موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

پایگاه داده‌ها

فصل اول: مفاهیم پایه (درسنامه)

(ویژه کنکور ارشد ۱۴۰۵)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتب مرجع

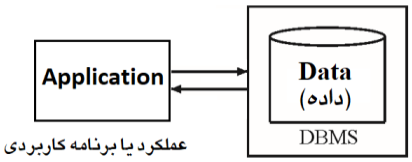
آبراهام سیلبرشاتز، راما کریشنان، رامز المصری، سی جی دیت، توماس کانلی و کارولین بگ

ارسطو خلیلی‌فر

۱ مفاهیم پایه

مقدمه

درس پایگاه داده با چه چیزی سروکار دارد؟ با data، چرا باید درس پایگاه داده را بخوانیم؟ چون انسان نیاز به **ذخیره و بازیابی اطلاعات** به صورت کامپیوتری و باسرعت زیاد دارد. این نیاز از طریق **ساخت محصول نرم‌افزاری** برای یک **محیط عملیاتی** (محیط زیست محصول نرم‌افزاری) مرتفع می‌گردد. این محصول نرم افزاری مثل تولیدکنندگان قطعات قطعا از دو بخش اساسی **داده** و **عملکرد** (برنامه کاربردی یا واسط کاربر یا اپلیکیشن) تشکیل شده است. بنابراین ما با یک سیستم ذخیره و بازیابی اطلاعات سروکار داریم. اما قبل از هر ذخیره‌سازی ابتدا ما باید یک جایی، ظرفی، چیزی مثل **جدول** داشته باشیم که بتوانیم **داده** را در آن ذخیره کنیم که به کل آن جداول برای یک برنامه کاربردی خاص می‌گوییم **پایگاه داده** تا در نهایت بر حسب لزوم بتوانیم **اطلاعات** را از طریق پرس‌وجو از آن **بازیابی** کنیم. برای مثال ابتدا داده‌های همه تولیدکنندگان در سیستم تولیدکنندگان قطعات داخل جداول پایگاه داده ذخیره می‌شود، سپس از طریق پرس‌وجو می‌گوییم لیست تولیدکنندگان ساکن شهر شیراز رو به من بده. به تفاوت داده و اطلاعات دقت کنید.



انواع نرم‌افزارها

در یک دسته‌بندی کلی برای نرم‌افزارهای موجود، می‌توان آنها را به دو گروه اصلی زیر تقسیم نمود:

نرم‌افزارهای کاربردی

برنامه‌هایی که برای رفع نیازهای کاربران کامپیوتر نوشته می‌شود، به بیان دیگر نرم‌افزارهای کاربردی به‌طور **مستقیم به انسان** سرویس می‌دهند. مانند نرم‌افزارهای حسابداری و نرم‌افزار فرهنگ لغات.

نرم‌افزارهای سیستمی

برنامه‌هایی که برای بهره‌برداری از سخت‌افزار یا سرویس‌دادن به سایر برنامه‌ها نوشته شده‌اند. به‌بیان دیگر نرم‌افزارهای سیستمی به طور مستقیم به نرم افزارهای دیگر و به طور **غیرمستقیم به انسان** سرویس می‌دهند. مانند سیستم عامل‌ها، کامپایلرها و DBMS.

تعریف داده (Data)

داده و **اطلاعات** دو اصطلاحی هستند که اغلب به جای هم استفاده می‌شوند، اما تفاوت‌های مهمی بین آن‌ها وجود دارد، درک این تفاوت‌ها برای بهره‌برداری موثر از داده‌ها بسیار مهم است.

داده (Data) چیست؟

- **حقایق خام و بی‌معنی**: داده‌ها در واقع حقایق خام و بی‌معنایی هستند که به تنهایی نمی‌توانند به ما اطلاعات مفیدی بدهند.
- **نمادها و اعداد**: این داده‌ها می‌توانند شامل اعداد، حروف، نمادها یا هر نوع دیگری از نمادهای قابل ثبت باشند.
- **بدون ساختار مشخص**: داده‌ها لزوماً ساختار مشخصی ندارند و ممکن است به صورت تصادفی یا بدون نظم خاصی جمع‌آوری شده باشند.

مثال: شماره دانشجویی یک دانشجو، کد ملی یک فرد، قیمت یک کالا

مطابق مفاهیم درس ذخیره و بازیابی اطلاعات، داده‌ها را می‌توان در **ساختارهای داده‌ای** مختلفی **سازماندهی** کرد:

۱- فایل Pile (فایل درهم): درج رکوردهای داده‌ای بدون هیچگونه نظم، که نتیجه می‌شود **ذخیره‌سازی سریع** چون هیچ نظمی ندارد، اما **بازیابی کند** است چون جستجوی خطی با مرتبه اجرایی O(n) دارد. بدین معنی که زمان اجرای این الگوریتم متناسب با اندازه ورودی (n) به صورت خطی افزایش می‌یابد. هرچه اندازه لیست بزرگ‌تر باشد، به طور متوسط، تعداد مقایسه‌ها نیز بیشتر می‌شود. شکل مقابل گویای مطلب است:

c2
d2
z3
a2
b1
c1
a1

a1
a2
b1
c1
c2
d2
z3

۲- فایل Sequential (فایل ترتیبی): درج رکوردهای داده‌ای به صورت مرتب‌شده، که نتیجه می‌شود **ذخیره‌سازی کند** چون به طور مرتب‌شده باید درج شود، اما **بازیابی سریع** است چون جستجوی دودویی با مرتبه اجرایی O(log n) دارد. بدین معنی که با افزایش اندازه لیست، زمان اجرای این الگوریتم به صورت لگاریتمی افزایش می‌یابد. به عبارت ساده‌تر، با دوبرابر شدن اندازه لیست، زمان اجرای جستجوی دودویی تنها کمی افزایش می‌یابد. شکل مقابل گویای مطلب است:

۳- فایل Indexed (فایل شاخص‌دار): درج رکوردهای داده‌ای به صورت طبقه‌بندی شده در طبقه مشخص شده از قبل (a در طبقه a، b در طبقه b و الی آخر)، که نتیجه می‌شود **ذخیره‌سازی سریع‌تر از فایل Sequential** چون به طور طبقه‌بندی شده (مشخص شده از قبل) درج می‌شود.

a	a2
	a1
b	b1
c	c2
	c1
d	d2
z	z3

از طرفی، **بازیابی** رکوردها **سریع‌تر از فایل Pile** و **کندتر** از **فایل ترتیبی** انجام می‌شود، چون با تعیین شاخص موردنظر (مثلا حرف ابتدایی کلید)، می‌توان مستقیما به طبقه مربوطه مراجعه کرد. اما با این حال، **رکوردهای درون هر طبقه به صورت مرتب‌نشده ذخیره می‌شوند**، بنابراین **جستجو در داخل هر طبقه همچنان به صورت ترتیبی (sequential)** انجام می‌شود.

شکل مقابل گویای مطلب است:

۴- فایل Index-Sequetial (فایل شاخص‌دار ترتیبی): درج رکوردهای داده‌ای به صورت طبقه‌بندی شده در طبقه مشخص شده از قبل (a در طبقه a، b در طبقه b و الی آخر)، که نتیجه می‌شود **ذخیره‌سازی سریع‌تر از فایل Sequential** چون به طور طبقه‌بندی شده (مشخص شده از قبل) درج می‌شود، از طرفی، **بازیابی** رکوردها **سریع‌تر از فایل Sequential (فایل ترتیبی)** انجام می‌شود، چون با تعیین شاخص موردنظر (مثلا حرف ابتدایی کلید)، می‌توان مستقیما به طبقه مربوطه مراجعه کرد. اما علاوه بر این، **رکوردهای درون هر طبقه به صورت مرتب‌شده ذخیره می‌شوند**، بنابراین **جستجو در داخل هر طبقه به صورت دودویی** انجام می‌شود. شکل مقابل گویای مطلب است:

a	a1
	a2
b	b1
c	c1
	c2
d	d2
z	z3

۵- درخت B^+ Tree، Hash، Bitmap Index : در فصل شاخص‌گذاری تشریح می‌شود.

توجه: محل ذخیره‌سازی داده‌های محیط عملیاتی **سیستم پرونده‌ای**، **فایل** و **سیستم پایگاه داده**، **جدول** است. ترکیبی از **ساختارهای داده‌ای** فوق برای **سازمان‌دهی داده‌ها** جهت ذخیره و بازیابی

اطلاعات (درج و جستجو) مورد استفاده قرار می‌گیرد.

تعریف اطلاعات (Information)

اگر داده‌ها داخل یک ساختار داده‌ای قرار بگیرد و سازماندهی شود، و سپس روی این داده‌ها **پرس‌وجو** بنویسیم، نتیجه این پرس‌وجو «**اطلاعات**» است. مانند دانشجویانی که ساکن شیراز هستند را در خروجی قرار بده. برای مثال علی و احمد ساکن شیراز هستند. بنابراین به نتیجه خروجی پرس‌وجو، **اطلاعات** می‌گوییم. اطلاعات ارزش افزوده‌ای نسبت به داده‌های خام دارد، زیرا به ما بینش و درک بهتری از پدیده‌ها می‌دهد.

محصول نرم‌افزاری

یک محصول نرم‌افزاری ماهیتی منطقی دارد و از دو بخش اساسی **داده** و **عملکرد** تشکیل شده‌است که در نهایت بر اساس **ورودی‌های مورد نظر مشتری**، باید **خروجی‌های مورد انتظار مشتری** را برآورده سازد. محصول نرم افزاری یک سیستم **مشتری‌محور** است.

ساخت محصول نرم‌افزاری

به منظور ساخت یک **محصول نرم افزاری** جهت ذخیره و بازیابی اطلاعات در یک محیط عملیاتی دو شیوه رایج وجود دارد:

۱- سیستم پرونده‌ای (File System)

۲- سیستم بانکی (Database)

ساخت محصول نرم‌افزاری پرونده‌ای (File System)

در گذشته علم مهندسی نرم‌افزار مثل امروز مطرح نبود. بنابراین در گذشته محصول نرم افزاری به صورت ضمنی بر اساس ذهن برنامه‌نویسان و بدون رعایت اصول مهندسی نرم افزار ساخته می‌شد.

به طور کلی هر **محصول نرم‌افزاری پرونده‌ای** از دو وجه اساسی زیر تشکیل شده است:

۱- ساختار داده (فایل داده): محل نگهداری داده‌های محیط عملیاتی به شکل فایل.

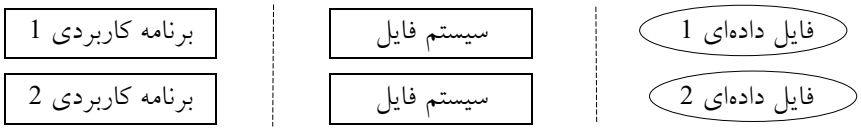
۲- عملکرد: دستورات یا کدهای قابل اجرا که باعث انجام وظایف مورد نظر می‌شود که به برنامه کاربردی (واسط کاربر) موسوم است.

توجه: در سیستم‌های پرونده‌ای تعاریف مربوط به ایجاد فایل داده‌ای درون خود برنامه کاربردی قرار می‌گیرد.

در گذشته سیستم‌های پرونده‌ای (File System) صرفا جهت ذخیره‌سازی و بازیابی اطلاعات مورد

استفاده قرار می‌گرفت. در این سیستم‌ها برای انجام هر عملی مثل **درج**، **حذف**، **جستجو** باید برنامه‌نویسی به زبان روالی مثل زبان پاسکال انجام می‌شد و هیچ اثری از زبان SQL جهت سهولت در بیان پرس‌وجوها و DBMS جهت مدیریت داده وجود نداشت، به بیان ساده‌تر هیچ خدمات مدیریتی از قبل آماده در این سیستم‌ها ارائه نمی‌شد. بنابراین به استفاده از ساختار داده‌ای و برنامه کاربردی بدون استفاده از DBMS **سیستم پرونده‌ای** گفته می‌شد.

