای است که به تدریج **به دانش آن افزوده** می‌شود. برای چنین پایگاهی باید از هوش‌مصنوعی و پردازش زبان طبیعی استفاده کرد که پایگاهی را ایجاد کند که **قادر به استنتاج منطقی از داده‌های ذخیره شده** باشد.

موجودیت (Entity): مفهوم کلی یک شخص یا یک پدیده یا هر چیزی است که می خواهیم در مورد آن اطلاعات داشته باشیم و اطلاعات خود را نسبت به آن افزایش دهیم.

ارتباط (Relation): در صورتیکه n تعداد موجودیت‌ها باشد مفهوم ارتباط یعنی تعامل بین نمونه‌های موجودیت که $n \geq 1$ می‌باشد. بر این مبنا یک موجودیت با خودش نیز در ارتباط است.

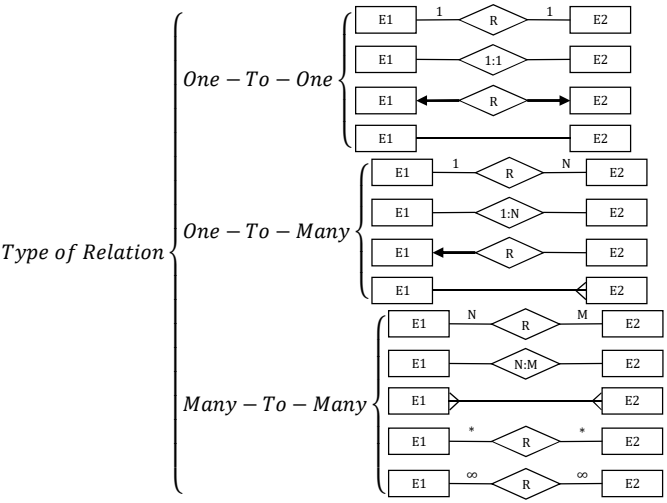
مفاهیم نمودار ER

- **درجه ارتباط:** تعداد موجودیت‌های شرکت‌کننده در یک ارتباط.
- **مجموعه صفات موجودیت** ($Attribute\ Set$) : ویژگی‌های یک موجودیت که آن را توصیف می‌کنند.
- **وابستگی موجودی:** وابستگی حضور یک پدیده به حضور پدیده دیگر
  - **موجودی ضعیف** ($Weak\ Entity$)
  - **موجودی قوی** ($Strong\ Entity$)
- **ارتباط بازگشتی** ($Recursive\ Relation$) : ارتباط هر موجودیت با خودش.
- **حد ارتباط** ($Cardinality\ Relation$) : کمترین و بیشترین موجودیت شرکت کننده در یک ارتباط.
- **نوع ارتباط (چندی اتصال)** ($Type\ of\ Relation$)
  - $One - to - One$
  - $One - to - Many$
  - $Many - to - Many$

نکته: هر ارتباط از درجه ۳ یا بالاتر، با یک موجودیت پل به دو ارتباط از درجه ۲ تجزیه خواهد شد.

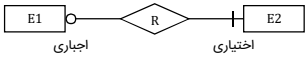
مجموعه صفات موجودیت

- **صفت کلید (شناسه):** تکرار و $Null$ نمی‌پذیرد.
- **صفت مرکب:**
- **صفت چند مقداری:**
- **صفت مشتق (محاسبه شدنی):**
  - انتخاب این صفت بعهده طراح پایگاه داده است.
  - به زمان نیز وابسته است.
  - نیازی نیست در جدول فیلدی تعریف شود.



نکته: هر ارتباط چند به چند بین دو موجودیت، با یک موجودیت پل، به دو ارتباط یک به چند تجزیه می‌شود.

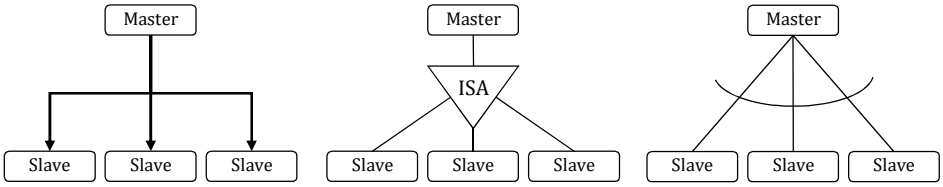
ارتباط اختیاری و اجباری: شرکت موجودیت E1 در ارتباط اجباری ولی شرکت موجودیت E2 اختیاری است.



نمودار موجودیت ارتباط توسعه یافته (EER (Extended Entity Relation: در این نمودار مفاهیمی که در نمودار ER قرار نمی‌گیرد، ارائه می‌شود که این مفاهیم با حضور مباحث شی‌گرا مطرح شده است.

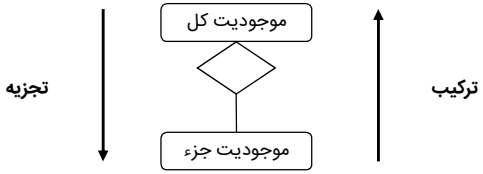
نمودار EER
- تعمیم (*Generalization*) **(یافتن اشتراک‌ها) (طراحی بالا به پایین)**
- تخصیص (*Specialization*) یافتن صفاتی که موجودیت‌ها را از هم متمایز می‌کند. **(یافتن تفاوت‌ها) (طراحی پایین به بالا)**
- تجمیع (*Aggregation*) موجودیت‌هایی در سطح پایین‌تر می‌توانند با یکدیگر، موجودیت در سطح بالاتر را تشکیل دهند.

نکته: اگر در نمودار EER عبارت Generalization یا Specialization ذکر نشود آنگاه هر دو دیدگاه را باید در نظر گرفت.



$$Master \equiv Parent \equiv Super\ Class$$
$$Slave \equiv Child \equiv SubClass$$

تجزیه و ترکیب:



مدل‌های پایگاه داده
- مدل‌های قدیمی (بر اساس ریاضی نیستند)
  - مدل سلسله مراتبی $(Hierarchical\ Model)$
  - مدل شبکه‌ای $(Network\ Model)$
- بر پایه ریاضیات
  - مدل رابطه‌ای (مدل سنتی) $(Relational\ Model)$
  - مدل فرا رابطه‌ای (شیءگرایی) (مدل مدرن) $(Object\ Oriented\ Model)$

مدل سلسله مراتبی:
- **قدیمی‌ترین** مدل است.
- در قالب سلسله‌ای از روابط **یک‌به‌یک** و **یک‌به‌چند یکسویه** سازماندهی شده است.
- یک **درخت‌واره** است. (گرافی است دارای یک ریشه بهم بسته و **غیرچرخشی** یا غیربرگشتی).
- **افزونگی** این مدل **بسیار بالا** است.
- دارای مشکلات **آنومالی** در درج، حذف و به‌روز رسانی است.
- **پیچیدگی بالا**

آنومالی (Anomaly): بطور کلی منظور از آنومالی **وجود مشکل در انجام عملیات** و **تحمیل هزینه زیاد** در اجرای عملیات مورد نظر است.

مدل شبکه‌ای:
- توسعه یافته مدل سلسله مراتبی.
- اطلاعات **در قالب یک گراف (شبکه)**، سازماندهی می‌شوند.
- تفاوت اصلی با مدل سلسله مراتبی در **پشتیبانی از روابط چندبه‌چند** است.
- **افزونگی داده** در این مدل **بسیار کمتر** از مدل سلسله مراتبی است.
- **مشکلات آنومالی** موجود در مدل سلسله مراتبی **حذف شده** است.
- **پیچیدگی بالا** همانند مدل سلسله مراتبی

مدل رابطه‌ای:
- **موفق‌ترین** و معروف‌ترین.
- اطلاعات در قالب مجموعه‌ای از **جداول** و روابط بین آنها در چهارچوب **جبر رابطه‌ای**، سازماندهی می‌شوند.
- پایه این مدل **بر مبنای ریاضیات مجموعه‌ها** می‌باشد.
- دارای مزیت **سادگی**

مدل رابطه‌ای:
- **مزایا**
  - سادگی
  - مجموعه‌ای بودن (ارائه پاسخ فقط در قالب مجموعه)
  - اتکا به یک تئوری قدرتمند بنام جبر رابطه‌ای **(مهمترین مزیت)**
  - برآورده نمودن کلیه نیازها در پایگاه داده متنی
- **معایب**
  - هر نوع اطلاعاتی را نمی‌توان در قالب مجموعه‌ای از جداول و روابط بین آنها مدل‌سازی کرد.
  - یا بطور کلی هر اطلاعاتی که ساختار پیچیده‌ای (اطلاعات چندرسانه‌ای) دارد قابل مدل‌سازی توسط این مدل نیست. مثال اطلاعات صوتی، تصویری یا ویدیوئی

مدل شیءگرایی:

- اطلاعات در لایه ادراکی پایگاه داده در قالب مجموعه‌ای از کلاس‌ها و روابط بین آن‌ها در چارچوب تئوری
- توانایی بالایی در مدل‌سازی اطلاعات با ساختارهای پیچیده‌تر دارد.
- بهترین گزینه برای مدل‌سازی اطلاعات چندرسانه‌ای.
- علاوه بر مدل‌سازی ساختار اطلاعاتی، سیستم عملیات مورد نیاز مربوط به این ساختار را نیز پشتیبانی می‌کند.

مدل رابطه‌ای و مفاهیم مرتبط با آن:

در این مدل هر کدام از موجودیت‌های عملیاتی تبدیل به یک رابطه می‌شود.

مدل رابطه‌ای	پیاده‌سازی
رابطه	جدول
تاپل (چندتایی)	سطر (رکورد)
صفت	ستون (فیلد)
دامنه (میدان)	نوع داده‌ای
کاردینالیتی	تعداد سطرها
درجه	تعداد ستون‌ها

نکته: **کاردینالیتی همواره** تغییر می‌کند ولی **درجه می‌تواند** تغییر کند. به عبارتی تغییر کاردینالیتی به مراتب بیشتر از درجه است.

روش‌های نمایش یک رابطه:

روش عادی:

$$\underbrace{Student}_{\text{عنوان، سرآیند } Head}\ (\ \underbrace{\underbrace{Student\ No.}_{\text{صفت (فیلد، ستون) } Attribute\ (Column)}\ :\ \underbrace{int}_{\text{دامنه (میدان، نوع) } Domain\ (Data\ Type)}\ ,\ \dots}_{\text{بدنه، پیکره } Body}\ )$$

روش جدولی:

$$Head\ \{\ St.No \quad Name \quad Address$$
$$Body\ \{\ 7221 \quad Ali \quad Tehran$$

$$A \equiv Attribute,\ D \equiv Domain,\ V \equiv Value$$
$$Head \rightarrow < A_i : D_i > \quad i = 1,2,3,\dots,n$$
$$Body \rightarrow < A_i : D_i : V_i > \quad i = 1,2,3,\dots,n$$

خواص یک رابطه:

۱. رابطه، تاپل تکراری ندارد. (همان خاصیت عدم تکرار اعضا در مجموعه‌ها).
۲. در رابطه، تاپل‌ها دارای نظم مکانی نیستند. (خاصیت جابجایی در مجموعه‌ها).
۳. در رابطه، صفات رابطه نیز نظم مکانی ندارند.
۴. تمامی مقادیر صفات یک رابطه باید بصورت تک‌مقدار $(atomic)$ باشند.

رابطه تهی: یعنی رابطه بدون تاپل.

مفهوم کلید: ترکیبی از یک یا چند صفت که می‌تواند یک سطر را از سطر دیگر متمایز کند.