در این روش با بهره‌گیری از امکانات سیستم فایل که در اختیار سیستم عامل است و یک زبان برنامه‌نویسی سطح بالا و با نگرش به ورودی‌ها و خروجی‌های موردنظر، برنامه‌های کاربردی به منظور کنترل و پردازش فایل‌های داده‌ای پیاده‌سازی می‌شود.



آدرس	موجودی	نام و نام‌خانوادگی	شمار حساب	شماره مشتری
تهران	100	علی علوی	1213	6037
تهران	200	علی علوی	1214	6037
شیراز	300	حسن حسنی	1520	6038

فایل حساب قرض الحسنه

توجه: برنامه کاربردی و **فایل حساب قرض الحسنه** در **دیسک (الف)** قرار دارد.

آدرس	موجودی	نام و نام‌خانوادگی	شمار حساب	شماره مشتری
تهران	400	علی علوی	2218	6037
شیراز	500	حسن حسنی	2219	6038

فایل حساب جاری

توجه: برنامه کاربردی و **فایل حساب جاری** در **دیسک (ب)** قرار دارد.

تعریف افزونگی (Redundancy)

افزونگی در معنای عام به معنی تکرار ذخیره‌سازی داده‌ها می‌باشد. به عبارت بهتر تکرار مقادیر یک یا چند صفت خاصه در نمونه‌های مختلف یک نوع رکورد از یک فایل، به بیانی دیگر ذخیره‌سازی آن مقادیر در بیش از یک نقطه از فایل. نوع دیگری از افزونگی، تکرار اطلاعات در فایل‌های مختلف می‌باشد.

توجه: افزونگی سبب هدر رفتن حافظه می‌گردد.

توجه: همانطور که مشاهده می‌شود، در **سیستم پرونده‌ای** به علت عدم **یکپارچگی یا یکپارچه‌سازی داده‌ها (Integration)**، پدیده افزونگی **تک فایلی** و **چند فایلی** را شاهد هستیم.

افزونگی طبیعی: به طور کلی بسیاری از داده‌های مورد نیاز یک برنامه کاربردی، همان داده‌های مورد نیاز برنامه‌های دیگر است، در روش پرونده‌ای هر برنامه کاربردی فایل داده‌ای ویژه خود را دارد، بنابراین داده‌های مشترک بین برنامه‌های مختلف باید مجدداً ذخیره شوند که سبب تکرار اطلاعات (افزونگی طبیعی چند فایلی) و موجب هدر رفتن رسانه ذخیره‌سازی می‌گردد. در سیستم‌های پرونده‌ای به علت عدم **یکپارچگی داده‌ها**، پدیده **افزونگی طبیعی** تک فایلی و چند فایلی را شاهد خواهیم بود.

افزونگی طبیعی تک فایلی: مانند تکرار نام و نام‌خانوادگی «علی علوی» برای شماره مشتری «6037» در فایل حساب قرض الحسنه. فرض کنید مشتری یک بانک مجبور باشد به ازای هربار افتتاح حساب کلیه اطلاعات شخصی خود را بازگو کند.

افزونگی طبیعی چند فایلی: مانند تکرار آدرس «تهران» برای شماره مشتری «6037» در فایل حساب قرض الحسنه و حساب جاری

ناسازگاری داده‌ای (آنومالی یا ناهنجاری یا بی‌نظمی)

علت ناسازگاری داده‌ای در **سیستم پرونده‌ای**، **افزونگی طبیعی** است. ناسازگاری زمانی بروز می‌کند که بنابر دلایلی یک فقره اطلاع در بیش از یک نقطه ذخیره گردد و لازم باشد بروزرسانی شود. اگر عمل بروزرسانی در تمام نقاطی که آن فقره اطلاع وجود دارد، انجام نشود، پدیده **ناسازگاری** رخ داده‌است. به عنوان مثال برای حالت تک فایلی، اگر در فایل داده‌ای سیستم حساب قرض الحسنه، نام و نام‌خانوادگی «علی علوی» در سطر اول تغییر کند و فراموش گردد تا در سطر دوم نیز بروزرسانی گردد، آنگاه شماره مشتری «6037» دارای دو نام و نام‌خانوادگی متفاوت می‌باشد، که ناسازگاری رخ داده‌است. به عنوان مثال برای حالت چند فایلی، زمانی که آدرس «علی علوی» در فایل داده‌ای سیستم حساب قرض الحسنه تغییر کند و آدرس «علی علوی» در فایل داده‌ای سیستم حساب جاری بروزرسانی نگردد، ناسازگاری رخ داده‌است.

سازگاری داده‌ای: در سیستم‌های فایلینگ کنترل سازگاری به دلیل عدم **یکپارچگی داده‌ها** و نبود DBMS بسیار مشکل و به صورت دستی است که خطای انسانی را به همراه خواهد داشت.

استقلال داده‌ای: در سیستم‌های فایلینگ هر برنامه کاربردی برای ایجاد و پردازش فایل داده‌ای مختص به خود نوشته شده‌است، بنابراین هرگونه تغییری در ساختار فایل داده‌ای منجر به تغییر در برنامه کاربردی می‌گردد. به بیان دیگر برنامه‌های کاربردی به جنبه و خصوصیات ساختار داده‌ها وابسته است. زیرا کلیه تعاریف ایجاد فایل‌های داده‌ای و کار با آن‌ها، درون برنامه‌های کاربردی

قرار دارد. که این موضوع به معنی عدم استقلال داده‌ای نیز می‌باشد. در این سیستم‌ها مولفه‌ای به صورت DBMS وجود ندارد که جداول داخل آن تعریف شود تا منجر به استقلال داده‌ای برنامه کاربردی از فایل داده شود. برنامه کاربردی به تغییرات در ساختار فایل و رسانه ذخیره‌سازی وابسته است. بنابراین در این سیستم‌ها برنامه کاربردی نسبت به تغییرات فایل داده‌ای **استقلال منطقی** ندارد و همچنین برنامه کاربردی نسبت به تغییرات رسانه ذخیره‌سازی **استقلال فیزیکی** هم ندارد، چون همه چیز هم فایل داده‌ای و هم عملکرد داخل برنامه کاربردی به صورت یکجا تعریف می‌شود. یک سیستم فایلینگ به صورت زیر است:

```
    type
        phone=record
        name:string [20];
        no:integer;
var
    fp:text;   یا   file of phone;
    i,n: integer;
begin
    assign (fp, 'sample.dat');
    rewrite (fp);
    readln (n);
for i:=1 to n do begin
    readln (no,name)
    writeln (fp,no,name)
end;
reset (fp);
    while TRUE do begin
    readln (fp,no,name);
    write (no,name)
if EOF (fp) then break;
    end;
    close (fp);
    end.
```

توجه: همانطور که در برنامه فوق ملاحظه می‌کنید، تعاریف مربوط به **عملکرد و فایل داده** با هم در یک برنامه کاربردی قرار دارد و همه چیز به صورت کدنویسی نوشته شده است.

سهولت پرس‌وجو نویسی: در سیستم‌های فایلینگ پرس‌وجونویسی توسط زبان‌های روالی مثل C انجام می‌شود، بنابراین سهولت در پرس‌وجو نویسی را ندارد.

امنیت منطقی: در سیستم‌های فایلینگ برقراری امنیت منطقی داده‌ها در برابر خطراتی از قبیل آتش‌سوزی و دستیابی غیرمجاز به دلیل عدم وجود یک مکانیزم پشتیبانی و حفاظتی و در نتیجه دسترسی ساده به داده‌ها بسیار مشکل است.

امنیت فیزیکی: در سیستم‌های فایلینگ به علت جدا بودن فایل داده‌ای برنامه‌ها از یکدیگر امنیت فیزیکی داده‌ها در سطح مناسبی برقرار است. زیرا داده‌های همه برنامه‌های کاربردی به دلیل عدم **یکپارچگی داده‌ها** در یک دیسک به یکباره در معرض **آسیب‌پذیری فیزیکی** قرار نمی‌گیرند. در این نوع سیستم، داده‌ها در مکان‌های مختلف با توجه به نوع برنامه‌های کاربردی ذخیره می‌شود. بنابراین هر برنامه کاربردی، داده‌های خاص خود را دارد، مثلاً مشخصات یک شخص مانند نام و نام‌خانوادگی و آدرس ممکن است هم در سیستم حساب قرض الحسنه باشد و هم در سیستم حساب جاری. حال اگر اطلاعات یک شخص در حساب قرض الحسنه از بین برود، در صورت وجود می‌توان اطلاعات شخص را از روی سیستم جاری استخراج نمود.

دسترسی همزمان و اشتراکی: با توجه به ماهیت سیستم‌های فایلینگ و محلی بودن برنامه‌ها و استفاده از الگوی صف جهت پردازش درخواست‌ها، دسترسی همزمان و اشتراکی به داده‌ها معنا و مفهومی ندارد.

زمان اجرای برنامه کاربردی: بستگی به شرایط مختلف دارد، البته در حالت کلی سیستم پرونده‌ای مکانیزم ویژه‌ای (مثل استفاده از شاخص و درخت جستجو) برای کاهش زمان اجرای برنامه کاربردی ندارد.

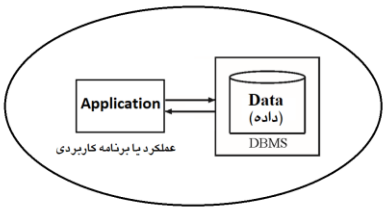
ساخت محصول نرم‌افزاری بانکی (Database)

امروزه علم مهندسی نرم‌افزار مطرح است و جایگاه ویژه‌ای در ساخت محصولات نرم افزاری به خود اختصاص داده است. بنابراین امروزه محصول نرم‌افزاری به صورت **صریح** بر اساس اصول مهندسی نرم‌افزار ساخته می‌شود. یک **محصول نرم‌افزاری** بر اساس ورودی‌های مورد نظر مشتری، باید خروجی‌های مورد انتظار مشتری را برآورده سازد. مشتری هیچ چیزی از نحوه ساخت محصول نرم‌افزاری نمی‌داند در نهایت می‌نشیند پشت سیستم و 2+2 می‌دهد و نتیجه 4 را می‌خواهد و اگر خروجی درست نباشد، کلا سیستم را نمی‌خواهد ولو اینکه با بهترین تکنولوژی ساخته شده باشد.

به طور کلی هر **محصول نرم‌افزاری بانکی** از دو بخش اساسی زیر تشکیل شده است:

۱- **ساختار داده (جدول)**: محل نگهداری داده‌های محیط عملیاتی به شکل جداول.

۲- **عملکرد**: دستورات یا کدهای قابل اجرا که باعث انجام وظایف مورد نظر می‌شود که به برنامه کاربردی موسوم است.



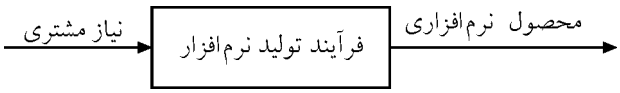
محصول نرم افزاری

مهندسی نرم‌افزار

علم مهندسی در همه ابعاد حرکتی؛ نتیجه موفق می‌دهد. وقتی فعالیتی شروع می‌شود، مسیر یا فرآیند درستی که منجر به نتیجه موفق بشود نیاز می‌شود. علم مهندسی نرم‌افزار **حرکت تولید نرم‌افزار** را توسط **فرآیند** (ارتباطات، تحلیل، طراحی و پیاده‌سازی)، **روش** (ساخت‌یافته یا شی‌گرا) و **ابزار** (ساخت‌یافته یا شی‌گرا) به سمت موفقیت سوق می‌دهد.

فرآیند تولید محصول نرم‌فزاری بانکی (فعالیت‌های چارچوبی)

فرآیند تولید محصول نرم‌افزاری همان نقشه حرکت است یعنی اول چه کاری بعد چه کاری بعد چه کاری، انقدر بعد چکار کنم تا اینکه بالاخره پروژه نهایی ایجاد شود. یعنی اول یه کارهایی انجام بدیم بعد یه کارها دیگه انجام بدیم. نمی‌شود یک شبه همه کارها را انجام بدهیم. باید فرآیندش طی بشود. هرچیزی نیاز داریم، مثل ماشین باید تولید بشود، چه بخواهیم ماشین تولید کنیم، چه بخواهیم غذا تولید کنیم، سوال بزرگ اینه که از کجا و چگونه؟ نقطه شروع کجاست؟ قدم اول کجاست؟ 0 تا 100 مسیر چگونه است؟ الگو و قالبی که چگونگی طی مراحل مختلف یک پروژه نرم‌افزاری را تعریف می‌نماید، اصطلاحاً **فرآیند تولید نرم‌افزار** نامیده می‌شود. شکل زیر فرآیند تولید نرم‌افزار و ورودی و خروجی آن را نشان می‌دهد:

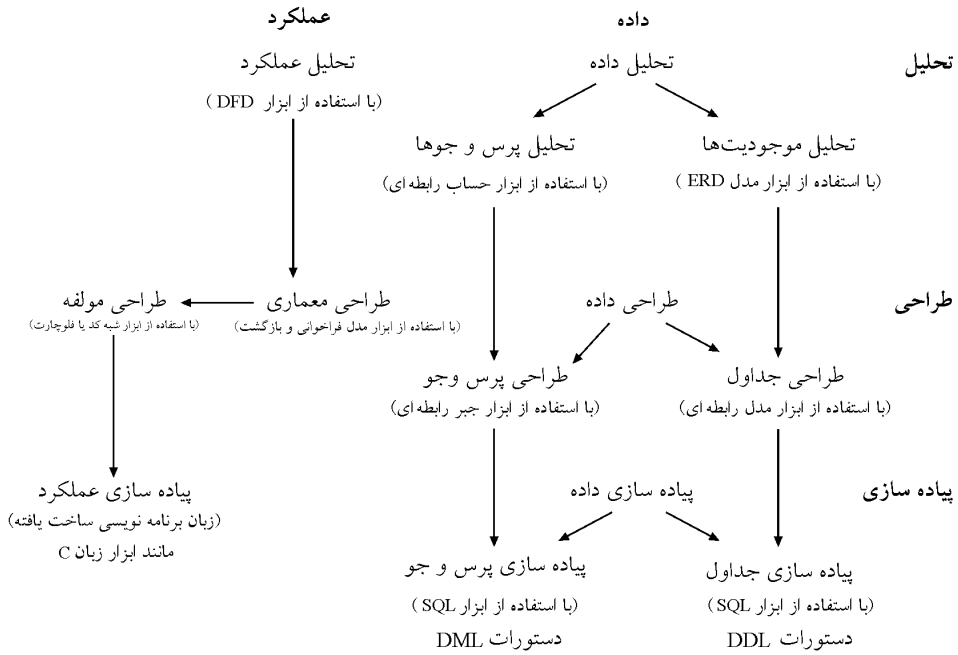


همانطور که ملاحظه می‌نمایید، ورودی این فرآیند، نیاز یا خواسته‌های مشتری و خروجی آن، یک **محصول نرم‌افزاری** است. یک فرآیند تولید در یک پروژه به ما می‌گوید که برای دستیابی به هدف مطلوب که همان تولید یک فرآورده‌ی نرم‌افزاری با کیفیت مطلوب است، چه کسی (Who)، چه کاری را (What)، چه موقع (When) و چگونه (How) باید انجام دهد، در واقع، بدون داشتن تعریف مشترکی از فرآیند تولید نرم‌افزار، هماهنگی انجام کار تیمی در یک پروژه‌ی نرم‌افزاری، امکان‌پذیر نخواهد بود.

یک **محصول نرم‌افزاری بانکی درست** به واسطه‌ی **فرآیند تولید نرم‌افزار** (نقشه حرکت یا اول

چه کاری بعد چه کاری) که شامل فعایت‌های **ارتباطات درست**، **تحلیل درست**، **طراحی درست** و **پیاده‌سازی درست** است، ایجاد می‌گردد.

مراحل ایجاد یک **محصول نرم‌افزاری** متشکل از بخش **داده** و **عملکرد** مطابق روال زیر می‌باشد:



۱- ارتباطات (زبان انسان): در فعالیت ارتباطات، **لیست نیازمندی‌های مشتری** بر اساس گفته‌ها و درخواست‌های مشتری توسط سازنده تهیه و تدوین می‌شود.

۲- تحلیل (به زبان شبیه انسان چک‌نویس اولیه (ترجمه اولیه))

۳- طراحی (به زبان شبیه ماشین چک‌نویس ثانویه (ترجمه ثانویه))

۴- پیاده‌سازی (زبان ماشین و پاک‌نویس)

توجه: تحلیل و طراحی نباشد، **ترجمه** از زبان انسان به زبان ماشین انجام نمی‌شود.

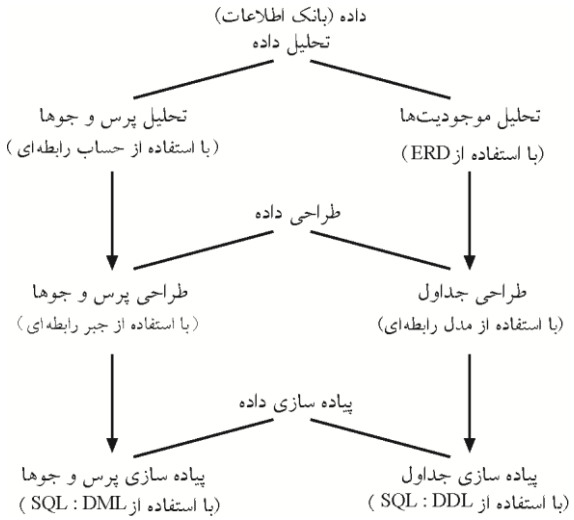
توجه: هر مرحله الگوی مرحله بعدی است. هر مرحله دستاورد (Artifact) دارد، خروجی هر مرحله ورودی مرحله بعد است.

توجه: در پایان مراحل ارتباطات، تحلیل، طراحی و پیاده‌سازی به یک **محصول نرم‌افزاری** درست که نیازهای مشتری را برآورده می‌سازد می‌رسیم.

توجه: از آنجا که ما قصد داریم در این کتاب مفاهیم مربوط به پایگاه داده‌ها را تشریح نماییم،

بنابراین از بیان مطلب مربوط به بخش عملکرد نرم‌افزار صرف‌نظر می‌نمائیم. بخش عملکـرد را در کتاب مهندسی نرم‌افزار مورد بحث و بررسی قرار داده‌ایم. پس در ادامه به طور مفصـل بـه مفـاهیم مربوط به پایگاه داده می‌پردازیم.

توجه: در سیستم‌های بانکی تعاریف مربوط به ایجاد جداول داده‌ای درون DBMS قـرار می‌گیـرد، در این روش به منظور **ایجاد جداول** روال زیر را خواهیم داشت:



۱- تحلیل
(طراحی ادراکی (مفهومی) یا ادراکی عام)
توسط DA

۲- طراحی
(طراحی منطقی یا ادراکی خاص)
توسط DBA

۳- پیاده‌سازی
توسط DBP

در مرحله بعد یک برنامه کاربردی (عملکرد) مطابق فرآیند تولیـد نرم‌افـزار بایـد ایجـاد گـردد. تـا کاربران نهایی بتوانند به کمک DBMS در نهایت به داده‌های ویژه خود با توجه به سطوح دسترسی تعریف شده در DBMS برای آنها دسترسی یابند.

انتزاع یا تجرید (abstraction)

مدل تحلیل، مدل‌سازی عالم خارج به زبانی شبیه انسان است، اما مـدل طراحـی مدل‌سـازی عـالم داخل ماشین به زبانی شبیه زبان ماشین است. بنابراین برای اینکه ساخت برنامه کامپیوتری کـه بـه زبان ماشین است، امکان‌پذیر باشد، باید مدل تحلیل به مدل طراحی نگاشت شود تا مـدل طراحـی بتواند به عنوان نقشه راهی شبیه به زبان ماشین، راهگشا باشد.

مدل‌های تحلیل، کلی‌تر و انتزاعی‌تر هستند، و برای مدل‌سـازی عـالم خـارج مـورد اسـتفاده قـرار می‌گیرند، بدون آنکه به جزئیات نحوه پیاده‌سازی بپردازند، در واقع، مدل تحلیل بـه کلی‌گـویی بـه زبان انسـان و حـذف جزئیـات نحـوه پیاده‌سـازی می‌پـردازد. امـا مـدل‌های طراحـی، جزئی‌تـر و غیرانتزاعی‌تر هستند، و برای مدل‌سازی عالم داخل ماشین مورد استفاده قرار می‌گیرنـد، و بـه بیـان جزئیات نحوه پیاده‌سازی می‌پردازند، در واقع مدل طراحی به جزئی‌گویی به زبـان شـبیه ماشـین و

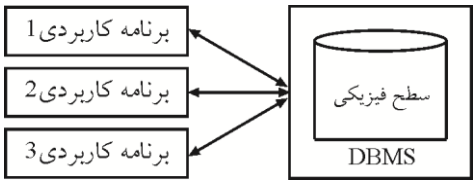
درج جزئیات نحوه پیاده‌سازی می‌پردازد.
با نگاهی سطح به سطح، به حل مساله، سطوح مختلفی از انتزاع را خواهیم داشت. در بالاترین سطح انتزاع، راه حل به صورت کلی به زبان محیط مساله بیان می‌گردد. در سطوح پایین‌تر، راه حل به جزئیات پیاده‌سازی نزدیک‌تر می‌شود و در پایین‌ترین سطح انتزاع راه حل به صورتی بیان می‌شود تا مستقیما قابل پیاده‌سازی باشد.
رعایت تجرید در فرآیند تولید یک نرم‌افزار، باعث کاهش شلوغی و پیچیدگی پروژه می‌گردد، زیرا از ذکر اطلاعات اضافی خودداری شده و افراد تیم توسعه مانند تحلیل‌گر تنها با اطلاعات ضروری سروکار خواهد داشت.
مثال: مکالمه محاوره‌ای زیر را در نظر بگیرید:
سوال: کجا هستی؟
پاسخ: بیرون
سوال: کی می‌یای؟
پاسخ: میام
همانطورکه مشاهده می‌کنید مکالمه فوق کاملا انتزاعی و با حذف جزئیات از طرف پاسخ‌دهنده انجام شده است. و برای سوال‌کننده همچنان سوالاتی نظیر کجای بیرون دقیقاً و چه زمانی دقیقاً، باقی‌مانده است، چون پاسخ‌ها کاملاً انتزاعی از طرف پاسخ‌دهنده بیان شده است. انسان موجودی انتزاعی و ماشین موجودی غیرانتزاعی است.

توجه: در مورد هر موضوعی هرچه‌قدر جزئیات **بیشتری** بدانیم، درجه انتزاع ذهن ما نسبت به آن موضوع **کمتر** است و بالعکس در مورد هر موضوعی هرچه‌قدر جزئیات **کمتری** بدانیم درجه انتزاع ذهن ما نسبت به آن موضوع **بیشتر** است.