کلید:
- اجباری غیر از کلید خارجی
  - تکرار قبول نمی‌کند.
  - نمی‌تواند $Null$ باشد.
- **بهتر است** کمینه باشد. (تا جای امکان ترکیبی هم نباشد).
- **بهتر است** طول رشته بایتی کلید کوتاه باشد.
- **بهتر است** صفتی باشد که مقدار آن تغییر نکند با به ندرت تغییر کند. (مثال آدرس تغییر می‌کند)

انواع کلید به ترتیب:
۱. ابرکلید $Super\ Key\ (SK)$
۲. کلید کاندید $Condidated\ Key\ (CK)$
۳. کلید اصلی $Primary\ Key\ (PK)$
۴. کلید ثانویه ≡ کلید جانشین $Alternative\ Key \equiv Secondary\ Key\ (SK)$
۵. کلید خارجی $Foreign\ Key\ (FK)$

ابرکلید (SK):
- هر ترکیبی از صفات یک رابطه که خاصیت کلیدی دارند.
- الزاما یک کلید کمینه نیست. (اگر کمینه باشد کلید کاندید محسوب می‌شود).

نکات:
- صفتی که دارای مقدار Null است نه خودش ابرکلید (SK) و نه می‌تواند با بقیه صفات در تبدیل شدن به SK شرکت کند.
- صفتی که مقدار آن تکراری است خودش نمی‌تواند SK باشد اما می‌تواند با بقیه صفات در تبدیل شدن به یک SK شرکت کند.

تعداد ابرکلید: با توجه به اینکه زیرمجموعه‌های یک مجموعه (Power Set) شامل تهی (ϕ) هم هست اما از آنجا که تهی یا Null نمی‌تواند کلید باشد، لذا از کل زیرمجموعه‌ها (2^n) یک واحد کم می‌کنیم یعنی $2^n - 1$:

$$\left.\begin{matrix} n = \text{درجه رابطه} \\ N = \text{تعداد ابرکلید} \end{matrix}\right\} 1 \le N \le 2^n - 1$$

اگر کلیدهای اصلی و کاندید تک ستونی (غیرترکیبی) باشند:

$$\left.\begin{matrix} n = \text{درجه رابطه (تعداد ستون‌ها)} \\ K = \text{تعداد کلیدهای کاندید تک ستونی} \\ N = \text{تعداد ابرکلید} \end{matrix}\right\} N = (2^k - 1) \times 2^{n-k}$$

رابطه رامزالمصری: [صفات غیرکلید] + [کلید کاندید و کلید اصلی] یا [کلید کاندید] یا [کلید اصلی] = **ابر کلید**

کلید کاندید (CK):
- هر ترکیبی از صفات یک رابطه که خاصیت کلید دارند.
- بهتر است کمینه باشد. (ابرکلید کمینه).

کلید اصلی (کلید جستجو) (PK): کلیدی است که طراح پایگاه داده بر اساس ۲ ضابطه، از روی کلیدهای کاندید، آن را تعیین می‌کند.
۱. ضابطه ادراکی: نقش و اهمیت کدام کلید در پاسخگویی به نیازهای عملیاتی بیشتر است.
۲. ضابطه فیزیکی: کدام کلید طول رشته بایتی کوتاهتری دارد.

نکته: هر جدول (رابطه)، فقط یک کلید اصلی دارد.

کلید جانشین (ثانویه) (SK):
- هر کلید کاندید به غیر از کلید اصلی را کلید جانشین می‌گویند.
- کاربرد در مبحث مهندسی فایل‌ها، ایجاد اندیس در پایگاه داده است.
- می‌تواند وجود نداشته باشد.

کلید خارجی (SF): صفتی در یک رابطه کلید خارجی است اگر این صفت در همان رابطه یا رابطه دیگری، کلید اصلی یا کاندید یا فرعی باشد.
- تکرار قبول می‌کند.
- می‌تواند $Null$ باشد.
- برای کلید خارجی ارجاع $Null$ قابل قبول نیست.
- می‌تواند خصیصه مرکب (ترکیبی) باشد زیرا کلید اصلی می‌تواند ترکیبی باشد.

نکته: اگر کلید خارجی ترکیبی بود، یا تماما Null است یا تماما غیر Null، یعنی نمی‌تواند فقط بخشی از آن Null باشد.

افزونگی تکنیکی: به افزونگی حاصل از **کلید خارجی** گویند.

تمام کلید (All Key): رابطه‌ای است که مجموع تمام صفات آن با هم، کلید کاندید است و **بدترین حالت** است.

قوانین جامعیت (Integrity Rule) در مدل رابطه‌ای: ضامن آن Control است و به ۲ گروه عمده تقسیم می‌شوند.

انواع قوانین جامعیت:
- **قواعد تعریف شده توسط کاربر** $(User\ Defined\ Rule)$ **محدودیت‌های انحصاری** وابسته به پایگاه **خاص**
- **قواعد عام** $(Meta\ Rule)$ **محدودیت‌های عمومی** مربوط به **کلیه** پایگاه داده‌ها

قواعد عام:
۱. قانون جامعیت درون رابطه‌ای $(Intra - Relation\ Integrity\ Rule)$
۲. قانون جامعیت موجودی ≡ قانون جامعیت دامنه $(Entity\ Integrity\ Rule \equiv Domain\ Integrity\ Rule)$
۳. قانون جامعیت ارجاعی $(Referential\ Integrity\ Rule)$

۱. **قانون جامعیت درون رابطه‌ای:** هر رابطه باید به تنهایی درست (True) باشد یعنی نباید تعریف رابطه را آنچنان که در تئوری جبر رابطه آمده، نقض کند.

به عبارتی
- رابطه تاپب تکراری نداشته باشد.
- کلیدهایش به درستی تعریف شده باشد. (باید حتما کلید اصلی داشته باشد و ...).
- مولفه‌های متناظر در تاپل های یک رابطه باید هم دامنه باشند.
- در یک رابطه نباید تاپل‌هایی با درجه‌های متفاوت وجود داشته باشد. (مثلا همه دوتایی مرتب باشند).

۲. **قانون جامعیت موجودیت:** مطابق این قانون که به **کلید اصلی** مربوط می‌شود هیچ جزء تشکیل دهنده کلید اصلی نباید Null باشد.

قانون جامعیت دامنه: در برخی منابع قانون جامعیت موجودیت کمی بسط داده و تحت این عنوان مطرح شده است.
- کلید اصلی یا بخش‌های مختلف آن نباید مقدار Null داشته باشند. (همانند قانون جامعیت موجودیت)
- مقادیر همه مولفه‌های تاپل‌ها در یک رابطه باید در دامنه تعریف شده برای آن مولفه قرار داشته باشند.

۳. **قانون جامعیت ارجاع:** این قانون محدودیت‌هایی را برای **کلید خارجی** مطرح می‌کند. مطابق این قانون، جامعیت ارجاع باید هم بصورت **ساختاری** و هم بصورت **محتوایی** برقرار باشد.

نکته: بصورت پیش فرض فعال نیست و باید فعال شود.

جامعیت ارجاع:
- **ساختاری:** عدم نقض **ارجاع** ***Null***
- **محتوایی:** همان ***CASCADE***

ارجاع Null: یعنی کلید خارجی نمی تواند دارای مقداری باشد که آن مقدار در جدول مرجع وجود نداشته باشد.

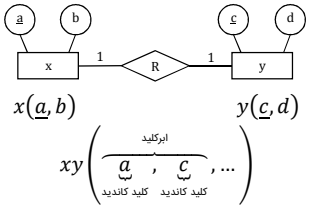
ارجاع CASCADE: یعنی هرگونه اعمال تغییر روی مقدار کلید خارجی باید هماهنگ با مقدار کلید کاندید مربوطه در جدول مرجع صورت پذیرد. علاوه بر این هرگونه تغییر محتوایی کلید کاندید در جدول مرجع نیز باید هماهنگ با کلیدهای خارجی مربوطه در جداول ارجاع کننده انجام گیرد.

هر موجودیت ← یک رابطه ← یک جدول

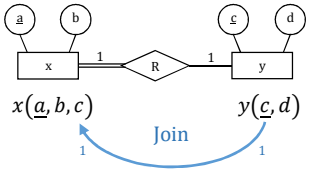
تبدیل نمودارهای ER به مدل رابطه‌ای:
۱. تبدیل ارتباط یک‌به‌یک بین دو موجودیت به مدل رابطه‌ای
۲. تبدیل ارتباط یک‌به‌چند بین دو موجودیت به مدل رابطه‌ای
۳. تبدیل ارتباط چندبه‌چند بین دو موجودیت به مدل رابطه‌ای

۱. تبدیل ارتباط یک‌به‌یک بین دو موجودیت به مدل رابطه‌ای: هر موجودیت تبدیل به یک رابطه می‌شود و یک رابطه مابین آنها به عنوان **رابطه اتصال دهنده** ایجاد خواهد شد.

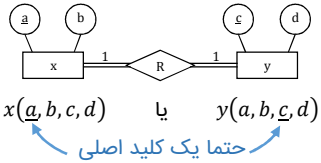
۱-۱. اگر دو طرف رابطه اختیاری باشند:



۱-۲. اگر یک طرف اختیاری و طرف دیگر اجباری باشد هر موجودیت به یک جدول تبدیل می‌شود و کلید کاندید موجودیت اختیاری در جدول دیگر می‌تواند به عنوان کلید ارتباط دهنده تعریف شود:

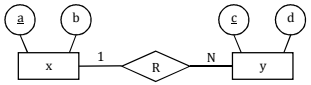


۱-۳. اگر هر دو طرف ارتباط اجباری باشند، هر دو موجودیت در قالب یک جدول قرار می‌گیرند و کلید کاندید جدول حاصل می‌تواند کلید کاندید هرکدام از موجودیت‌ها باشد.



نکته: در ارتباط یک‌به‌یک، کلید خارجی ایجاد نمی‌شود زیرا کلید خارجی برای چند استفاده می‌شود.

۲. **تبدیل ارتباط یک‌به‌چند بین دو موجودیت به مدل رابطه‌ای:** مستقل از اختیاری یا اجباری بودن موجودیت‌ها در ارتباط، هر جدول به یک رابطه تبدیل می‌شود و کلید کاندید سمت یک، به عنوان کلید خارجی در سمت چند تعریف می‌شود.

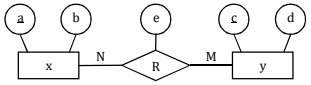


$x(\underline{a}, b)$ $\qquad$ $y(\underline{c}, d, a)$

1 Join N

نکته: در ارتباط یک‌به‌چند اگر شرکت موجودیت‌ها در ارتباط اختیاری باشد کلید خارجی می‌تواند Null قبول کند و اگر اجباری باشد نباید کلید خارجی Null بپذیرد.

۳. **تبدیل ارتباط چندبه‌چند بین دو موجودیت به مدل رابطه‌ای:** مستقل از اختیاری یا اجباری بودن موجودیت-ها در ارتباط، هر موجودیت به یک رابطه تبدیل می‌شود و یک جدول پل (Bridge) بین آنها مطرح خواهد شد که ارتباط چندبه‌چند را به دو ارتباط یک به چند تبدیل خواهد کرد.



$x(\underline{a}, b)$ $\qquad$ $y(\underline{c}, d)$

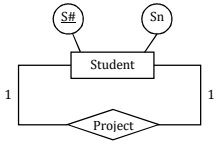
$$xy\left(\underbrace{\underline{a}}_{\text{کلید خارجی}}, \underbrace{\underline{c}}_{\text{کلید خارجی}}, e\right)$$

نکته: در ارتباط یک‌به‌چند یا چندبه‌چند، اگر ارتباط چند، اختیاری باشد آنگاه می‌تواند Null بپذیرد.

تبدیل ارتباطات درجه یک (یگانی) به مدل رابطه‌ای:

۱. **اگر یک موجودیت با خودش ارتباط یک‌به‌یک داشته باشد:**

۱-۱. اگر شرکت این موجودیت در ارتباط **اختیاری** باشد، موجودیت به یک جدول تبدیل می‌شود و یک جدول دیگر جهت نمایش رابطه موجود، مطرح خواهد شد.