مدل‌سازی

یک مدل، ساده شده یک واقعیت است. ایجاد یک مدل برای سیستم‌های نرم‌افزاری قبل از ساخت یا بازساخت آن، به اندازه داشتن نقشه برای ساختن یک ساختمان ضروری وحیاتی است. علت اصلی مدل کردن سیستم‌های پیچیده این است که نمی‌توان به یکباره کل سیستم را تجسم کرد و ممکن است سیستم دارای ابهامات بسیاری باشد. لذا برای رفع این ابهامات و نیز برای فهم کامل سیستم و یافتن و نمایش ارتباط بین قسمت‌های مختلف آن، از مدل‌سازی استفاده می‌شود. فعالیت مدل‌سازی خود شامل دو مرحله‌ی مدل تحلیل و مدل طراحی می‌باشد. مدل تحلیل پس از فعالیت ارتباطات (جمع‌آوری نیازمندی‌ها) و قبل از مدل طراحی انجام می‌شود، در واقع خروجی مدل تحلیل، ورودی مدل طراحی می‌باشد.

توجه: DBMS سرواژه عبارت **D**ata**B**ase **M**anagement **S**ystem به معنی **سیستم مدیریت پایگاه** داده یا **سمپاد** است. در واقع DBMS رابط بین برنامه‌های کاربردی و داده‌هاست و هرگونه دستیابی به داده‌ها از طریق DBMS صورت می‌گیرد. DBMS همچون قلعه‌ای می‌ماند که محل زندگی پادشاه، ملکه و شاه‌زادگان را درون خود محصور کرده‌است، که هرگونه آمد و شد به محل زندگی پادشاه باید با عبور از قلعه و اجازه قلعه‌بانان صورت گیرد. پادشاه، ملکه و شاهزادگان همان داده‌ها هستند و قلعه و قلعه‌بانان همان DBMS.



تعریف پایگاه داده (Database): امروزه به ترکیب ساختار داده‌ای (داده‌های سازمان‌دهی شده و مرتبط به هم و ذخیره شده دائمی در دیسک) و DBMS، **پایگاه داده** گفته می‌شود که انواع آن به صورت زیر است:

- پایگاه داده رابطه‌ای (Relational DBMS)
- پایگاه داده شی‌گرا (Object Oriented DBMS)
- پایگاه داده غیررابطه‌ای (NoSQL DBMS)

توجه: هنگامی‌که ساختار داده‌ای (داده‌های سازمان‌دهی شده) و DBMS در کنار هم قرار گیرد مفهوم **پایگاه داده** خلق می‌شود. دقت کنید ساختار داده‌ای بدون DBMS پایگاه داده نیست و صرفا همان سیستم پرونده‌ای (File System) است.

توجه: با استفاده از پایگاه داده، می‌توان داده‌های خام را به صورت سازمان‌دهی‌شده ذخیره کرد و سپس این داده‌ها را پردازش کرد تا **اطلاعات** تولید شود. پایگاه داده نقش کارخانه‌ای را ایفا می‌کند که داده‌ها را به عنوان **مواد اولیه** به **محصول نهایی (اطلاعات)** تبدیل می‌کند.

معماری بانک اطلاعات

معماری به معنی نحوه ارتباط بخش‌های مختلف یک سازه است. معماری ANSI برای پایگاه داده‌ها شامل سه لایه زیر است:

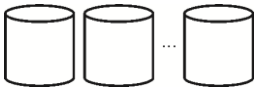
۱- لایه خارجی (External Layer یا View Layer)

۲- لایه ادراکی یا مفهومی (Conceptual Layer) شامل زیر لایه‌های مدل تحلیل (طراحی ادراکی Conceptual Design یا لایه ادراکی عام) و مدل طراحی (طراحی منطقی Logical Design یا لایه ادراکی خاص).

۳- لایه داخلی یا فیزیکی (Internal Layer یا Physical Layer)

توجه: در معماری بانک اطلاعات، کلمه تصویر، مترادف لایه یا دید است. برای مثال لایه خارجی همان مفهوم تصویر یا دید خارجی را دارد.

توجه: یک محصول نرم‌افزاری به واسطه فرآیند تولید نرم‌افزار که شامل فعالیت‌های تحلیل، طراحی، پیاده‌سازی و تست می‌باشد، ایجاد می‌گردد. کاربران نهایی در لایه خارجی، تحلیل و طراحی در لایه ادراکی و پیاده‌سازی در لایه داخلی قرار دارد. شکل زیر گویای مطلب است:

	لایه خارجی	کاربر n ... کاربر 2 کاربر 1	دیدهای کاربران (view)
تحلیل	لایه ادراکی عام (طراحی ادراکی)	مدل ER موجودیت 1 — ارتباط — موجودیت 2	کل پایگاه داده بدون توجه به مدل خاصی
طراحی	لایه ادراکی خاص (طراحی منطقی)	مدل رابطه‌ای A B C	کل پایگاه داده در قالب مدل انتخابی
پیاده‌سازی	لایه داخلی (فیزیکی)	SQL 	کل پایگاه داده روی رسانه

لایه خارجی

لایه خارجی بالاترین سطح انتزاع را دارد. موضوع این لایه **مدیریت کاربران پایگاه داده** است. در این لایه، میزان دسترسی کاربران به پایگاه داده، مشخص و مدیریت می‌گردد. لایه خارجی، تنها لایه‌ای است که به کاربران مربوط می‌شود. هر کاربر با بخشی از بانک سروکار دارد و فقط اجازه دستیابی به بخشی از بانک به او داده می‌شود. برای مثال در بانک اطلاعات مربوط به بانک مرکزی، کاربر نهایی (کارمندان) بخش ارز مسافری، حق دستیابی به بخش‌های دیگر بانک مثل حساب افراد و وام ها را ندارند. قانون «**پنهان‌سازی اطلاعات**» که می‌گوید «**به هر کس به همان اندازه اطلاعات بده که نیاز دارد و نه بیشتر**» در اینجا صادق است. بنابراین در لایه خارجی دیدهای مختلف کاربران مطرح است و هر کاربر فقط بخشی از بانک را می‌بیند. در واقع به ازای هر کاربر

یک حوزه دید (View) تعریف می‌شود. حوزه دید یک کاربر در پایگاه داده عبارتند از بخشی از پایگاه داده که مطابق حق دسترسی، آن کاربر فقط اجازه کار با آن بخش را دارد. تعریف حوزه دید برای هر کاربر باعث محدودسازی سطح دسترسی کاربران و به تبع تامین امنیت می‌شود. بنابراین تعریف دید (View) دو مزیت دارد اول اینکه امنیت منطقی ایجاد می‌کند تا کاربر همه بانک را نبیند، و دوم اینکه سهولت برای کاربر ایجاد می‌کند، چون همه بانک را نمی‌بیند و راحت‌تر کار می‌کند. ساختار View توسط دستورات DDL در SQl ایجاد می‌شود که در فصل مربوطه بررسی می‌شود.

توجه: لایه خارجی لایه‌ای است که کاربران با آن سر و کار دارند، لایه‌های دیگر به مدیر و برنامه‌سازان بانک اطلاعات مربوط می‌شود.

لایه ادراکی

لایه ادراکی سطح میانی انتزاع را دارد. موضوع این لایه **مدل‌سازی داده‌ها** در پایگاه داده است. لایه ادراکی شامل دو زیر لایه **مدل تحلیل**(طراحی ادراکی یا لایه ادراکی عام) و **مدل طراحی**(طراحی منطقی یا لایه ادراکی خاص) می‌باشد.

توجه مهم: داده‌ها در پایگاه داده رابطه‌ای به شکل رابطه‌ای سازماندهی می‌شود.

مدل تحلیل (طراحی ادراکی یا لایه ادراکی عام)

مدل تحلیل، سطح انتزاع نزدیک به لایه خارجی را دارد. موضوع این لایه **مدل تحلیل داده** است.

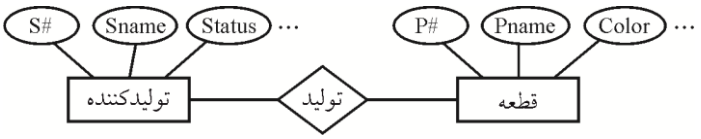
مثال: بانک اطلاعات و پرس‌وجوهای مربوط به نگهداری اطلاعات سیستم تولیدکنندگان و قطعات را **تحلیل**، **طراحی** و **پیاده‌سازی** نمایید.

مدل تحلیل

مدل تحلیل داده بر دو بخش تحلیل موجودیت‌ها و تحلیل پرس‌وجوها می‌باشد. تحلیل موجودیت‌ها توسط ابزار مدل ER و تحلیل پرس‌وجوها توسط ابزار حساب رابطه‌ای مدل می‌شود.

الف) تحلیل موجودیت‌ها (توسط مدل ER)

در یک محیط عملیاتی اسامی عام، موجودیت‌ها می‌باشند. بنابراین موجودیت‌های تهیه‌کننده (S) و قطعه (P) موجود می‌باشند.



توجه: مبحث مدل ER در فصل مدل ER تشریح می‌گردد.

ب) تحلیل پرس و جوها (توسط حساب رابطه‌ای)

پرس‌وجو: نام تولیدکنندگان ساکن شهر C2 :

$$\{t \mid \exists\ sx \in s(t[sname] = sx[sname] \ \text{ and } sx[city] = 'c2')\} \Rightarrow$$

خروجی
Sname
Sn2
Sn3

توجه: حساب رابطه‌ای به دو طبقه کلی حساب رابطه‌ای دامنه‌ای و حساب رابطه‌ای تاپلی تقسیم می‌شود. پرس و جوی فوق بر اساس حساب رابطه‌ای تاپلی نوشته شده است.

توجه: مبحث حساب رابطه‌ای (دامنه‌ای و تاپلی) در فصل حساب رابطه‌ای تشریح می‌گردد.

مدل طراحی (طراحی منطقی یا لایه ادراکی خاص)

مدل طراحی، سطح انتزاع نزدیک به لایه داخلی را دارد. موضوع این لایه **مدل طراحی داده** است.

توجه: مدل تحلیل (طراحی ادراکی یا ادراکی عام) به مدل خاصی وابستگی ندارد، چون در شروع فعالیت‌ها قرار دارد، اما مدل طراحی (طراحی منطقی یا ادراکی خاص) به مدل خاصی بستگی دارد، بسته به اینکه در فعالیت تحلیل چه مدلی انتخاب شود، مدل طراحی باید تبعیت کند. اگر مدل تحلیل ساخت یافته (مدل ER) بود، مدل طراحی و فعالیت پیاده‌سازی نیز باید رویکرد ساخت‌یافتگی را دنبال کند و اگر مدل تحلیل شیءگرا (مدل UML) بود، مدل طراحی و فعالیت پیاده‌سازی نیز باید رویکرد شیءگرایی را دنبال کند.

مدل طراحی

پس از مدل تحلیل داده، نوبت به مدل طراحی داده می‌رسد. مدل طراحی داده بر دو بخش طراحی جداول و طراحی پرس وجو می‌باشد. طراحی جداول از بخش طراحی داده، تحلیل موجودیت (ERD) از مدل تحلیل را به عنوان ورودی دریافت کرده و توسط مدل رابطه‌ای، طراحی جداول را انجام می‌دهد. طراحی پرس‌وجو از بخش طراحی داده، تحلیل پرس‌وجو (حساب رابطه‌ای) از مدل تحلیل را به عنوان ورودی دریافت کرده و توسط جبر رابطه‌ای، طراحی پرس‌وجو را انجام می‌دهد.

الف) طراحی جدول (توسط مدل رابطه‌ای)

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

جدول S
(تولیدکنندگان)

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

جدول SP
(تولید)

P#	Pname	Color
P1	Pn1	Red
P2	Pn2	Blue

جدول P
(قطعات)

توجه: مبحث مدل رابطه‌ای در فصل مدل رابطه‌ای تشریح می‌گردد.