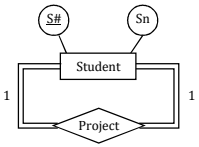
1 1

$S_{\#}$	S_{name}
S_1	S_{n1}
S_2	S_{n2}
S_3	S_{n3}

$S_{\#}$	$SP_{\#}$
S_1	S_2
S_2	S_1

توجه: S_1 دیگر نمی‌تواند هم گروه دیگر داشته باشد زیرا ارتباط آن یک‌به‌یک است.

۲-۱. اگر شرکت این موجودیت در ارتباط **اجباری** باشد، فقط یک جدول ایجاد می‌شود که در این جدول یک حالت ارتباط داخلی ایجاد خواهد شد.

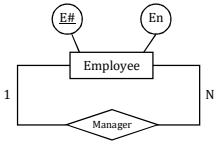


$S_{\#}$	S_{name}	$SP_{\#}$
S_1	S_{n1}	S_2
S_2	S_{n2}	S_1
S_3	S_{n3}	S_4
S_4	S_{n4}	S_3

طوقه

توجه: با اینکه $SP_{\#}$ کلید خارجی است ولی چون اجباری است نمی تواند Null قبول کند و از طرفی چون یک‌به‌یک است تکرار هم قبول نمی‌کند.

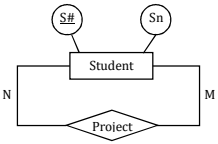
۲. اگر یک موجودیت با خودش ارتباط یک‌به‌چند داشته باشد: مستقل از اختیاری یا اجباری بودن موجودیت، موجودیت مورد نظر به یک جدول تبدیل می‌شود و کلید اصلی این جدول در خود همین جدول به عنوان کلید خارجی تعریف می‌شود.



$E_{\#}$	E_{name}	$EE_{\#}$	
e_1	e_{n1}	$Null$	به علت وجود اختیار در رابطه
e_2	e_{n2}	e_1	
e_3	e_{n3}	e_2	به علت وجود N در رابطه
e_4	e_{n4}	e_1	
e_5	e_{n5}	e_3	

طوقه

۳. اگر یک موجودیت با خودش ارتباط چندبه‌چند داشته باشد: مستقل از اختیاری یا اجباری بودن شرکت در ارتباط، موجودیت به یک جدول اصلی تبدیل می‌شود و یک جدل ارتباط دهنده برای نمایش ارتباط آنها ایجاد خواهد شد.

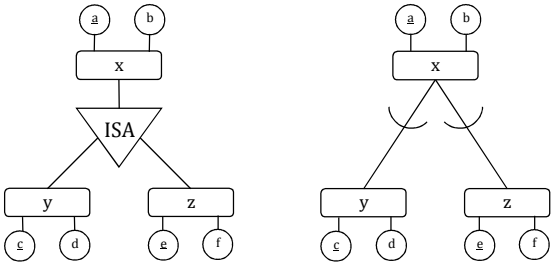


N M

$S_{\#}$	S_{name}
S_1	S_{n1}
S_2	S_{n2}
S_3	S_{n3}
S_4	S_{n4}

$S_{\#}$	$SS_{\#}$
S_1	S_2
S_2	S_1
S_3	S_2

تبدیل نمودار EER به مدل رابطه‌ای:



تعمیم (Generalization):

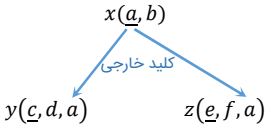
۱. **گسسته** (***Disjoint Generaliztion***)

۲. **اشتراکی** (***Overlap Generaliztion***)

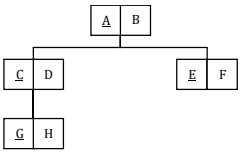
۱. **تعمیم گسسته:** برای پدر جدولی ایجاد نمی‌شود. برای هر فرزند جدول ایجاد می‌کنیم و صفات پدر را در هرکدام از فرزندان قرار می‌دهیم. طبیعتا **افزونگی** دارد و حالت بهینه نیست.

$$y(\underline{a},b,c,d) \quad , \quad z(\underline{a},b,e,f)$$

۲. **تعمیم اشتراکی:** برای پدر و فرزندان جدول ایجاد می‌کنیم و کلید اصلی پدر در جداول فرزند، کلید خارجی خواهد شد.



تبدیل پایگاه داده سلسله مراتبی به روابط: در پایگاه داده سلسله مراتبی به ازای هر نود یک جدول ایجاد می‌شود و کلید هر جدول، بصورت کلید جداول والد و کلید خود جدول است.



$(\underline{A},B)$
$(\underline{A,E},F)$
$(\underline{A,C},D)$
$(\underline{A,C,G},H)$

جبر رابطه‌ای
- نوع داده: رابطه
- عملگر:
  - عملگرهای ساده
    - محدودیت (گزینش) $\sigma\ (Select, Restrict)$
    - پرتو (تصویر) $\pi\ (Projection)$
  - عملگرهای مجموعه‌ای
    - اجتماع $\cup\ (Union)$
    - اشتراک $\cap\ (Intersect)$
    - تفاضل $-\ (Minus, Set\ Difference)$
  - عملگرهای پیوند (الحاق)
    - ضرب دکارتی $\text{X}\ (Cartesian\ Product, Times)$
    - پیوند طبیعی $(\infty \equiv \bowtie)\ Natural\ Join$
    - پیوند بیرونی (خارجی، فراپیوند) $Outer\ Join$ $L-O-Join$ $R-O-Join$ $F-O-Join$
    - پیوند شرطی $(\text{X}_\theta \equiv \infty_\theta \equiv \bowtie_\theta)\ Conditional\ Join$
    - نیم پیوند (شبه پیوند) $(\propto \equiv \ltimes)\ Semi\ Join$
  - عملگرهای متفرقه
    - عملگر تقسیم ($\div$ یا Div) $Divide\ By$
    - عملگر تغییر نام $Rename\ (\rho)$
    - عملگر جایگزینی $(\longleftarrow \equiv := \equiv)\ Substitute\ (Giving$

تقسیم‌بندی دیگر:
- **عملگرهای اصلی:** $-\quad \text{X}\quad \rho\quad \sigma\quad \pi\quad \cup$
- **عملگرهای فرعی:** $\leftarrow\quad \text{X}_\theta\quad \propto\quad \infty\quad \cap$

تقسیم‌بندی دیگر:
- **عملگرهای خاص:** $\pi\quad \div\quad \infty\quad \sigma$
- **عملگرهای عام:** $\text{X}\quad \cup\quad \cap$

عملگرهای جبر رابطه‌ای:

۱. عملگرهای ساده

۱-۱. عملگر انتخاب (σ): برای انتخاب تاپل‌هایی از یک رابطه با اعمال شرط.

$$\sigma_{Condition}^{(Relation\ Name)}$$

۱-۲. عملگر پرتو (π): برای انتخاب ستون‌هایی از یک رابطه. ضعف عملگر نداشتن شرط است. در خروجی این عملگر تاپل تکراری حذف می‌شود.

$$\pi_{Col1,\ Col2,\ \ldots,\ Coln}^{(Relation\ Name)}$$

نکته: در صورتی که بخواهیم ستون‌هایی از یک رابطه را با اعمال شرط انتخاب کنیم باید این دو عملگر را بصورت زیر ترکیب کنیم:

$$\pi_{Col1,\ Col2,\ \ldots,\ Coln}\left(\sigma_{Condition}^{(Relation\ Name)}\right)$$

۲. عملگرهای مجموعه‌ای

۲-۱. اجتماع (∪): همه سطرهای رابطه اول و دوم بطوریکه تاپل‌های تکراری حذف می‌شوند. اگر بخواهیم تاپل‌های تکراری حذف نشوند باید از Union All استفاده شود.

۲-۲. اشتراک (∩): برای اینکه تاپل‌های تکراری حذف نشوند باید از Intersect All استفاده شود.

۲-۳. تفاضل (−): در این عملگر خاصیت جابجایی نداریم.

$$R_1\ minus\ R_2 = R_1\ minus\ (R_1\ Intersect\ R_2)$$
$$R_1\ minus\ R_2 \neq R_2\ minus\ R_1$$

نکته: در استفاده از عملگرهای مجموعه‌ای باید شرط همتایی عملوندها رعایت شود.

شرط همتایی (Same Arity):
- تعداد ستون‌ها برابر باشند.
- ستون‌ها نظیر به ظیر هم نام باشند.
- ستون‌ها نظیر به نظیر هم دامنه باشند.

۳. عملگرهای پیوند (الحاق) (Join)

طی این نوع عملگرها انتخاب ستون‌ها و سطرها بصورت **همزمان** از بیشتر از یک جدول انجام می‌شود.

۳-۱. ضرب دکارتی (X): بیشتر زمانی استفاده می‌کنیم که دو رابطه جهت پیوند هیچ ستون مشترک کلیدی نداشته باشند.

نکات:
- ضرب دکارتی در جبر رابطه‌ای برخلاف ریاضیات مجموعه‌ها **خاصیت جابجایی** دارد زیرا ترتیب قرارگیری ستون‌ها و سطرها اهمیت ندارد.
- در ضرب دکارتی اگر ستون مشترک غیرکلیدی وجود داشت، هر دو بار در خروجی ظاهر می‌شود.
- اگر $\begin{cases} R_1\ (\text{ستون } m \text{ و سطر } n) \\ R_2\ (\text{ستون } q \text{ و سطر } p) \end{cases}$ آنگاه $R_1 \times R_2 \begin{cases} \text{سطر} = m \times p \\ \text{ستون} = n \times q \end{cases}$
- میان عملگرهای پیوند **بیشترین افزونگی** را دارد.

۳-۲. پیوند طبیعی (∞): زمانی استفاده می‌شود که دو جدول برای پیوند، فیلد مشترک کلیدی داشته باشند. هر سطر از رابطه اول در کنار سطرهایی از رابطه دو پیوند می‌خورد که فیلد مشترک کلیدی آنها مقداری برابر دارد.

$$R_1 \infty R_2 \equiv R_1 \bowtie R_2 \equiv R_1 \text{ Join } R_2 \equiv Join\ R_1 \text{and } R_2\ over *$$

نکات:

- افزونگی کمتر نسبت به ضرب دکارتی.
- **خاصیت جابجایی** دارد.
- ستون‌های مشترک کلیدی در خروجی آن فقط یک بار ظاهر می‌شود.
- اگر نوع پیوند ذکر نشود **پیش‌فرض** است.
- ضعف این عملگر در آن است که فقط می‌تواند سطرهایی از دو رابطه را پیوند دهد که قابل پیوند شدن هستند. بعبارتی سطری که فیلد کلیدی آن در رابطه دیگر وجود نداشته باشد حذف می‌شود.
- اگر دو رابطه که هیچ ستون مشترکی ندارند پیوند طبیعی شوند، خروجی همان عملگر دکارتی خواهد بود.
- اگر دو رابطه که تمام ستون‌هایش همانند یکدیگر باشند پیوند طبیعی شوند، خروجی همان عملگر اشتراک خواهد بود. دقت شود که ستون‌های یکسان به معنای محتوا یا سطرهای یکسان نخواهد بود.

۳-۳. پیوند بیرونی (فراپیوند، خارجی) (Outer Join):

۳-۳-۱. عملگر الحاق خارجی چپ (⟕) (Left Outer Join): زمانی استفاده می‌شود که مقدار فیلد کلیدی در جدول سمت چپ وجود دارد ولی همان مقار در جدول سمت راست وجود ندارد. Null در قسمت راست خروجی وجود دارد.

۳-۳-۲. عملگر الحاق خارجی راست (⟖) (Right Outer Join): زمانی استفاده می‌شود که مقدار فیلد کلیدی در جدول سمت راست وجود دارد ولی همان مقدار در جدول سمت چپ وجود ندارد. Null در قسمت چپ خروجی وجود دارد.

۳-۳-۳. عملگر الحاق خارجی کامل (⟗) (Full Outer Join): کلیه سطرهای **پیوندپذیر** و **پیوندناپذیر** دو جدول در خروجی ظاهر می‌شوند و در مورد سطرهای پیوندناپذیر برای ستون‌های مربوطه مقادیر Null در نظر گرفته می‌شود.

نکته: عملگرهای الحاق خارجی چپ و راست خاصیت جابجایی ندارند ولی عملگر الحاق خارجی کامل دارای خاصیت جابجایی است.

۳-۴. پیوند شرطی ($X_\theta \equiv \infty_\theta \equiv \bowtie_\theta$) (Conditional Join): ابتدا دو رابطه را به هم ضرب دکارتی کرده و سپس شرط θ را بر روی سطرهای آن اعمال می‌کند.

۳-۵. نیم پیوند (شبه پیوند) ($\propto \equiv \ltimes$) (Semi Join): از لحاظ ارزش مشابه عملگر پیوند طبیعی است با این تفاوت که در خروجی این عملگر فقط ستون‌های جدول سمت چپ نماد ظاهر می‌شود.

نکات:

- میان عملگرهای پیوند **کمترین افزونگی** را دارد.
- خاصیت جابجایی ندارد: $R_1 \ltimes R_2 \neq R_2 \ltimes R_1$

۴. عملگرهای متفرقه:

۴-۱. عملگر تقسیم (÷ یا Div یا Divide By): زمانی استفاده می‌کنیم که بخواهیم **همه حالات** یک اتفاق را بررسی کنیم. در جبر رابطه‌ای شرط تقسیم این است که ستون‌هایی که در مقسوم علیه آمده همگی زیرمجموعه مقسوم باشند.

نکته: تقسیم رابطه به خودش تهی می‌شود.

۴-۲. عملگر تغییر نام (÷ یا Div یا Divide By): یک عبارت جبر رابطه‌ای یا یک رابطه را دریافت و در خروجی یک نام جدید ایجاد می‌نماید که در واقع اشاره‌گر به نام قبلی دارد.

$$\rho_{\text{(نام جدید)}}^{\text{(یک عبارت جبر رابطه‌ای یا یک رابطه)}}$$

نکته: یکی از کاربردهای اساسی این عملگر زمانی است که نیاز است سطرهای یک جدول با سطرهای دیگری از همین جدول مقایسه شود.

۴-۳. عملگر جایگزینی ($\leftarrow \equiv := \equiv Giving$): با استفاده از این عملگر می‌توانیم یک عبارت جبر رابطه‌ای طولانی و پیچیده را بصورت مرحله به مرحله و با استفاده از جداول موقت محاسبه کرده و در خروجی نمایش دهیم. یعنی خروجی هر مرحله را در یک جدول موقت بنام Temp قرار داده و در مرحله بعدی استفاده نماییم.
از دیگر کاربردهای آن پشتیبانی سه دستور مهم Insert، Update و Delete می‌باشد. جز تقسیم برای بقیه دستورات SQL معادل جبر رابطه‌ای داریم:

جبر رابطه‌ای	زبان SQL
$\leftarrow, \cup$	Insert
$\leftarrow, \cap$	Update
$\leftarrow, -$	Delete

۴-۴. عملگر بسط (گسترش) (Extend): این عملگر یک رابطه را گرفته و رابطه دیگری همانند رابطه اولیه را برمی‌گرداند با این تفاوت که حاوی صفت دیگری است که مقدار آن با ارزیابی یک عبارت محاسباتی یا استفاده از توابع تجمعی حاصل می‌شود.

Extend نام رابطه ADD (عبارت محاسباتی یا توابع تجمعی) AS اسم مستعار

۴-۵. عملگر خلاصه (گروه‌بندی) (Summarize): وقتی بر روی یک ستون از این عملگر استفاده می‌کنیم مقادیری که بصورت مشابه در سطرهای مختلف تکرار شده‌اند فقط یکبار در خروجی بصورت یک گروه تکرار می‌شوند و بدین صورت می‌توانیم بر روی فیلدهای دیگر محاسبات بیشتری را انجام دهیم.

SUMMARIZE نام رابطه PER نام رابطه { نام صفت مورد نظر }
ADD (عبارت محاسباتی) AS اسم مستعار

بهینه‌سازی پرس‌وجوهای جبر رابطه‌ای: در ایجاد یک عبارت جبر رابطه‌ای قواعد زیر باید رعایت شود:

۱. عملگر σ باید هر چه زودتر انجام شود زیرا کم هزینه است و جدول حاصل را برای عملگرهای بعدی کوچکتر می‌کند.

۲. در عملگر σ ، بهتر است شرط‌های ترکیبی (فقط ترکیب عطفی) به شرط‌های متوالی تبدیل شوند.

$$\sigma^{(r)}_{p_1 \wedge p_2} \Longrightarrow \sigma_{p_1}\left(\sigma^{(r)}_{p_2}\right)$$

۳. بعد از عملگر σ باید عملگر π هرچه روزدتر انجام شود تا باز هم جداول حاصل کوچکتر شود.

۴. اگر کاردینالیتی رابطه r بیشتر از کاردینالیتی رابطه s باشد آنگاه در عملگرهای جبر رابطه‌ای باید جدول بزرگتر سمت چپ عملگر قرار گیرد: $r \times s$ یا $r \cup s$ یا $r \cap s$ یا $r \infty s$ و ...

۵. در هنگام استفاده از عملگرهای اشتراک و تفاضل می‌توان جهت بهینه‌سازی از روابط زیر استفاده کرد:

$$\sigma^{(r \cap s)}_{p} = \overbrace{\sigma^{(r)}_{p} \cap \sigma^{(s)}_{p}}^{\text{هزینه کمتر}}$$

$$\sigma^{(r-s)}_{p} \equiv \sigma^{(r)}_{p} - s \equiv r - \sigma^{(r)}_{p} \equiv \overbrace{\sigma^{(r)}_{p} - \sigma^{(s)}_{p}}^{\text{هزینه کمتر}} \quad : (floating) \text{خاصیت شناوری}$$

۶. بهینه‌سازی عملگر ∞ :

$$\sigma^{(r \infty s)}_{p_1 \wedge p_2} = \overbrace{\sigma^{(r)}_{p_1} \infty \sigma^{(s)}_{p_2}}^{\text{هزینه کمتر}}$$

مجموعه کامل عملگرها:

عملگرهای {σ, π, ∪, −, X}، عملگرهای مبنایی (اصلی) هستند یعنی می‌توان از روی آنها سایر عملگرها را ایجاد کرد.

$$A \cap B = A - (A - B) = B - (B - A) = (A \cup B) - (A - B) \cup (B - A)$$

$$A(y, x) \div B(x) = A[y] - \big((A[y] \times B) - A\big)[y]$$

$$R_1 \; semi \; minus \; R_2 = R_1 \; minus \; (R_1 \; semi \; join \; R_2)$$

$$R \div S = \pi^{(R)}_{r-s} - \pi_{r-s}\left(\left(\pi^{(R)}_{r-s} \times S\right) - \pi^{(R)}_{r-s,\ s}\right)$$

خواص عملگرهای جبر رابطه‌ای:

$$\sigma^{(R)}_{p \wedge q \wedge r} = \sigma_p\left(\sigma_q\left(\sigma^{(R)}_r\right)\right)$$

$$\sigma^{(R)}_{c-d} = \sigma^{(R)}_{c \wedge \sim d} = \sigma^{(R)}_{c} \cap \sigma^{(R)}_{\sim d} = \sigma_c\left(\sigma^{(R)}_{\sim d}\right)$$

$$\pi_{\langle L \rangle}\pi_{\langle M \rangle} \dots \pi_{\langle N \rangle} \quad , \quad L \subset M \subset N$$

$$R_1 \times R_2 = R_2 \times R_1 \qquad R_1 \bowtie R_2 = R_2 \bowtie R_1 \qquad R_1 - R_2 \neq R_2 - R_1$$
$$R_1 \times (R_2 \times R_3) = (R_1 \times R_2) \times R_3 \qquad R_1 \bowtie_\theta R_2 = R_2 \bowtie_\theta R_1 \qquad R_1 \ltimes R_2 \neq R_2 \ltimes R_1$$

حساب رابطه‌ای (Relational Calculus):

جهت پیاده‌سازی مدل رابطه‌ای یک تئوری بنام حساب رابطه‌ای وجود دارد که در این تئوری نیز رابطه‌ها مولفه‌های اصلی و اولیه هستند، اما در این تئوری عملگری وجود ندارد و به جای آن از صورهای وجودی و عمومی استفاده می‌شود. در این تئوری تاکید بر چگونگی تولید پاسخ مورد نظر از روی رابطه‌ها می‌باشد.

انواع حساب رابطه‌ای:
۱- حساب رابطه‌ای در تاپل‌ها
۲- حساب رابطه‌ای در دامنه‌ها

۱- حساب رابطه‌ای در تاپل‌ها:

وقتی یک عبارت جبر رابطه‌ای را می‌نویسیم، دنباله‌ای از روال‌ها را فراهم می‌کنیم که پاسخ پرس‌وجوی ما را تولید می‌کند. حساب رابطه‌ای روی تاپل‌ها، یک زبان پرس‌وجوی غیرروالی (Non Procedural) است. این زبان، اطلاعات موردنظر را بدون ارائه روال خاصی برای بدست آورن آن اطلاعات، توصیف می‌کند.

نکات:

- پرس‌وجو در حساب رابطه‌ای تاپل‌ها به صورت $\{ t \mid P(t) \}$ بیان می‌شود یعنی سطرهایی از رابطه که در آن شرط P بر روی آنها برقرار است را برگردان.
- در حساب رابطه‌ای در تاپل‌ها می‌توان از متغیر بازه‌ای که بر روی یک رابطه تعریف می‌شود و فقط تاپل‌های آن رابطه را به عنوان مقدار می‌پذیرد به شکل زیر استفاده کرد:

Rangevar RX Ranges Over R

که R یک رابطه است و RX یک متغیر بازه‌ای است.

- اگر دستور حساب رابطه‌ای در تاپل‌ها، از دو رابطه باشد باید دو بخش "وجود دارد" در عبارت حساب رابطه‌ای تاپل‌ها داشته باشیم که با and (علامت ∧) به هم متصل شوند.
- برای ایجاد عملگر اجتماع در حساب رابطه‌ای تاپل‌ها، به دو بخش "وجود دارد" نیاز داریم که با or (علامت ∨) به هم متصل شوند.
- در حساب رابطه‌ای می‌توان برای not کردن از نماد ¬ استفاده کرد.
- در حساب رابطه‌ای تاپل‌ها، استلزام یا ایجاب (implication) با استفاده از نماد ⇒ نشان داده می‌شود. فرمول $P \Rightarrow Q$ به معنای این است که "P ایجاب می‌کند Q را" و از نظر منطقی معادل $\neg P \vee Q$ است.
- در حساب رابطه‌ای تاپل‌ها، ساختار "forall" یا :به ازای تمام" را معرفی می‌کند که با ∀ نشان داده می‌شود. عبارت $\forall t \in r(Q(t))$ یعنی برای تمام تاپل‌های موجود در رابطه r درست است.
- اگر P_1 یک فرمول باشد، آنگاه $\neg P_1$ و (P_1) نیز فرمول هستند.
- اگر P_1 و P_1 فرمول باشن، آنگاه $P_1 \vee P_2$، $P_1 \wedge P_2$ و $P_1 \Rightarrow P_2$ فرمول هستند.
- اگر $P_1(s)$ فرمولی باشد که حاوی متغیر آزاد S باشد و r یم فرمول رابطه باشد، آنگاه عبارت زیر نیز فرمول است: $\exists s \in r(P_1(s)) \; and \; \forall s \in r(P_1(s))$
- $P_1 \wedge P_2$ معادل $\neg(\neg P_1 \vee \neg P_2)$ است.
- $\forall t \in r(P_1(t))$ معادل $\neg \exists t \in r(\neg P_1(t))$ است.
- $P_1 \Rightarrow P_2$ معادل $\neg(P_1) \vee P_2$ است.