ب) طراحی پرس‌وجو (توسط جبر رابطه‌ای)

پرس‌وجو: نام تولیدکنندگان ساکن شهر C2 :

$$\Pi_{sname}(\sigma_{city=c2}(s)) \Rightarrow$$

خروجی

Sname
Sn2
Sn3

توجه: مبحث جبررابطه‌ای در فصل جبررابطه‌ای تشریح می‌گردد.

لایه داخلی(فیزیکی)

لایه داخلی پایین‌ترین سطح انتزاع را دارد. موضوع این لایه **چگونگی ذخیره‌سازی فیزیکی اطلاعات پایگاه داده در دیسک** است که توسط DBMS انجام می‌شود.

زبان‌های پیاده‌سازی

همانطور که گفتیم یک محصول نرم‌افزاری از دو وجه عملکرد (برنامه کاربردی) و داده (بانک اطلاعات) تشکیل می‌شود. در ادامه به معرفی انواع زبان‌های برنامه‌سازی می‌پردازیم.

زبان پیاده‌سازی برنامه کاربردی (وجه عملکرد)

برنامه کاربردی نیز مانند بخش داده، حاصل مراحل تحلیل، طراحی و پیاده‌سازی می‌باشد که شرح این مراحل مربوط به درس مهندسی‌نرم‌افزار می‌باشد. مرحله پیاده‌سازی برنامه کاربردی توسط یکی از زبان‌های برنامه‌نویسی سطح بالا انجام می‌شود.

توجه: به زبان‌های سطح بالا، **زبان میزبان** (Host language) یا زبان روالی (Procedural) نیز گفته می‌شود.

زبان پیاده‌سازی بانک اطلاعات (وجه داده)

در بانک اطلاعات از زبان‌های بیانی (Declarative) که به آنها زبان پرس‌وجو (Query Language) نیز گفته می‌شود، استفاده می‌شود. در زبان‌های بیانی کاربر برنامه‌ساز کافیست بگوید چه چیزی لازم دارد تا سیستم برای او ایجاد (مثل ساخت Table یا Index یا View) یا استخراج (مثل پرس‌وجوها) کند. در واقع چگونگی ایجاد جداول یا استخراج پرس‌وجوها از دید کاربر برنامه‌ساز وکاربر نهایی مخفی است.

برای مثال در یک سیستم کامپیوتری بخش آموزش یک دانشگاه برای استخراج «نام و شماره دانشجویانی که معدل آنها بالای 18 است» کافی است در ابتدا نام جدول یا جداول به عنوان مکان یا محل پرس‌وجو، سپس شرط انتخاب سطر به عنوان ملاک انتخاب سطرها و در نهایت ستون‌های نام و شماره دانشجو به عنوان ملاک انتخاب ستون‌های مورد نیاز بیان شود. و به همین دلیل است

که کلمه «بیانی» به این نوع زبان‌ها اطلاق می‌شود، زیرا کاربر برنامه‌ساز فقط مشخصات کامل وکلی آنچه را که لازم دارد بیان می‌کند و به جزئیات چگونگی و نحوه استخراج کاری ندارد، این مشخصات کامل و کلی به ترتیب شامل سه بخش نام جدول یا جداول به عنوان مکان یا محل پرس‌وجو، شرط انتخاب سطر به عنوان ملاک انتخاب سطرها و مدل انتخاب ستون به عنوان ملاک انتخاب ستون‌های مورد نیاز است. در مثال فوق لازم نیست از تعداد سطرهای جدول، حلقه تکرار و غیره حرفی به میان آید. اصولاً حلقه تکرار با زبان‌های بیانی بیگانه است.

توجه: به زبان‌های بیانی، زبان‌های میهمان (Client language) نیز گفته می‌شود.

توجه: یک برنامه کاربردی نوشته شده به یک زبان روالی سطح بالا پس از ارتباط با یک سرویس‌دهنده زبان بیانی مانند SQL Server از طریق مکانیزم‌های ویژه (Connection String) اقدام به استفاده از بانک اطلاعات می‌نماید. همچنین برای اتصال به پایگاه داده، در رشته اتصال (Connection String) نحوه احراز هویت (Authentication) کاربر مشخص می‌شود.

توجه: هر مدل داده‌ای (مدل رابطه‌ای)، زبان بیانی خاص خود را دارد. به طور مثال SQL برای مدل رابطه‌ای ایجاد گردیده است.

در مرحله پیاده‌سازی بانک اطلاعات سه مقوله ایجاد جداول، ایجاد پرس‌وجوها و ایجاد سطوح امنیتی کاربران نهایی انجام می‌گردد. بنابراین دستورات SQL به سه دسته زیر تقسیم می‌گردد:

۱ـ دستورات تعریف داده‌ها (DDL: Data Definition Language)

کارهای مربوط به ظرف‌سازی و ساختارسازی، مثل ایجاد و حذف جداول، دیدها و شاخص.

مثال: دستور Create Table برای ایجاد جداول و دستور Drop Table برای حذف جداول

۲ـ دستورات تغییر داده‌ها (DML: Data Manipulation Language)

کارهای مربوط به تغییرات محتوای جداول، مثل ذخیره و بروزرسانی داده‌ها در جداول و بازیابی و حذف داده‌ها از جداول

مثال : دستورات Insert، Select، Update و Delete

توجه: به طور کلی دستورات DML به دو نوع (1) Procedural DMLs (Low Level DML) و (2) Non-Procedural DMLs (High Level DML) تقسیم می‌شود. دستورات Procedural DMLs در سیستم‌های فایلینگ مورد استفاده قرار می‌گیرد. برای مثال در سیستم‌های فایلینگ علاوه بر ساخت بخش عملکرد توسط زبان پاسکال، بخش داده‌ای و روند درج، حذف و آپدیت نیز توسط همان دستورات پاسکال و استفاده از دستوراتی مثل حلقه و شرط و با جزئیات ذخیره و بازیابی نوشته می‌شود. اما دستورات Non-Procedural DMLs در سیستم‌های بانکی (پایگاه داده) مورد استفاده قرار می‌گیرد. برای مثال در سیستم‌های بانکی دستوراتی همچون Insert، Select، Update و Delete در SQL جهت ذخیره و بازیابی اطلاعات مورد استفاده قرار می‌گیرد.

DML چه زبان سطح پایین دستکاری داده‌ها (Low Level DML) باشد و چه زبان سطح بالای دستکاری داده‌ها (High Level DML) باید در یک زبان برنامه همه منظوره (General purpose Language) یا زبان‌های روالی متعارف یا سطح بالا نهفته شود. به زبان سطح بالای دستکاری داده‌ها (High Level DML) زبان بیانی (Declarative) نیز گفته می‌شود.

۳ـ دستورات کنترل داده‌ها (Data Control Language : DCL)

کارهای مربوط به امنیت جداول، مثل ایجاد سطوح دسترسی برای کاربران نهایی

مثال: دستور Grant برای ایجاد حق دسترسی و دستور Revoke برای حذف حق دسترسی

۴ـ دستورات کنترل تراکنش‌ها (TCL: Transaction Control Language)

کارهای مربوط به کنترل تراکنش‌ها، مثل تثبیت و لغو تراکنش‌ها. این دستورات در واقع تغییرات اعمال شده توسط دستورات DML را اداره می کند. SQL هیچ یک از دستورات DML را ماندگار نمی‌کند، مگر اینکه به طور کامل اجرا و Commit شود.

مثال: دستور Commit برای تثبیت تراکنش و دستور Rollback برای لغو تراکنش

توجه: اجتماع مجموعه دستورات DDL، DML و DCL را DSL می‌گویند.

توجه: DSL سرواژه عبارت Data Sub Language است.

توجه: به اجتماع مجموعه دستورات DDL، DML و DCL، زبان پرس‌وجو (QL: Query Language) نیز گفته می‌شود.

به طور کلی دو دسته زبان داده‌ای وجود دارد :

الف) زبان‌های داده‌ای نامستقل یا ادغام شده

در این نوع زبان‌ها، زبان داده‌ای حتماً باید میهمان یک زبان سطح بالا باشد مثل SQL که در ویژوال بیسیک استفاده می‌شود. خود زبان ادغام شده (embedded) به دو شکل صریح و ضمنی طبقه‌بندی می‌شود. در حالت ادغام صریح عین دستور زبان داده‌ای فرعی در برنامه میزبان نوشته می‌شود که موسوم به LINQ است. برای مثال دستورات SQL مابین دستورات #C استفاده می‌شود. اما در حالت ادغام ضمنی دستورات زبان داده‌ای در قالب توابع فراخوانی (stored procedure) می‌شوند.

ب) زبان‌های داده‌ای مستقل

در این نوع زبان‌ها، زبان داده‌ای نیازی به زبان سطح بالا ندارد. مانند Access و Foxpro.

توجه مهم: داده‌ها در پایگاه داده رابطه‌ای به شکل **مدل رابطه‌ای** (Relational Model) سازماندهی می‌شود. در ریاضیات به هر زیرمجموعه دلخواه از حاصلضرب دکارتی دنباله‌ای از دامنه‌ها یک

رابطه گفته می‌شود. بهترین راه نمایش و پیاده‌سازی رابطه استفاده از جدول است، پس جدول یک شیوه نمایش تاپل‌های رابطه است. به بیان دیگر رابطه مفهومی ریاضی دارد، اما از نظر کاربردی نمایشی جدولی دارد، بنابراین از دیدگاه کاربردی رابطه یک جدول است. نتیجه اینکه در این مدل، داده‌ها در داخل جداول مطابق خواص مجموعه‌ها در ریاضی سازماندهی می‌شود. از آنجاکه این مدل پایه ریاضی دارد دو ویژگی پرفایده دارد، اول اینکه کامل است چون برای هر درخواستی همه پاسخ‌های درست را دارد. و دوم اینکه درست است چون هر پاسخی که تولید می‌کند قطعا درست است. در مبحث رابطه در ریاضی نوع داده چندرسانه‌ای مثل تصویر، صوت و ویدیو نداریم، پس امکان ذخیره‌سازی این اطلاعات در جدول هم وجود ندارد.

پیاده‌سازی

پس از مدل طراحی نوبت به پیاده‌سازی می‌رسد. پیاده‌سازی داده بر دو بخش پیاده‌سازی جداول و پیاده‌سازی پرس‌وجو می‌باشد. پیاده‌سازی جداول از بخش پیاده‌سازی داده، طراحی جداول از مدل طراحی را به عنوان ورودی دریافت کرده و توسط دستورات DDL در SQL، پیاده‌سازی جداول را انجام می‌دهد. پیاده‌سازی پرس‌وجو از بخش پیاده‌سازی داده، طراحی پرس‌وجو از مدل طراحی را به عنوان ورودی دریافت کرده و توسط دستورات DML در SQL پیاده‌سازی پرس‌وجو را انجام می‌دهد.

الف) پیاده‌سازی جدول (توسط SQL:DDL)

پیاده‌سازی جدول تولیدکنندگان (S)

```
Create Table S
        (
        S# char (5),
        Sname char (20),
        City char (15),
        Primary key (S#)
        )
```

پیاده‌سازی جدول قطعات (P)

```
Create Table P
        (
        P# char (5),
        Pname char (20),
        Color char (10),
        Primary key (P#)
        )
```

پیاده‌سازی جدول تولید (SP)

```
Create Table SP
        (
        S# char (5),
        P# char (5),
        QTY numeric (10),
        Primary key (S#, P#),
        Foreign key (S#) References S(S#)
          on delete cascade
          on update cascade,
        Foreign key (P#) References P(P#)
          on delete cascade
          on update cascade,
        Check (QTY>1 AND QTY<1000)
        )
```

توجه: مبحث SQL:DDL در فصل SQL:DDL تشریح می‌گردد.