۲- حساب رابطه‌ای دامنه‌ای (Domain Relational Calculus):

از متغیرهای دامنه‌ای استفاده می‌کند که از مقادیر دامنه‌ی صفات گرفته می‌شود نه از مقادیر کل تاپل.

$$\{< x_1, x_2, \dots, x_n > | P(x_1, x_2, \dots, x_n)\}$$

که $x_n, \dots, x_2, x_1$ متغیرهای دامنه را نشان می‌دهند. P فرمولی را نشان می‌دهد که شامل اتم‌ها است. اتم در حساب رابطه‌ای دامنه، به یکی از شکل‌های زیر است:

- $< x_1, x_2, \dots, x_n > \in r$ که r یک رابطه روی n صفت است و $x_n, \dots, x_2, x_1$ متغیرهای دامنه است.
- $x\ \theta\ y$ که و متغیرهای دامنه و θ عملگر مقایسه‌ای است $(\geq, >, \neq, =, <, \leq)$. لازم است صفات و دامنه-ای داشته باشند که بتوانند به وسیله θ با هم مقایسه شوند.
- $x\ \theta\ c$ که x متغیر دامنه، θ عملگر مقایسه و c ثابتی در دامنه‌ی صفتی است که x متغیر دامنه‌ای آن می‌باشد.

براساس قواعد زیر، فرمول‌ها را از اتم می‌سازیم:

- اتم یک فرمول است.
- اگر P_1 فرمول باشد، آنگاه $\neg P_1$ و (P_1) نیز فرمول است.
- اگر P_1 و P_1 فرمول باشند، آنگاه $P_1 \vee P_2$، $P_1 \wedge P_2$ و $P_1 \Rightarrow P_2$ نیز یک فرمول هستند.
- اگر $P_1(x)$ فرمولی در x باشد که متغیر x ، متغیر آزاد دامنه است، آنگاه عبارت زیر نیز فرمول است:
$\exists x \in r(P_1(x))\ and\ \forall x \in r(P_1(x))$

SQL:
- یک زبان برنامه‌نویسی نیست بلکه زبان پرس‌وجو نویسی است.
- یک زبانی توصیفی است.
- یک زبان بیانی($Declarative$) است، یعنی کاربر تنها می‌گوید چه می‌خواهد ولی چگونگی آن را مشخص نمی‌کند.
- حساسیت به حروف کوچک و بزرگ ندارد.

انواع داده در SQL:
- $Char(n)$: رشته کاراکتر با طول ثابت و مشخص n و اگر طول رشته‌ای برابر یک باشد لازم نیست ذکر شود.
- $Varchar(n)$: رشته کاراکتر با طول متغیر و حداکثر n .
- Int: زیرمجموعه محدودی از اعداد صحیح.
- $Small\ int$: زیرمجموعه‌ای از توع داده‌ای int .
- $Numeric(x,y)$: مجموعه اعداد حقیقی که دامنه آن توسط کاربر تعیین می‌شود.
  - طول کل عدد: x
  - طول اعشار: y
  - طول صحیح: $x-y$
- $Logical$: فقط دو مقدار T و F را می‌پذیرد. شبیه نوع داده‌ای $Boolean$ در برخی زبان‌های برنامه‌نویسی.
- $Date$: تاریخ (سال، ماه، روز) را قبول می‌کند و شبیه رشته کاراکتری نیست بلکه عملگر هم دارد.
- $Time$: وقت (ساعت، دقیقه، ثانیه) را می‌پذیرد و عملگر هم دارد.
- $Timestamp$: ترکیبی از دو نوع داده $date$ و $time$ است و در پایگاه داده زمانی کاربرد دارد.
- $Blob$:
  - کاربرد در ذخیره‌سازی داده‌های حجیم که ساختار ناشناخته‌ای برای $DBMS$ رابطه‌ای دارند.
  - پایگاه داده رابطه‌ای فقط می‌تواند ذخیره و بازیابی کند، امکان انجام هیچ نوع عملیات دیگری روی آنها وجود ندارد.

دسته‌بندی دستورات SQL:

۱. **دستورات *DDL*** زبان تعریف داده‌ها:
- $Create$
- $Alter$
- $Drop$

۲. **دستورات *DML*** زبان دستکاری داده‌ها:
- $Select$
- $Insert$
- $Update$
- $Delete$

۳. **دستورات تعریف حوزه دید کاربران (*View*)**
- $Create\ View$
- $Alter\ View$
- $Drop\ View$

۴. **دستورات *Authorization*** مربوط به تامین امنیت پایگاه داده
- $Grant$
- $Revoke$

۵. **دستورات *Integrity*** کنترل محدودیت‌های جامعیتی:
- $Check$
- $Create\ Assertion$
- $Trigger$

۶. **دستورات کنترل تراکنش‌ها (*Transaction Control*)**
- $Commit$
- $Exit$
- $Rollback$

۷. **دستورات *Embedded SQL* & *Dynamic SQL***

دستورات در زبان SQL:

۱. دستورات DDL:

۱-۱. دستور Create:

Create Database [name]

Create Table [نام جدول] ([نام ستون] [نوع داده‌ای] [options: Not Null, ...],
...,
Primary Key ([نام فیلد]),
Foreign Key ([نام فیلد]) *References* [نام جدول],
...,
Check (عبارت مورد نظر), → برای اعمال محدودیت جامعیتی
Unique → برای تعریف کلید ثانویه
On Delete / *On Update* } برای اعمال کنترل‌های جامعیت ارجاعی (*No Action, Cascade, Set Null, Set Default*)
)

۱-۲. دستور Alter:

تغییر ساختاری:
- **افزودن یک فیلد جدید:** *Alter table* نام جدول ***Add*** (نام ستون جدید)
- **حذف یک فیلد:** *Alter table* نام جدول ***Drop*** (نام ستون)
- **تغییر نوع داده‌ای فیلد:** *Alter table* نام جدول ***Modify*** نام ستون (نوع داده‌ای جدید)

۱-۳. دستور Drop:

یعنی حذف فیزیکی به همراه متعلقات. پس از حذف جدول، نه تنها سطرهای داخلی آن، بلکه تمام دیدهای تعریف شده (View) و شمای جدول نیز حذف می‌شوند. حذفی که قابل برگشت نیست.

Drop Database [name]

Drop Table [name]

۲. دستورات DML:

۲-۱. دستور Insert Into:

Insert Into *table* ***Values*** $(val_1, val_2, \dots)$

نکات:

- اگر هنگام درج اطلاعات در یک جدول، مقداری برای یکی از خصیصه‌ها وارد نشود، اگر آن خصیصه کلید اصلی نباشد و یا Not Null برای آن تعریف نشده باشد، DBMS بصورت خودکار مقدار آن خصیصه را Null قرار می‌دهد.
- در استفاده از این دستور می‌توان در مقابل نام جدولی که قرار است اطلاعات در آن درج شود نام ستون‌های مورد نظر را نوشت و به این ترتیب عملیات درج را به آن ستون‌ها محدود کرد.

۲-۲. دستور Delete:

Delete From *table* ***Where*** (*condition*)

۳-۲. دستور Update:

$$\boldsymbol{Update}\quad table\quad \boldsymbol{Set}\quad col_1 = val_1, \dots\quad \boldsymbol{Where}\quad (condition)$$

نکات:

- اگر کلید اصلی جدولی در جای دیگر به عنوان کلید خارجی تعریف شده باشد، یعنی جدول، مورد ارجاع جداول دیگر باشد امکان **حذف یا بروزرسانی** سطرهای جدول مرجع بدون پیش‌شرط وجود ندارد.در این شرایط **حذف یا بروزرسانی** هر سطری از جدول مرجع ممکن است موجب نقض قانون سوم جامعیت شود، در واقع در این حالت ممکن است جامعیت ارجاع به صورت **محتوایی** نقض شود. (استفاده از on delete cascade هنگام تعریف کلید خارجی، امکان حذف در این شرایط را فراهم می‌سازد).
- در صورتی که بخش شرط (**Where**) از دستور **Delete** حذف شود، کل محتویات جدول حذف خواهد شد.
- در صورتی که بخش شرط (**Where**) از دستور **Update** حذف شود، کل محتویات جدول در ستون مورد نظر بروزرسانی خواهد شد.

۴-۲. دستور Select:

برای استخراج رکوردهایی که دارای شرط خاصی هستند بکار می‌رود. در واقع این دستور ترکیبی از دو عملگر σ و π در جبر رابطه‌ای است در صورتی که در زبان SQL دستور Select شرطی نداشته باشد این دستور معادل π خواهد بود.

$$SELECT\quad Col_1, Col_2, \dots, Col_n$$
$$FROM\quad [table\ name]$$
$$WHERE\quad [condition]$$

۴-۱-۱. عملگر Order By: اگر بعد از Order By یک ستون گفته باشد باید آن ستون حتما بعد از Select آمده باشد در غیر اینصورت دستور اشکال ساختاری دارد.

$$SELECT\quad Col_1, Col_2, \dots, Col_n$$
$$FROM\quad [table\ name]$$
$$WHERE\quad [condition]$$
$$ORDER\ BY\quad \text{(نام یا شماره ستون)}$$

۴-۱-۲. عملگر Between و In :

- هر دو با not قابل استفاده است.
- عملگر between برای بررسی وجود یک مقدار در یک محدوده.
- عملگر in برای بررسی وجود یک مقدار در مجموعه‌ای از مقادیر.

نکته: ساختار Between به شکلی است که حد پایین و بالا نیز مورد بررسی قرار می‌گیرد و اگر بخواهیم حد بالا و پایین بررسی نشود، باید از علامتهای کوچکتر و بزرگتر استفاده شود.

۴-۱-۳. عملگرهای Like و Not Like : این عملگر جزو عملگرهای رشته‌ای است.
Case—Sensitive نیست ولی در نسخه‌های قدیمی بوده.
کاراکترهای جایگزین: % جایگزین مجموعه‌ای از کاراکترها
_ جایگزین یک کاراکتر

۴-۱-۴. عملگرهای Is Null و Is Not Null :
Is Null برای بررسی *null* بودن یک فیلد
Is Not Null برای بررسی عدم *null* بودن یک فیلد

نکات:

- هر وقت کلمه "ناشناخته" مشاهده شد منظور همان Null است.
- مقدار صفر برابر Null **نیست**. $(Null \neq 0)$
- به جای کلمه "is" از = **نمی‌توان** استفاده کرد. عبارت "= Null" **غلط** است.

۴-۱-۵. توابع تجمعی در SELECT:
$avg\ ( )$ و $sum\ ( )$ پارامتر ورودی حتما باید عدد باشد.
$\max ( )$، $\min ( )$، $count\ ( )$ پارامتر ورودی می‌تواند عدد یا غیر عدد باشد.

نکات:

- توابع تجمعی حتما باید با SELECT همراه باشند و تنها جایی که می‌توان بدون SELECT استفاده کرد، بعد از دستور Having است.
- اگر بخواهیم توابع تجمعی را بعد از WHERE بیاوریم، این امکان وجود دارد به شرط اینکه تابع را با یک SELECT دیگر همراه کرده و بصورت تو در تو نمایش دهیم.
- در پارامتر ورودی تابع count() حتما باید کلید اصلی باشد لذا بهتر است (پیشنهاد می‌شود) از علامت * به عنوان پارامتر ورودی استفاده شود یعنی count(*).
- تابع count(*) بصورت تنها باید بعد از SELECT ظاهر شود.
- در توابع تجمعی سطرهای تکراری حذف نمی‌شود.
- اگر بخواهیم از توابع تجمعی استفاده کنیم و سطرهای تکراری حذف شوند باید بعد از SELECT و قبل از نام تابع از کلمه کلیدی **DISTINCT** استفاده کنیم.
- در تابع count(*) استفاده از کلمه کلیدی **DISTINCT** جایز نیست.

۴-۱-۶. عملگر Group By : گروه‌بندی نتایج خروجی یک پرس‌وجو بر اساس ستون خاص. این دستور مقادیری را که بصورت تکراری در سطرهای مختلف تکرار شده است را فقط یکبار در خروجی به عنوان یک گروه نمایش می‌دهد و به این ترتیب می‌توان بر روی ستون‌های دیگر توابع تجمعی استفاده کرد.

نکات:

- دستور GROUP BY پس از اعمال شرایط در قسمت WHERE اجرا می شود.
- ستون‌هایی که جلوی GROUP BY قرار می‌گیرند باید حتما در بخش SELECT نیز وجود داشته باشند.

- اگر در بخش SELECT، صفاتی غیر از ستون‌هایی که بر روی آنها گروه‌بندی انجام شده است، وجود داشته باشند آنگاه این صفات حتما باید با یک تابع تجمعی همراه باشند.
- GROUP BY جزء آخرین بخش‌های دستور SELECT است و تنها بعد از آن HAVING یا ORDER BY می‌تواند ظاهر شود.

۴-۱-۷.عملگر Having : مربوط به زمانی است که گروه‌بندی انجام شده است و در این زمان می‌خواهیم شرط اعمال کنیم.

$WHERE \quad (1)$
$GROUP\ BY \quad (2)$
$HAVING \quad (3)$

نکته: همانطور که در قسمت توابع تجمعی گفته شد، تنها جایی که می‌توان توابع تجمعی را بدون SELECT استفاده کرد، بعد از دستور Having است.

۴-۱-۸.عملگرهای Union، Intersect و Except:

Relational Algebra	SQL
Union	Union
Intersect	Intersect
Minus (Set Difference)	Except

نکته: دستوری را که بتوان با IN و OR نوشت با Union هم می‌توان نوشت.

۴-۱-۹.نوشتن دستورات Select مبتنی بر پیوند جداول (پیوند سنتی، پیوند قدیمی):

انواع روش‌های پیوند معمولی
۱. پیوند به شرط تساوی فیلدهای مشترک کلیدی
۲. استفاده از $Select$ های تو در تو $(Nested\ Select)$

1.
$$\begin{cases} SELECT & S.S_{name} \\ FROM & S, SP \\ WHERE & S.S_{\#} = SP.S_{\#} \quad AND \quad SP.P_{\#} = 'P_2' \end{cases}$$

2.
$$\begin{cases} SELECT & S.S_{name} & & \\ FROM & S & & \\ WHERE & S_{\#} & IN & (SELECT \quad S_{\#} \\ & & & FROM \quad SP \\ & & & WHERE \quad P_{\#} = 'P_2') \end{cases}$$

در تمام مراجع، **روش ۱ از نقطه نظر سرعت و کارایی** بهتر است ولی **روش ۲ از نظر درک ظاهری** برنامه پیشنهاد می‌شود.

پرس‌وجوهای تو در تو: عبارت است از یک دستور SELECT که در داخل دستور SELECT دیگر بکار رود.

کاربرد عملگرهای پرس‌وجو تو در تو:

۱. عضویت در مجموعه $\begin{matrix} in \\ not\ in \end{matrix}\}$

۲. مقایسه مجموعه‌ای $\begin{matrix} contain \\ all \\ in \equiv any \equiv some \end{matrix}\}$

۳. بررسی وجود یا عدم وجود رکورد در مجموعه جواب پرس‌وجو $\begin{matrix} exist \\ not\ exist \end{matrix}\}$

۴. بررسی وجود یا عدم وجود رکورد تکراری در مجموعه جواب پرس‌وجو

نکات:

- هنگام استفاده از عملگرهای in و not in و some، نوع داده‌ای اعضای مجموعه و عضو مورد بررسی باید یکسان باشد، در غیر اینصورت خطا تولید می‌شود.
- دستور **some** با کلید واژه **حداقل** همراه است.
- دستور **all** با کلید واژه **تک‌تک** یا **همه** همراه است.
- دستورهای some و all با عملوندهای = ، <> (بجای ≠) ، > ، < ، <= ، => قابل استفاده است.
- دستور exist زمانی True است که مقدار select داخلی تهی نباشد (حداقل یک سطر داشته باشد).
- عملگرهای exist و not exist بازدهی خوبی ندارند.

پرس‌وجوی به‌هم پیوسته (Correlated Query):

به پرس‌وجویی که در آن، زیرپرس‌وجو (Sub Query) در بخش شروطش به پرس‌وجوی اصلی مرتبط می‌شود، پرس‌وجوی به‌هم پیوسته گفته می‌شود.

$$\begin{array}{lll} SELECT & C_{name} & \\ FROM & table_1 & \\ WHERE & EXISTS & (SELECT \quad * \\ & & FROM \quad table_2 \\ & & WHERE \quad \underline{table_1.C_{name} = table_2.C_{name}}) \end{array}$$

نام‌گذاری مجدد در SQL:

در SQL امکان نام‌گذاری مجدد، هم برای یک جدول، هم برای ستون‌های یک جدول وجود دارد. استفاده از نام‌گذاری مجدد جداول در برخی موارد اختیای بوده و با هدف ساده‌سازی پرس‌وجو انجام می‌شود.

۳. دستورات حوزه دید کاربران (View):

۳-۱. ایجاد جدول مجازی در SQL:

$Create\ view$ نام دید (نام ستون ۱ , نام ستون ۲ , ...)

As (عبارت مورد جستجو (یک دستور SQL)

همانطور که در قسمت لایه‌های ANSI_SPARC توضیح داده شد:

ایجاد $view$ ها ⟵ لایه خارجی $(External\ Level)$ ⟵ بالاترین لایه (بیرونی‌ترین) ⟵ $ANSI\_SPARC$

نکات:

- جدول مجازی (View) استقلال وجودی ندارد و به جداول پایه متکی است، بطوریکه هر عملیات که روی view انجام می‌شود در واقع عملیاتی است که بر روی جداول پایه آن اعمال می‌شود.
- بصورت پیش‌فرض فقط امکان خواندن از جدول مجازی برای کاربر فراهم است. هرگونه پردازش نظیر درج، حذف و بروزرسانی روی جداول مجازی با مشکلات و هزینه‌های زیادی مواجه می‌شود و در برخی شرایط غیر ممکن است لذا **جدول مجازی (View)، Updatable نیست مگر تحت شرایط زیر:**
  ۱. View فقط از یک جدول ایجاد شده باشد.
  ۲. در ایجاد view از select distinct استفاده نشده باشد.
  ۳. در ایجاد view از توابع تجمعی یا آماری استفاده نشده باشد.
  ۴. در ایجاد view از دستور group by استفاده نشده باشد.
- در صورتی که در عبارت مورد جستجو از * Select استفاده شود، نیازی به ذکر نام ستون‌ها بعد از "نام دید" نیست.

تعریف index (شاخص) در SQL:

استفاده از index در پایگاه داده بیشتر با هدف افزایش سرعت مطرح می‌شود. در پایگاه داده از ساختارهای درختی استفاده می‌شود که این ساختارها توسط طراح پایگاه داده و DBMS ایجاد می‌شود و به عنوان یک نمونه دیگر از فراداده‌ها در کاتالوگ سیستم ذخیره می شود.

$Create\ index$ نام شاخص نام جدول (نام ستونی که می‌خواهد $index$ شود)

نکته: یکی دیگر از کاربردهای index زمانی است که می‌خواهیم یک فیلد در یک جدل مقدار تکراری قبول نکند، در این صورت دستور index ساختارش به شکل زیر تغییر می‌کند:

$Create\ \ unique\ \ index$ نام شی on (نام ستون) نام جدول

یادآوری: ایجاد viewها و indexها باعث تغییر در **لغت‌نامه** و طبیعتا تغییر در بخش **کاتالوگ سیستم** در پایگاه داده می‌شود.

۴. دستورات تعریف محدودیت‌های امنیتی (Authorization) در SQL:

$Grant$ ⟨لیست مجوزها⟩ on ⟨نام جدول⟩ $\underline{\boldsymbol{to}}$ ⟨نام کاربران⟩

$Revoke$ ⟨لیست مجوزها⟩ on ⟨نام جدول⟩ $\underline{\boldsymbol{from}}$ ⟨نام کاربران⟩

انواع مجوزها:

۱. مجوز خواندن اطلاعات: $Select$
۲. مجوز درج اطلاعات جدید: $Insert$
۳. مجوز بروزرسانی اطلاعات: $Update$
۴. مجوز حذف اطلاعات: $Delete$

نکته: این مجوزها را می‌توان برای ستون‌ها (فیلدها) هم اعمال کرد و در صورتیکه نام ستون ذکر نشود یعنی همه ستون‌ها.

۵. تعریف محدودیت‌های جامعیتی (Integrity) در SQL:

یعنی **بی‌نقصی** تراکنش‌های پایگاه داده و پیروی کردن آنها از مقررات تعریف شده در سیستم.

۵-۱. اعمال Check هنگام ایجاد جدول: در قسمت create table اشاره شد.

۵-۲. عمال Check بعد از ایجاد جدول: این دستور بعد از ایجاد جدول حتی بعد از ورود داده در جدول (Data Entry) هم استفاده می‌شود.

Create Assertion نام محدودیت شرح محدودیت

یا

Create Trigger نام محدودیت شرح محدودیت

نکته: در هنگام استفاده از دستور Assertion، در قسمت شرح محدودیت همواره دستور SELECT به همراه EXIST و NOT EXIST ظاهر می‌شود:

Create Assertion test ***Exist*** (*Select* * *From student*
Where $\boldsymbol{avg > 0}$)

یا

Create Assertion test ***Not Exist*** (*Select* * *From student*
Where $\boldsymbol{avg < 0}$)

انواع پیوند در SQL:

- *Cross Join*
- *Inner Join*:
  - ***Natural*** *Inner Join*
  - *Inner Join* ***on***
  - *Inner Join* ***Using***
- *Outer Join*:
  - ***Natural*** *Left/Right Outer Join*
  - *Left/Right Outer Join* ***on***
  - *Left/Right Outer Join* ***Using***

۱. **پیوند Cross Join:** معادل ضرب دکارتی است. (پایه همه پیوندها)

Select * *From* $table_1$ ***Cross Join*** $table_2$

۲. **پیوند Inner Join:** در واقع همان ضرب دکارتی است که برای بیان شروط آن از ۳ دستورالعمل ویژه زیر می‌توان استفاده کرد:

1) $table_1$ ***Natural*** *Inner Join* $table_2$

گفتن Natural باعث می‌شود فیلد مشترک کلیدی فقط یکبار در خروجی ظاهر شود و همچنین فقط سطرهایی از پیوند دو جدول در خروجی ظاهر شوند که مقدارشان در ستون مشترک با هم برابر است.

2) $table_1$ *Inner Join* $table_2$ ***On*** مساوی قرار دادن فیلدهای مشترک کلیدی

باعث می‌شود فیلد مشترک کلیدی هر دو بار در خروجی ظاهر شود که **بدتر است** اما کارایی آن در فهم راحت و امکان استفاده از AND در شرط می‌باشد.

3) $table_1$ $Inner\ Join$ $table_2$ $\boldsymbol{Using}$ (فیلد مشترک کلیدی)

همانند Natural باعث می‌شود فیلد مشترک کلیدی یکبار در خروجی بیاید. دستور Using فقط زمانی می‌تواند استفاده شود که فیلد مشترک کلیدی در هر دو جدول هم‌نام باشد و همچنین در استفاده از دستور Inner Join بیان شروط الزامی نمی‌باشد.