ب) پیاده‌سازی پرس و جو (توسط SQL : DML)

پرس‌وجو: نام تولید کنندگان ساکن شهر C2 :

```
Select  Sname                خروجی      Sname
From S                                   Sn2
Where  City = 'C2'           ⇒           Sn3
```

توجه: مبحث SQL:DML در فصل SQL:DML تشریح می‌گردد.

مولفه‌های سیستم پایگاه داده

یک سیستم پایگاه داده از ۴ مولفه اساسی تشکیل شده است که هر یک مسئولیتی را بر عهده گرفته است، این مولفه‌ها شامل (الف) **مولفه ساختار داده‌ای** (ب) **مولفه نرم‌افزار**، (ج) **مولفه کابران** و (د) **مولفه سخت‌افزار** است که در ادامه به تفکیک شرح خواهیم داد:

الف) مولفه ساختار داده‌ای

تعریف یکپارچگی یا یکپارچه‌سازی (Integration): یکپارچگی خاصیت مولفه ساختار داده‌ای سیستم بانکی است که نتیجه آن **کاهش افزونگی طبیعی** است. یکپارچگی (همگرایی) سیستم پایگاه داده در مقابل چندپارگی (واگرایی) سیستم پرونده‌ای، باعث حذف پدیده جزایر پراکنده می‌شود. داده‌ها بجای تکرار در چند محل مختلف، در یک محل نگهداری می‌شود و تکرار داده‌ها

و به تبع افزونگی طبیعی کاهش می‌یابد.

مثال: پیاده‌سازی سیستم حساب قرض الحسنه و جاری در دو حالت **سیستم پرونده‌ای** و **بانکی**

سیستم پرونده‌ای (قبل از یکپارچگی داده‌ها)

آدرس	موجودی	نام و نام‌خانوادگی	شمار حساب	شماره مشتری
تهران	100	علی علوی	1213	6037
تهران	200	علی علوی	1214	6037
شیراز	300	حسن حسنی	1520	6038

فایل حساب قرض الحسنه

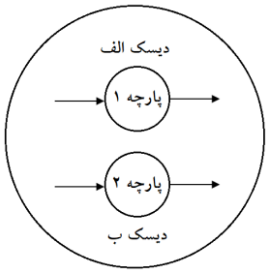
توجه: برنامه کاربردی و **فایل حساب قرض الحسنه** در **دیسک (الف)** قرار دارد.

آدرس	موجودی	نام و نام‌خانوادگی	شمار حساب	شماره مشتری
تهران	400	علی علوی	2218	6037
شیراز	500	حسن حسنی	2219	6038

فایل حساب جاری

توجه: برنامه کاربردی و **فایل حساب جاری** در **دیسک (ب)** قرار دارد.

شکل زیر گویای مطلب است:



توجه: همانطور که مشاهده می‌شود، در **سیستم پرونده‌ای** به علت عدم **یکپارچگی داده‌ها (Integration)**، پدیده **افزونگی طبیعی** تک فایلی و چند فایلی را شاهد هستیم.

سیستم بانکی (پس از یکپارچگی داده‌ها)

آدرس	نام و نام‌خانوادگی	شماره مشتری
تهران	علی علوی	6037
شیراز	حسن حسنی	6038

جدول مشتری

موجودی	شمار حساب	شماره مشتری
400	2218	6037
500	2219	6038

جدول حساب قرض الحسنه

موجودی	شماره حساب	شماره مشتری
100	1213	6037
200	1214	6037
300	1520	6038

جدول حساب قرض الحسنه

توجه: جدول حساب قرض الحسنه و **جاری** در **یک دیسک** قرار دارد.

انواع افزونگی در پایگاه داده

افزونگی بر دو نوع است:

۱- افزونگی طبیعی (محتوایی)

در افزونگی طبیعی، یک مقدار مشخص از صفت خاصه در تعدادی از نمونه رکوردهای یک جدول وجود دارد.

توجه: افزونگی طبیعی حالت‌های مختلفی دارد که به طور سلسله مراتبی در سطوح مختلف آنرا کاهش می‌دهیم. این مورد در فصل نرمال‌سازی به طور مفصل بررسی می‌شود.

توجه: همانطور که مشاهده می‌شود، در مثال فوق به علت **یکپارچگی داده‌ها** (Integration)، پدیده **افزونگی طبیعی** را شاهد نیستیم. دقت کنید تکرار مقدار 6037 در جدول حساب قرض الحسنه افزونگی طبیعی نیست چون بخشی از کلید اصلی است. کلید اصلی این جدول ستون‌های شماره مشتری و شماره حساب باهم است.

۲- افزونگی تکنیکی

تکرار بعضی از مقادیر یک یا چند صفت خاصه در محیط ذخیره‌سازی جهت ایجاد یک شیوه دستیابی کاراتر برای جداول را **افزونگی تکنیکی** می‌گویند. مثل **کلید خارجی** برای ارتباط بین جداول یا وقتی که روی صفت خاصه‌ای از یک جدول، **شاخص (Index)** ایجاد می‌کنیم، مقادیر آن صفت در جدول شاخص تکرار خواهد شد.

توجه: مبحث کلید خارجی در فصل مدل رابطه‌ای و مبحث شاخص در فصل SQL و شاخص‌گذاری اطلاعات تشریح می‌گردد.

افزونگی تکنیکی: ستون شماره مشتری در جداول حساب قرض الحسنه و جاری به عنوان **کلید خارجی** جهت ارتباط با جدول مشتری تعریف شده است، که **افزونگی تکنیکی** محسوب می‌شود.

امنیت فیزیکی: در سیستم‌های بانکی به علت **یکپارچگی داده‌ها** در **یک دیسک**، امنیت فیزیکی داده‌ها در سطح مناسبی برقرار نیست. زیرا داده‌ها به دلیل یکپارچگی ممکن است به یکباره در معرض **آسیب‌پذیری فیزیکی** قرار گیرند. آسیب‌پذیری فیزیکی یعنی مثلا دیسک ضربه بخورد.

البته در اغلب موارد از داده‌ها در دیسک‌های دیگر پشتیبان تهیه می‌شود، که در این حالت مساله امنیت فیزیکی هم وجود نخواهد داشت. در نهایت باید مراقب باشیم پایگاه داده ضربه منطقی و فیزیکی نخورد.

توجه بسیار مهم: یکپارچگی یعنی رعایت اصول تحلیل (مدل ER) و طراحی (مدل رابطه‌ای) که نتیجه آن **جداول نرمال** است و اگر به درستی انجام نشود نتیجه آن **جداول غیرنرمال** است که دچار **تکرار اطلاعات** و به تبع دچار **افزونگی طبیعی** است و از آن **رنج** می‌برد که با عملیات **نرمال سازی** باید **درمان** شود. به مثال زیر دقت کنید:

مثال: جدول غیرنرمال studclg با مقادیر زیر را در نظر بگیرید:

S#	Sname	Clg#	Clgname
8621	Ali	12	Computer
8442	Reza	12	Computer
8731	Abbas	NULL	NULL
9215	Hassan	13	Bargh

← افزونگی طبیعی

در جدول studclg این اطلاعات که «کد دانشکده‌ی کامپیوتر برابر 12 است.» در سطرهای اول و دوم تکرار شده است، پس **افزونگی طبیعی** وجود دارد. اگر شخصی بخواهد کد دانشکده‌ی کامپیوتر را به 22 تغییر دهد و به اشتباه، فقط مقدار #Clg در سطر اول جدول studclg را عوض کند، از این به بعد دانشکده کامپیوتر دو کد خواهد داشت: 12 و 22 یعنی بالاخره کد دانشکده کامپیوتر 12 هست یا 22 ؟ این موضوع یعنی **ناسازگاری** یا **بی‌نظمی یا آنومالی یا ناهنجاری**. به این مدل ناسازگاری، **ناسازگاری ارجاعی** هم گفته می‌شود چون یک مقدار است که همه جا باید یکسان باشد ولی در اثر تغییر در جدول غیرنرمال همه جا یکسان نیست. پدیده ناسازگاری ارجاعی با نرمال‌سازی و تعریف کلید خارجی مرتفع می‌شود. **ناسازگاری داده‌ای** انواع مختلف دارد که به مرور تشریح خواهیم کرد. **سازگاری یعنی وجود داده‌های درست در پایگاه داده.**

در سطر ۳ از جدول فوق، مقادیر NULL وجود دارد. به طور کلی، به ازای هر سطر از جدول غیرنرمال studclg که #Clg آن برابر NULL باشد یک سطر در جدول studclg خواهیم داشت که مقادیر دو ستون آخر آن NULL خواهد بود. پس مشکل مقادیر NULL نیز وجود دارد.

بنابراین اگر عملیات یکپارچگی بر اساس اصول تحلیل (مدل ER) و طراحی (مدل رابطه‌ای) انجام نشود، مشکلات زیر را خواهیم داشت:

۱- افزونگی طبیعی (Data Redundancy)

۲- ناسازگاری یا آنومالی یا بی‌نظمی یا ناهنجاری (Anomaly)

۳- مقادیر تهی (NULL Values)

حال اگر جدول studclg را مطابق اصول نرمال‌سازی به دو جدول clg و stud تجزیه کنیم:

Clg#	Clgname
12	Computer
13	Bargh

جدول clg

S#	Sname	Clg#
8621	Ali	12
8442	Reza	12
8731	Abbas	NULL
9215	Hassan	13

جدول Stud

مشکلات فوق مرتفع و نتایج زیر حاصل خواهد شد:

- کاهش افزونگی طبیعی (محتوایی) مانند تکرار بی‌دلیل سطر (12,Computer)
- کاهش مقادیر NULL
- سازگاری
- افزایش افزونگی تکنیکی به دلیل تعریف کلید خارجی (در قالب ستون های مشترک)

توجه: هرجا جدول دیدیم به آن نمی‌گوییم پایگاه داده نرمال، بلکه هرجا جدول نرمال دیدیم به آن می‌گوییم پایگاه داده نرمال.

توجه: مبحث نرمال‌سازی در فصل نرمال‌سازی تشریح می‌گردد.

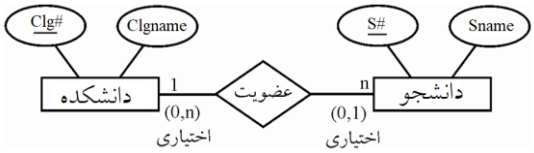
توجه: دقت کنید نرمال‌سازی زمانی کاربرد دارد که خود مدل رابطه‌ای در سطح مدل طراحی به شکل بهینه انجام نشود، برای مثال رابطه یک به چند بین دو موجودیت در یک ERD یک مدل رابطه‌ای **بهینه** دارد. که اگر عمل نگاشت مدل تحلیل به مدل طراحی بهینه انجام شود جداول حاصل خود به خود نرمال هستن و نیاز به نرمال‌سازی ندارد.

نگاشت رابطه یک به چند بین دو موجودیت به مدل رابطه‌ای

مستقل از اختیاری یا اجباری بودن موجودیت‌ها، هر موجودیت به یک جدول تبدیل می‌گردد. و کلید کاندید جدول یک در جدول چند به عنوان کلید خارجی تعریف می‌گردد.

روال کلی نگاشت در این حالت به صورت زیر است:

مدل تحلیل:



مدل طراحی:

S#	Sname	Clg#
8621	Ali	12
8442	Reza	12
8731	Abbas	NULL
9215	Hassan	13

جدول Stud

Clg#	Clgname
12	Computer
13	Bargh

جدول clg

توجه: اگر دانشجویی عضو دانشکده‌ای نباشد، ستون #Clg برای آن مقدار NULL می‌گیرد، مانند شماره دانشجویی 8731 در جدول Stud. همچنین #Clg در جدول Stud **کلید خارجی** است.