۳. پیوند Outer Join: در syntax ها می‌توان Outer را ذکر نکرد.

1) $table_1$ $\boldsymbol{Natural}$ $Inner\ Join$ $table_2$
2) $table_1$ $Inner\ Join$ $table_2$ $\boldsymbol{On}$ مساوی قرار دادن فیلدهای مشترک کلیدی
3) $table_1$ $Inner\ Join$ $table_2$ $\boldsymbol{Using}$ (فیلد مشترک کلیدی)

عملگر تقسیم در SQL: اگر در سوال قید **همه** استفاده شود از عملگر تقسیم استفاده می‌شود.

روش‌های پیاده‌سای عملگر تقسیم در SQL:
۱. استفاده از تابع $count()$ و دستور $Group\ By$
۲. ترکیب دستورات $Not\ Exist$ و $Except$
۳. ذکر دو بار متوالی دستور $Not\ Exist$

انواع جدول در SQL:
جدول پایه: با دستور $Create\ Table$ ایجاد می‌شود.
جدول میانی $(Intermediate\ Table)$: $DBMS$ آن را ایجاد می‌کند.
جدول مجازی: با دستور $Create\ View$ ایجاد می‌شود.

نکته: جدول میانی را سیستم SQL یعنی DBMS در Select های تو در تو ایجاد می‌کند و هنگام نیاز آن را بکار می‌گیرید و اگر نیاز نباشد آن را حذف می‌کند.

نکته: در SQLهای نسخه 2.0 و 3.0 شی‌گرایی پشتیبانی می‌شوند لذا به آنها **OSQL** هم می‌گویند.

وابستگی (Dependency):

یکی از اهداف پایگاه داده ← کاهش افزونگی ← علت وجود افزونگی؟ ← وجود برخی از وابستگی‌ها

انواع وابستگی:

۱. **وابستگی تابعی** $(\boldsymbol{FD}: Functional\ Dependency)$

۲. **وابستگی تابعی کامل** $(\boldsymbol{FFD}: Full\ Functional\ Dependency)$

۳. **وابستگی چند مقداری** $(\boldsymbol{MVD}: Multi\ Valued\ Dependency)$

۴. **وابستگی پیوندی** $(\boldsymbol{JD}: Join\ Dependency)$

وابستگی تابعی (FD):

در رابطه R، صفت B با صفت A وابستگی تابعی دارد اگر و فقط اگر در طول حیاط رابطه R به ازای هر مقدار از صفت A فقط یک مقدار از صفت B متناظر باشد.

A	$\longrightarrow$	B
دترمینان		وابسته
$Determinant$		$Dependent$

و می‌خوانیم، B با A وابستگی تابعی دارد یا A صفت B را تعیین می‌کند.

نکات:

- سایر مولفه‌های غیرکلیدی، به کلید وابستگی دارند و همچنین مولفه‌های غیرکلیدی به هم وابستگی ندارند.
- FDها در واقع محدودیت جامعیت را نشان می‌دهند و بنابراین DBMS باید آنها را در نظر گرفته و اعمال کند.
- در مفهوم FD در یک رابطه، خاصیت جابجایی الزاما برقرار نیست. (اگر $A \to B$ لزومی ندارد $B \to A$).
- طرفین یک عبارت وابستگی می‌تواند ترکیبی از صفات نیز باشد. $(A, B) \to (C, D)$
- وابستگی تابعی بین صفات یک رابطه یک مفهوم مستقل از زمان است.
- در رابطه تمام کلید (All Key) می‌توان اثبات کرد که بین اجزای کلید، وابستگی تابعی برقرار نیست.

وابستگی تابعی بدیهی (Trivial Functional Dependency):

اگر عضوی که در سمت راست وابستگی قرار دارد زیرمجموعه سمت چپ باشد، وابستگی بدیهی است.

$$\alpha \to \beta\ , \beta \subseteq \alpha$$

وابستگی تابعی بخشی (جزئی) (Partial Functional Dependency):

وابستگی صفات یا مولفه‌های غیرکلیدی به بخشی از کلید.

مولفه غیرکلیدی ← بخشی از کلید

وابستگی انتقالی (Transitive Dependency):

وابستگی یک مولفه غیرکلیدی به مولفه غیرکلیدی دیگر.

مولفه غیرکلیدی ← مولفه غیرکلیدی

وابستگی معکوس (Reverse Dependency):

وابستگی یک مولفه کلیدی به مولفه دیگر (کلیدی یا غیرکلیدی).

مولفه کلیدی ⟵ مولفه کلیدی

مولفه کلیدی ⟵ مولفه غیرکلیدی

مجموعه بستار وابستگی (پوششی وابستگی):

اگر F یک مجموعه از وابستگی تابعی باشد، مجموعه تمامی وابستگی‌هایی که از مجموعه F استنتاج می‌شود بنام مجموعه پوششی وابستگی (بستار وابستگی) معرفی می‌شود. در این مجموعه تمامی اعضای مجموعه پوششی وابستگی و خود مجموعه وابستگی دیده می‌شود.

قوانین (قواعد) آرمسترانگ جهت بدست آوردن مجموعه پوششی (بستار) وابستگی:

۱. **قانون بازتاب:** اگر B زیرمجموعه A باشد آنگاه $A \to B$.
۲. **قانون افزایش (بسط‌پذیری):** اگر $A \to B$ و C یک صفت باشد آنگاه: $AC \to BC$.
۳. **قانون انتقال تعدی:** اگر $A \to B$ و $B \to C$ آنگاه $A \to C$.
۴. **قانون اجتماع:** اگر $A \to B$ و $A \to C$ آنگاه $A \to BC$.
۵. **قانون تجزیه (عکس اجتماع):** اگر $A \to BC$ آنگاه $A \to B$ و $A \to C$.
۶. **قانون ترکیب:** اگر $A \to B$ و $C \to D$ آنگاه $AC \to BD$.
۷. **قانون خودتعیینی:** $A \to A$ یا $AB \to AB$ یا $ABC \to ABC$.
۸. **قانون شبه‌تعدی:** اگر $A \to B$ و $BC \to D$ آنگاه $AC \to D$.
۹. **اتحاد کلی:** اگر $A \to B$ و $AB \to C$ آنگاه $A \to C$.

تذکر: با این قوانین علاوه بر بدست آوردن مجموعه بستار وابستگی، می‌توان بستار صفات کلید کاندید و ابرکلید را بدست آورد.

بستار مجموعه‌ای از صفات (خصیصه‌ها):

اگر R یک رابطه و Aمجموعه F مجموعه وابستگی‌های تابعی تعریف شده روی آن رابطه باشد و همچنین A زیرمجموعه‌ای از خصیصه‌های رابطه R باشد، بستار A (A^+) عبارت است از همه خصیصه‌هایی که با استفاده از مجموعه F قابل استنتاج از اعضای A هستند.

نکات:

- در مجموعه بستار صفات (خصیصه‌ها)، صفت یا ترکیب صفاتی کلید کاندید می‌باشد که بتواند سایر صفات دیگر رابطه را تعیین کند. در غیر اینصورت آن صفت یا ترکیب آنها، کلید کاندید نمی‌باشد.
- طبق تعریف کلید کاندید صفتی است که سمت چپ وابستگی‌ها باشد و کلید کاندید باید قادر به تولید همه خصیصه ها باشد.
- طبق توصیه مرجع استفاده از قانون تجزیه در ابتدا، سرعت رسیدن به کلید کاندید را بیشتر می‌کند.

- اگر صفتی در رابطه R باشد که در مجموعه وابستگی حضور نداشته باشد، در این حالت کلید کاندید ترکیبی بوده و یک بخش از این ترکیب همان صفتی است که در مجموعه وابستگی حضور ندارد.
- اگر بخواهیم بررسی کنیم که دو مجموعه معادل هستند، باید سمت راست تمامی وابستگی‌های یک مجموعه در سمت راست وابستگی‌های مجموعه دیگر حضور داشته باشند.

تعریف کلید کاندید با استفاده از مفهوم وابستگی تابعی:

اگر R یک رابطه و F مجموعه وابستگی‌های تابعی موجود روی آن رابطه باشد، در این صورت A (زیرمجموعه‌ای از خصیصه R) به عنوان یک کلید کاندید برای R محسوب می‌شود اگر و فقط اگر

الف) همه اعضای A مستقل باشند (هیچ یک از اعضای A قابل استنتاج از روی سایرین نباشد)

ب) بستار A (A^+) همه خصیصه‌های رابطه R را شامل شود.

نحوه شناسایی کلیدهای کاندید یک رابطه به کمک مجموعه وابستگی‌های تابعی آن:

بر اساس تعرف فوق برای R، F و کلید کاندید، جهت شناسایی کلیدهای کاندید در یک رابطه باید به نکات زیر توجه کرد:

۱- اگر خصیصه‌ای در سمت راست هیچ یک از وابستگی‌های تابعی در F نباشد یعنی قابل استنتاج از روی هیچ خصیصه دیگری نیست و آن خصیصه حتما **عضو** کلید کاندید است.

۲- اگر مجموعه خصیصه‌هایی که در سمت راست هیچ وابستگی تابعی در F نیستند به تنهایی قادر به تولید کلیه خصیصه‌های رابطه باشند این مجموعه تنها کلید کاندید رابطه محسوب می‌شود.

۳- اگر مجموعه خصیصه‌هایی که در سمت راست هیچ وابستگی تابعی در F نیستند به تنهایی قادر به تولید کلیه خصیصه‌های رابطه نباشند، باید به دنبال کمترین تعداد خصیصه‌های دیگر بود که با اضافه شدن به مجموعه پیش‌گفته قادر به تولید کلیه خصیصه‌های رابطه باشند.

خصیصه کمکی باید از بین خصیصه‌هایی انتخاب شوند که در سمت چپ وابستگی‌های تابعی حضور دارند (قدرت تولید خصیصه‌های دیگر را دارند).

۴- اگر همه خصیصه‌های یک رابطه در سمت راست وابستگی‌های تابعی در F باشند. شناسایی کلید کاندید با استفاده از سه نکته قبل امکان‌پذیر نیست. در چنین شرایطی باید ابتدا خصیصه‌هایی را که در سمت چپ هیچ وابستگی تابعی حضور ندارند (قدرت تولید هیچ خصیصه دیگری را ندارند) کنار گذاشت. سپس با بررسی بستار هر زیرمجموعه‌ای از خصیصه‌های باقی‌مانده کلید کاندید را مورد شناسایی قرار داد.

۵- هر ترکیبی از خصیصه‌های مستقل (خصیصه‌هایی که از یکدیگر قابل استنتاج نیستند) که بتواند یک کلید کاندید را بدهد، خودش یک کلید کاندید است.

مجموعه وابستگی بهینه (Optimal):

مجموعه‌ای است که طرفین وابستگی‌های آن تک صفتی باشد.

مجموعه وابستگی کاهش‌ناپذیر (حداقل، کهینه) (Minimal):

اگر در مجموعه وابستگی بهینه عضوهای اضافی را حذف کنیم، مجموعه ای بنام مجموعه وابستگی حداقل ساخته می‌شود که این مجموعه، عضو اضافه ندارد.

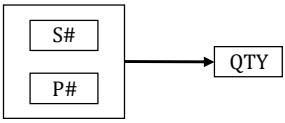
تعریف وابستگی کامل (Full Functional Dependency):

صفت y با ترکیب x_1 تا x_n وابستگی تابعی کامل دارد اگر y با ترکیب x_1 تا x_n وابستگی کامل داشته باشد ولی y با هر کدام از x_1 تا x_n به تنهایی وابستگی کامل داشته باشد.