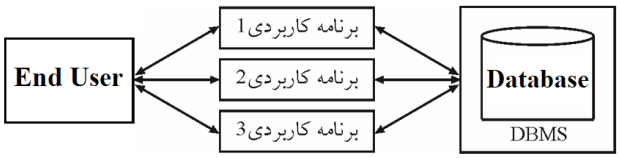
توجه: همانطور که مشاهده می‌کنید اگر اصول تحلیل (مدل ER) و طراحی (مدل رابطه‌ای) رعایت شود. جداول حاصل، خود به خود نرمال هستن و نیاز به نرمال‌سازی ندارد.

ب) مولفه نرم‌افزار

در محیط پایگاه داده دو دسته نرم‌افزار وجود دارد:

نرم‌افزار کاربردی: واسط بین کاربر نهایی (End user) و سیستم مدیریت پایگاه داده‌ها (DBMS) است. این نرم‌افزار به کمک یک زبان سطح بالا و یک زبان داده‌ای (مانند SQL) ایجاد می‌گردد.

نرم‌افزار سیستمی: واسط بین نرم‌افزار کاربردی و پایگاه داده (Database) است. نرم‌افزار سیستمی از دو جزء DBMS و سیستم عامل تشکیل شده است. DBMS که نرم‌افزاری سیستمی است میهمان یک سیستم عامل است و از امکانات سیستم عامل در انجام وظایفش استفاده می‌کند.



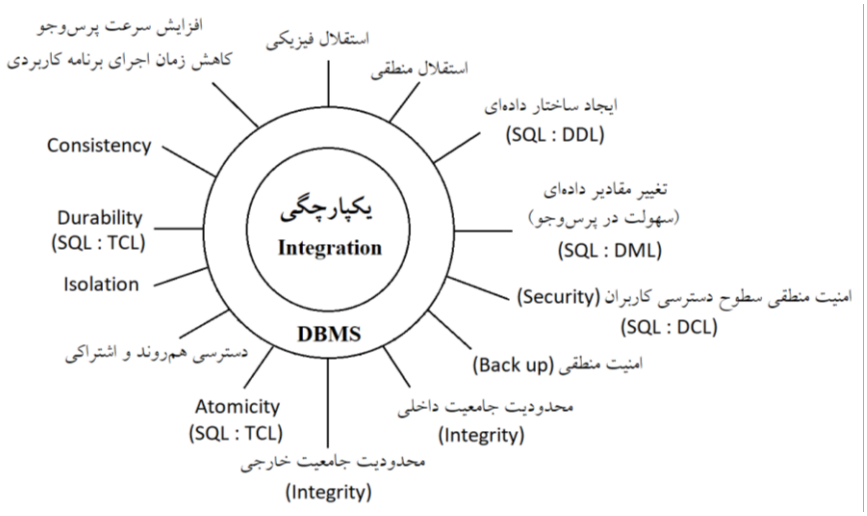
تعریف DBMS (Database Management System): سیستم مدیریت پایگاه داده یا **سمپاد**، یک **نرم افزار سیستمی** است. DBMS مجموعه‌ای از **کارهای امنیتی** و **اجرایی** را انجام می‌دهد تا پایگاه داده **سازگار** و **پایدار** بماند. نرم‌افزارهای سیستمی SQL-Server و Oracle نمونه‌های مطرح DBMS هستند. DBMS باعث شده است تا طراحان و برنامه‌نویسان از برنامه‌نویسی روالی برای مدیریت داده‌ها بی‌نیاز شوند. چون BDMS تمامی **سرویس‌های مرتبط** با **مدیریت پایگاه داده** را به صورت سرویس‌های از قبل آماده در اختیار طراحان و برنامه‌نویسان قرار می‌دهد.

توجه: کاربران نهایی (End user) و برنامه‌کاربردی (واسط کاربر) در بخش User قـرار دارنـد و از طریق DBMS با Database در ارتباط هستند.

توجه: DBMS به طراحان پایگاه داده می‌گوید، تو تعیین و طراحی کن، ایجاد و کنترل با من.

تو تعیین میکن و در دجله انداز که DBMS در بیابانت دهد باز

شکل زیر گویای مطلب است:



سرویس‌های ۱۴ گانه DBMS

در ادامه تمامی سرویس‌های DBMS از مورد ۱ تا ۱۴ معرفی می‌شود:

I. **استقلال فیزیکی داده‌ای:** با استفاده از Create Table برای ایجاد جدول

II. **استقلال منطقی داده‌ای:** دستور Create View باعث افزایش استقلال منطقی می‌شود.

استقلال داده‌ای: مهمترین خصیصه استفاده از DBMS در سیستم‌های بانکی، **اسـتقلال** برنامـه‌های کاربردی از جنبه و خصوصیات فیزیکی ذخیره‌سازی داده‌ها است. زیرا کلیه تعاریف و ایجاد داده‌ها و جداول در DBMS انجام می‌گیرد و مانند سیستم‌های فایلینـگ تعـاریف فایل‌هـای داده‌ای درون برنامه‌های کاربردی قرار ندارد که این موضوع به معنی استقلال داده‌ای است.

شمای بانک اطلاعات (Database schema)

اسکیما (شما) در واقع شکل ساختار و سازماندهی داده‌ها در پایگاه داده است. به عبارت ساده‌تر، اسکیما تعریف می‌کند که هر جدول در بانک اطلاعات چه ستون‌هایی دارد، نوع داده هر ستون چیست، کدام ستون کلید اصلی است و روابط بین جداول چگونه است. به طور خلاصه، schema نقشه یا **طرحواره بانک اطلاعات** است. به مجموعه ساختارهای طراحی شده در یک بانک بدون توجه به داده‌هایی که در آن‌ها قرار می‌گیرد، شمای (Schema) پایگاه داده گفته می‌شود، برای مثال در مدل رابطه‌ای، شمای یک پایگاه داده را جداول تشکیل می‌دهد یا نوع داده هر ستون به شمای بانک مربوط می‌شود ولی تعداد سطرها و مقادیر موجود در جداول ربطی به شمای پایگاه داده ندارد.

توجه: هر پایگاه داده دارای قالب یا شما (Schema) است و مجموعه‌ای از داده‌های ذخیره شده در پایگاه داده **در یک زمان خاص** (snapshot) را **نمونه پایگاه داده** (Database Instance) می‌نامند. شمای پایگاه داده مانند تعریف نوع متغیر و نمونه شمای پایگاه داده، مشابه مقدار متغیر در یک لحظه خاص است.

اجزای اصلی یک اسکیما (شما)

- **جدول‌ها (Tables)** : هر جدول نماینده یک موجودیت است. (مثلا موجودیت تولیدکننده)
- **ستون‌ها (Columns)** : هر ستون یک ویژگی از موجودیت را نشان می‌دهد. (مثلا نام تولیدکننده)
- **نوع داده (Data Types)** : نوع داده هر ستون مشخص می‌کند که چه نوع اطلاعاتی می‌تواند در آن ستون ذخیره شود. (مثلا عدد صحیح، متن، تاریخ)
- **کلید اصلی (Primary Key)** : ستونی که به طور منحصر به فرد هر رکورد را شناسایی می‌کند.
- **کلید خارجی (Foreign Key)** : ستونی که به کلید کاندید (اصلی یا فرعی) جدول دیگری اشاره می‌کند و رابطه بین جداول را برقرار می‌کند.
- **شاخص‌ها (Indexes)** : ساختارهایی هستند که برای افزایش سرعت جستجو در داده‌ها استفاده می‌شود.
- **قیدها یا محدودیت‌ها (Constraints)** : قوانین و محدودیت‌هایی هستند که برای حفظ یکپارچگی داده‌ها در بانک اطلاعات اعمال می‌شود. (مثلا ستون کلید اصلی نمی‌تواند مقدار تهی یا تکراری داشته باشد)

چرا اسکیما مهم است؟

- **سازماندهی داده‌ها:** اسکیما به شما کمک می‌کند تا داده‌های خود را به صورت منظم و قابل فهم سازماندهی کنید.
- **بهبود عملکرد:** یک اسکیما خوب می‌تواند به بهبود عملکرد پایگاه داده کمک کند.
- **حفظ یکپارچگی داده‌ها:** اسکیما با تعریف قیدها، از وارد شدن داده‌های نادرست به پایگاه داده جلوگیری می‌کند.
- **تسهیل در توسعه نرم‌افزار:** اسکیما به توسعه‌دهندگان نرم‌افزار کمک می‌کند تا با بانک اطلاعات تعامل برقرار کنند.

طراحی اسکیما

طراحی یک اسکیما مناسب یکی از مهم‌ترین مراحل در ایجاد یک پایگاه داده است. برای طراحی یک اسکیما خوب باید به موارد زیر توجه کرد:

- **نیازهای کاربردی:** اسکیما باید به گونه‌ای طراحی شود که نیازهای کاربردی سیستم را برآورده کند.
- **انعطاف‌پذیری:** اسکیما باید به گونه‌ای طراحی شود که بتوان آن را در آینده توسعه داد.
- **کارایی:** اسکیما باید به گونه‌ای طراحی شود که عملکرد پایگاه داده را بهینه کند، برای مثال منجر به کاهش **افزونگی طبیعی** شود.

ابزارهای طراحی اسکیما

برای طراحی اسکیما از ابزارهای مختلفی مانند نرم‌افزارهای طراحی پایگاه داده، زبان‌های تعریف داده (DDL) و ابزارهای مدل‌سازی داده (مدل رابطه‌ای) استفاده می‌شود.

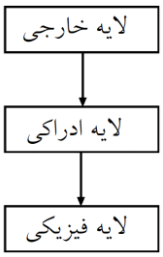
استقلال داده‌ای

یکی از مهم‌ترین مزایای تکنولوژی پایگاه داده (مدل مفهومی پایگاه داده)، بلکه مهمترین هدف آن تأمین و **افزایش استقلال داده‌ای** است، به معنی وابسته نبودن برنامه‌های کاربردی به داده‌های ذخیره شده. استقلال داده‌ای بر دو نوع می‌باشد:

۱- استقلال فیزیکی داده‌ها

به معنی مصونیت لایه خارجی (برنامه کاربردی) و لایه ادراکی (مدل تحلیل و طراحی) در قبال تغییراتی که در سطح لایه فیزیکی (رسانه ذخیره‌سازی) پایگاه داده بروز می‌کند. یعنی اگر تغییری در لایه فیزیکی انجام گیرد (نوع دیسک عوض شود) لایه خارجی و لایه ادراکی نیاز به هیچ تغییری نداشته‌باشد. بنابراین در معماری ANSI لایه خارجی و لایه ادراکی نسبت به تغییرات لایه فیزیکی به صورت **کامل و 100 درصد استقلال** دارد.

توجه: دستور **Create Table** در SQL باعث ایجاد استقلال فیزیکی می‌شود. زیرا پایگاه داده از تجریدی (انتزاع) به نام جدول استفاده می‌کند و داده‌ها هر چه باشند در قالب چند جدول ریخته می‌شود و نحوه ذخیره‌سازی داده‌ها روی رسانه دیسک در لایه فیزیکی از دید لایه خارجی و لایه ادراکی مخفی است. شکل زیر گویای مطلب است:

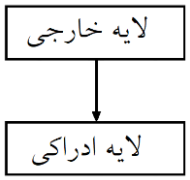


۲- استقلال منطقی داده‌ها

به معنی مصونیت لایه خارجی (برنامه کاربردی) در قبال تعاریف و تغییراتی که در لایه ادراکی (مدل تحلیل و طراحی) پایگاه داده بروز می‌کند. یعنی تعریف و تغییر لایه ادراکی از دید لایه خارجی مخفی بماند.