دیاگرام وابستگی تابعی (FD Diagram):

شکل گرافیکی نمایش وابستگی‌ها است و یکی از کاربردهای اصلی آن در مبحث نرمال‌سازی می‌باشد.

$$(S\#, P\#) \rightarrow QTY$$



نرمال‌سازی (Normalization):

در بانک‌های اطلاعاتی رابطه‌ای، نرمال‌سازی فرآیند تجریه یک رابطه به دو یا چند رابطه است که پیش از پیاده‌سازی جداول انجام می‌شود و بدین صورت افزونگی‌ها و آنومالی‌ها کاهش خواهند یافت.

نرمال‌سازی مطلوب:

یک اصل مهم در بانک‌های اطلاعاتی رابطه‌ای رعایت شرط Non-Loss-Join می‌باشد یعنی اگر یک رابطه نرمال شود و دوباره جداول آن به هم پیوند شوند چیزی از دست نرود و همان دادهای اولیه حفظ شود و البته چیزی نیز اضافه نشود.

اهداف نرمال‌سازی:

- **کاهش** برخی از افزونگی‌های داده‌ای
- مدل‌سازی بهتر دنیای واقعی
- **کاهش** هزینه اعمال برخی از محدودیت‌های جامعیتی
- **کاهش** آنومالی‌ها

لایه‌های نرمال‌سازی:

- شکل اول نرمال $(1NF)$
- شکل دوم نرمال $(2NF)$
- شکل سوم نرمال $(3NF)$
- شکل نرمال $Boyce/Codd$ $(BCNF)$
- شکل چهارم نرمال $(4NF)$
- شکل پنجم نرمال $(5NF)$

فرم اول نرمال‌سازی (1NF):

نخستین مرحله نرمال‌سازی تبدیل جداول به فرم نرمال اول است. برای قرار گرفتن یک جدول در فرم نرمال اول باید ۳ شرط زیر **همزمان** برقرار باشد:

۱- رابطه حداقل دارای یک کلید کاندید باشد.

۲- همه صفات رابطه بصورت غیرقابل تجزیه (atomic) باشند یعنی رابطه باید فاقد صفات مرکب باشد.
۳- همه صفات رابطه تک مقداری باشند یعنی رابطه فاقد صفات چند مقداری باشند.

مشکل اساسی لایه اول نرمال،ایجاد **وابستگی جزئی (بخشی)** است.

فرم دوم نرمال‌سازی (2NF):

رابطه R در لایه دوم نرمال است اگر و فقط اگر در فرم نرمال اول باشد و همچنین فاقد وابستگی بخشی (جزئی) باشد.

نحوه تبدیل به 2NF:

برای انجام این کار کافی است مولفه‌های غیرکلیدی را که به بخشی از کلید وابستگی دارند را به عنوان یک رابطه جدید تعریف کنیم و کلید اصلی این جدول را مجددا در جدول جدید به عنوان کلید خارجی معرفی نماییم. بدین صورت جداول حاصل فاقد وابستگی‌های بخشی خواهند شد.

نکته: اگر رابطه‌ای در 1NF باشد و در آن رابطه تمام کلیدهای کاندید تک صفتی باشند آنگاه این جدول فاقد وابستگی بخشی است و در 2NF قرار می‌گیرید.

فرم سوم نرمال‌سازی (3NF):

رابطه R در لایه سوم نرمال است اگر و فقط اگر در 2NF باشد و همچنین فاقد وابستگی انتقالی باشد.

نحوه تبدیل به 3NF:

برای تبدیل ابتدا صفت واسطه را شناسایی میکنیم و جدول را از طریق این صفت واسطه تجزیه می‌کنیم. صفت واسطه در یک رابطه کلید کاندید (دترمینان) و در رابطه دیگر کلید خارجی (وابسته) است.

تجزیه مطلوب در نرمال‌سازی:

بطور کلی یک تجزیه مطلوب دو ویژگی اصلی دارد:

۱- **عدم گمشدگی:** اگر یک رابطه را به دو رابطه تجزیه کردیم و دوباره آنها را به هم پیوند دادیم سطری از دست نرود و سطر اضافه پدید نیاید.
۲- **حفظ وابستگی:** یعنی هیچ یکی از وابستگی‌های تابعی موجود در جدول اولیه در اثر تجزیه از دست نرود.

ضوابط ریسانن (Rissanen) برای تجزیه مطلوب:

تجزیه یک رابطه مانند رابطه R به دو رابطه R_1 و R_2 زمانی مطلوب است که R_1 و R_2 مستقل از یکدیگر باشند. رابطه R_1 و R_2 مستقل از یکدیگرند اگر و فقط اگر دو شرط زیر برقرار باشد:

۱- صفت مشترک در دو رابطه، **حداقل** در یکی از آنها کلید کاندید باشد
۲- تجزیه انجام شده حافظ وابستگی‌های تابعی باشد.

قضیه هیث برای تجزیه مطلوب (Heath):

حالت اول: در صورتیکه رابطه $R(A,B,C)$ وابستگی $A \to B$ مفروض باشد آنگاه می‌توان رابطه R را به دو رابطه $R_1(A,\cdots)$ و $R_2(A,\cdots)$ تجزیه کرد.

حالت دوم: در صورتیکه رابطه $R(A,B,C)$ و وابستگی‌های $A \to B$ و $B \to C$ مفروض باشد آنگاه می‌توان رابطه R را به دو رابطه $R_1(A,B)$ و $R_2(A,C)$ تجزیه کرد.

لایه BCNF:

شکل خاصی از لایه 3NF یا تعریف سختگیرانه از 3NF. (ANSI لایه جدا محسوب می‌کند).

تعریف اول (جدید): رابطه R، BCNF است اگر و فقط اگر 3NF باشد و فاقد وابستگی معکوس باشد.

تعریف دوم (قدیم): رابطه R، BCNF است اگر و فقط اگر 3NF باشد و با داشتن تمام وابستگی‌های تابعی، هر دترمینان در وابستگی‌ها ابرکلید یا کلید کاندید باشد.

نکات:

- در لایه BCNF اگر جدولی در این لایه قرار گیرد تمام وابستگی‌های تابعی از جمله وابستگی بخشی، انتقالی و معکوس که همگی ریشه در وابستگی تابعی دارند حذف شده است و تنها وابستگی‌ای که در این لایه باقی می‌ماند وابستگی جدیدی بنام وابستگی چندمقداری می‌باشد.
- جدولی که بصورت All Key باشد فاقد وابستگی‌های بخشی، انتقالی و معکوس است در نتیجه هر جدول All Key، BCNF است و همچنین هر جدول باینری (دو ستونی) نیز BCNF است.
- اگر جدولی سه شرط زیر را همزمان داشته باشد ممکن است در لایه سوم نرمال باشد اما در BCNF نباشد:
  ۱- رابطه دارای چندین کلید کاندید باشد.
  ۲- کلیدهای کاندید ترکیبی باشند.
  ۳- همه کلیدهای کاندید حداقل در یک صفت با یکدیگر اشتراک داشته باشند.
- در صورتیکه به لایه BCNF برسیم افزونگی‌هایی که منشا آنها وابستگی تابعی (FD) بوده است حذف شده است. در صورتیکه باز هم افزونگی موجود باشد منشاء آن افزونگی، وجود یک نوع وابستگی است که به آن وابستگی چندمقداری (MVD) می‌گویند.

وابستگی چندمقداری (MVD):

در رابطه R صفت B با صفت A وابستگی چندمقداری دارد اگر و فقط اگر در طول حیاط رابطه R به ازای هر مقدار از صفت A چندمقدار (بیشتر از یک) از صفت B متناظر باشد.

$$A \twoheadrightarrow B$$

B با A وابستگی چندمقداری دارد

A، B را به صورت چندگانه تعیین می‌کند.

نکات:

- در وابستگی چندمقداری در صورتیکه به ازای هر مقدار از صفت A فقط یک مقدار از صفت B بدست آید در این صورت تعریف وابستگی تابعی را خواهیم داشت بنابراین وابستگی تابعی نوع خاص از وابستگی چندمقداری است.

- هر وابستگی تابعی حتما یک وابستگی چندمقداری است اما وابستگی چند مقداری **ممکن** است معادل یک وابستگی تابعی نباشد.

قوانین آرمسترانگ در مورد وابستگی‌های چندمقداری:

۱- اگر $\beta \subset \alpha$ آنگاه $\alpha \twoheadrightarrow \beta$

۲- اگر $\alpha \twoheadrightarrow \beta$ و $\alpha \twoheadrightarrow \gamma$ آنگاه $\alpha \twoheadrightarrow \gamma\beta$

۳- اگر $\alpha \twoheadrightarrow \beta$ و $\beta \twoheadrightarrow \gamma$ آنگاه $\alpha \twoheadrightarrow \gamma$

۴- اگر $\alpha \twoheadrightarrow \beta$ و $\alpha \twoheadrightarrow \gamma$ آنگاه $\alpha \twoheadrightarrow \gamma - \beta$ و $\alpha \twoheadrightarrow \beta - \gamma$

۵- اگر $\alpha \twoheadrightarrow \beta$ و γ یک صفت از همان رابطه باشد آنگاه $\alpha\gamma \twoheadrightarrow \beta\gamma$

۶- اگر $x \to y$ آنگاه $x \twoheadrightarrow y$ ولی عکس آن صادق نیست.

۷- اگر $\alpha \twoheadrightarrow \beta$ و $\alpha \twoheadrightarrow \gamma$ آنگاه $\alpha \twoheadrightarrow \beta \cap \gamma$

۸- اگر $A \twoheadrightarrow B$ و $A \twoheadrightarrow C$ آنگاه می‌توان نتیجه گرفت که $A \twoheadrightarrow B \mid C$ در این صورت می‌توان گفت که MVD از نوع FD است.

قانون تجزیه مطلوب در وابستگی‌های چندمقداری (قضیه فاگین fagin):

اگر در رابطه $R(\alpha,\beta,\gamma)$ وابستگی‌های $\alpha \twoheadrightarrow \beta$ و $\alpha \twoheadrightarrow \gamma$ را داشته باشیم آنگاه تجزیه رابطه R به دو رابطه $R_1(\alpha,\beta)$ و $R_2(\beta,\gamma)$ بهتر است از تجزیه رابطه R به دو رابطه $R_3(\alpha,\beta)$ و $R_4(\alpha,\gamma)$ است.

فرم چهارم نرمال‌سازی (4NF):

تعریف اول: رابطه R در لایه چهارم نرمال است اگر و فقط اگر BCNF باشد و پدیده MVD در آن نباشد و اگر پدیده MVD در آن موجود باشد از نوع FD باشد.

تعریف دوم: رابطه R در لایه چهارم نرمال است اگر و فقط اگر به ازای هر یک از وابستگی‌های چندمقداری نظیر $\alpha \twoheadrightarrow \beta$ حداقل یکی از دو شرط زیر برقرار باشد:

۱- وابستگی $\alpha \twoheadrightarrow \beta$ بدیهی باشد.

۲- سمت چپ این وابستگی چندمقداری ابرکلید باشد.

جدول تجزیه‌پذیر مرتبه n:

در برخی موارد نمی‌توان یک جدول را بدون گمشدگی به دو جدول تقسیم کرد در حالی که می‌توان آن را بدون گمشدگی به **بیش از دو جدول** تجزیه نمود. اگر جدولی چنین خاصیتی را داشته باشد به آن **تجزیه‌پذیر مرتبه n** (n-decomposable) می‌گویند ($n \geq 3$).

فرم پنجم نرمال‌سازی (5NF):

رابطه R در لایه پنجم نرمال است اگر و فقط اگر به ازای همه وابستگی‌های پیوندی (الحاقی) حداقل یکی از دو شرط زیر برقرار باشد:

۱- این وابستگی‌های پیوندی بدیهی باشند.

۲- این وابستگی‌های پیوندی مبتنی بر کلید کاندید باشند.