از آنجاکه لایه خارجی براساس لایه ادراکی تعریف می‌شود، بنابراین به طور بالقوه در معرض تأثیرپذیری از **تغییرات** در لایه ادراکی قرار دارد. لایه خارجی نسبت به **تغییرات جزئی** لایه ادراکی (مثل تغییر در نوع صفات خاصه، تغییر در اندازه صفات) استقلال منطقی دارد و پایدار است، اما نسبت به **تغییرات کلی** مثل حذف یک جدول، دیگر استقلال منطقی ندارد، زیرا برنامه کاربردی با جدول حذف شده در حال تعامل بوده، حال اینکه جدول دیگر وجود ندارد. با حذف یک جدول، برنامه کاربردی دیگر به داده‌های آن دسترسی ندارد.

بنابراین در معماری ANSI لایه خارجی (برنامه کاربردی) نسبت به تغییرات لایه ادراکی به صورت **ناقص** (نسبی) استقلال دارد.



توجه: در سیستم‌های امروزی، این نوع استقلال هم تا حدی (و نه صددرصد) تأمین شده است.

توجه: دستور **Create View** در SQL باعث **افزایش استقلال منطقی** می‌شود که در ادامه با مثال بررسی می‌کنیم.

انواع تغییر در مدل طراحی (طراحی منطقی یا ادراکی خاص)

۱- **رشد پایگاه داده‌ها به دلیل مطرح شدن نیازهای جدید مشتری**: مانند تعریف و درج جدول جدید، ترکیب جداول، تجزیه جداول به معنی جایگزینی یک جدول با چند جدول دیگر و پخش داده‌های موجود در جداول جدید

۲- **سازماندهی مجدد:** مانند تغییر در نوع صفات خاصه، تغییر در اندازه صفات.

مثال: اگر جدولی دارای چهار ستون باشد و ستون پنجمی نیز به آن اضافه گردد، در صورتی که برنامه کاربردی سابق نیاز به دستکاری و تغییر نداشته باشد، استقلال منطقی داده‌ها براساس تغییرات نیز لحاظ شده است.

توجه: از آن جا که با حذف جداول، داده‌ها هم از بین می‌رود، بنابراین برنامه‌های کاربردی نسبت به حذف جداول **هیچگاه** استقلال منطقی نخواهند داشت. بنابراین برنامه‌های کاربردی نسبت تغییرات پایگاه داده استقلال منطقی حداکثری دارد اما صددرصد نیست.

تکامل شمای داده (Schema Evolution)

طراحی پایگاه داده به روش ساخت‌یافته یا شی‌گرا یک فعالیت یک‌باره و برای همیشه ماندگار نیست. تغییر و توسعه، جزء جدایی ناپذیر محصولات نرم‌افزاری است. نیازهای یک سازمان به‌طور مداوم تکامل می‌یابد و داده‌هایی که نیاز به ذخیره‌سازی آن‌ها دارد نیز به‌طور متناظر تغییر می‌کنند و به تبع منجر به تغییر ساختار جداول می‌شود و در نهایت ممکن است منجر به تغییرات برنامه‌کاربردی شود. بنابراین تکامل شمای (ساختار یا اسکیما) داده در طول زمان بسته به تغییر نیازهای کسب‌وکار امری طبیعی است. تکامل شمای داده فقط مختص روش ساخت‌یافته (تابع‌گرا) یا شی‌گرا (کلاس‌گرا) نیست، بلکه پایگاه داده و برنامه کاربردی مرتبط به آن چه به روش ساخت‌یافته و با ابزارهای مدل ER ایجاد شود و چه به روش شی‌گرا و با ابزارهای مدل UML ایجاد شود قطعا در طول زمان نیاز به توسعه و تکامل دارد. اما آنچه اهمیت دارد این است که شمای داده اولیه چه به روش ساخت‌یافته و چه به روش شی‌گرا طوری نرمال‌سازی و طراحی شود که امکان توسعه‌های بعدی را داشته باشد. بدیهی است که تغییرات ساختار پایگاه داده منجر به تغییرات برنامه‌کاربردی خواهد شد. اما در یک طراحی پایگاه داده نرمال، تلاش این است که تغییرات در ساختار پایگاه داده منجر به حداقل تغییرات در برنامه کاربردی شود. و این همان معنای استقلال داده‌ای است. یک طراحی پایگاه داده نرمال، نیازهای آینده یک سازمان را پیش‌بینی می‌کند و اطمینان می‌دهد که شمای داده حداقل تغییرات را در هنگام تکامل نیازها نیاز دارد.

توجه: تغییر و توسعه Schema بدون اختلال، در دسترسی کاربران به پایگاه داده و بدون از دست دادن یا آسیب به داده‌های موجود، یک چالش است و این چالش در اغلب موارد قابل

حل و کنترل است. اما این چالش آنقدرها بزرگ نیست که منجر به تغییر کل ساختار پایگاه داده شود تا به تبع منجر به تغییر کل برنامه‌کاربردی شود. تکامل شِمای داده به تغییر و تکامل ساختار پایگاه داده مرتبط است و این موضع ارتباطی به محتوای داخلِ جداول پایگاه داده به معنی تغییر داده‌های ذخیره شده در پایگاه داده ندارد.

جمله تاکیدی: هرچه میزان استقلال منطقی ساختار پایگاه داده بیشتر باشد، آنگاه تکامل شِمای داده‌ها آسان‌تر و راحت‌تر و با چالش کمتری همراه خواهد بود.

افزایش استقلال داده‌ای به کمک Veiw

همانطور که گفتیم یک محصول نرم‌افزاری از دو وجه **داده** (بانک اطلاعات) و **عملکرد** (برنامه‌کاربردی) و تشکیل شده است، **بخش داده** که با SQL پیاده‌سازی می‌شود به مفاهیم **استقلال داده‌ای** مرتبط است. **ساختار داده** توسط دستورات ساختاری DDL نظیر Create Table، Create View و Create Index و دیگر دستورات آن ایجاد و مدیریت می‌گردد. و **مقادیر وجه داده** توسط دستورات DML نظیر Insert، Delete، Update و Select و دیگر دستورات آن ایجاد و مدیریت می‌گردد.

دستور **Create View** با ساخت مفهوم دید، تا حدی کمک به برقراری **استقلال منطقی** داده‌ها میان لایه خارجی (برنامه‌کاربردی) و لایه ادراکی (مدل تحلیل و طراحی) می‌کند، به معنی وابسته نبودن برنامه‌کاربردی به لایه ادراکی، یعنی همانطور که گفتیم، اگر جدولی دارای چهار ستون باشد و ستون پنجمی نیز به آن اضافه گردد، در صورتی که برنامه کاربردی جاری نیاز به دستکاری و تغییر نداشته باشد، استقلال منطقی داده‌ها براساس تغییرات نیز لحاظ شده است. از آنجاکه View روی ساختار قدیم شامل نام جدول قدیم و ستون‌های قدیم ایجاد می‌شود، اگر جدولی دارای چهار ستون باشد و ستون پنجمی نیز به آن اضافه گردد، آنگاه بدون تغییرات در ساختار View در بخش داده و به تبع تغییرات در ساختار بخش عملکرد (برنامه کاربردی)، امکان حیات برنامه کاربردی بدون اشکال همچنان وجود دارد و این یعنی View حافظ استقلال داده‌ای از نوع استقلال منطقی داده‌ها است. دقت کنید که View بخشی از وجه داده است. در واقع **ساختار وجه داده** از بخش‌های مختلف Table، View و Index تشکیل شده است.

مثال: پیاده‌سازی جدول S به صورت زیر را در نظر بگیرید:

```
Create Table S
    (
    S# char (5),
    Sname char (20),
    Primary key (S#)
    )
```

توجه: ساخت جدول با دستور Create Table منجر به استقلال فیزیکی می‌شود.

جدول S با مقادیر زیر را در نظر بگیرید:

S#	Sname
S1	Sn1
S2	Sn2
S3	Sn3

پرس‌وجوی زیر را در نظر بگیرید:

```
Select *
From S
```

خروجی پرس‌وجو به صورت زیر است:

S#	Sname
S1	Sn1
S2	Sn2
S3	Sn3

پیاده‌سازی View با نام Report1 را روی جدول S به صورت زیر در نظر بگیرید:

```
Create View Report1 (SID , SN)
AS Select S# , Sname
    From S
```

پرس‌وجوی زیر بر روی View با نام Report1 را به صورت زیر در نظر بگیرید:

```
Select *
From Report1
```

خروجی پرس‌وجو بر روی View با نام Report1 به صورت زیر است:

SID	SN
S1	Sn1
S2	Sn2
S3	Sn3

اضافه‌شدن ستون City به جدول S را به صورت زیر در نظر بگیرید:

```
Alter Table S add City char(15)
```

ساختار و مقادیر جدول S پس از اضافه‌شدن ستون City به صورت زیر است:

S#	Sname	City
S1	Sn1	NULL
S2	Sn2	NULL
S3	Sn3	NULL

مجددا پرس‌وجوی زیر بر روی View با نام Report1 را به صورت زیر در نظر بگیرید:

```
Select *
From Report1
```

خروجی پرس‌وجو بر روی View با نام Report1 به صورت زیر است:

SID	SN
S1	Sn1
S2	Sn2
S3	Sn3

توجه: همانطور که واضح است، از آنجاکه View روی ساختار قدیم شامل نام جدول قدیم و ستون‌های جدول قدیم ایجاد می‌شود، اگر جدولی دارای دو ستون باشد و ستون سومی نیز به آن اضافه گردد، آنگاه بدون تغییرات در ساختار View در بخش داده و به تبع تغییرات در ساختار بخش عملکرد (برنامه کاربردی)، امکان حیات برنامه کاربردی بدون اشکال همچنان وجود دارد و این یعنی **ساختار View حافظ استقلال داده‌ای از نوع استقلال منطقی داده‌ها است.**

حال برگردیم به بررسی ادامه سرویس‌های DBMS از مورد ۳ تا ۱۴:

III. **ایجاد ساختارهای داده‌ای:** با استفاده از Create Table برای ایجاد جدول، Create View برای ایجاد دید و Create Index برای ایجاد شاخص از دستورات DDL

IV. **تغییر مقادیر داده‌ای و سهولت در پرس‌وجونویسی بیانی:** با استفاده از Select برای خواندن، Insert برای درج، Delete برای حذف و Update برای بروزرسانی از دستورات DML

سهولت پرس‌وجونویسی: در سیستم‌های بانکی پرس‌وجونویسی توسط زبان‌های بیانی مثل SQL انجام می‌شود، بنابراین سهولت در پرس‌وجونویسی را دارد.

V. **کنترل امنیت منطقی با تعیین محدودیت‌های امنیتی (Security constraints): با** استفاده از Grant برای اعطای مجوز دسترسی و Revoke برای بازپس‌گیری مجوز دسترسی از دستورات DCL

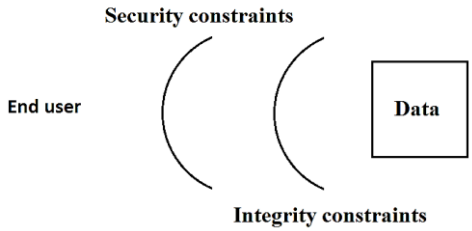
توجه: در سیستم‌های بانکی برقراری امنیت منطقی داده‌ها در برابر دستیابی غیرمجاز (کپی، سرقت، درج، حذف و آپدیت غیرمجاز) به دلیل وجود DBMS و تعریف سطوح امنیتی برای کاربران در

سطح بسیار مناسبی قرار دارد.

محدودیت‌های امنیتی (Security constraints)

یکی از مهمترین موضوعات سیستم‌های بانک اطلاعات، بحث کنترل دسترسی‌های کاربران به بانک اطلاعات و حفاظت داده‌ها در برابر دسترسی‌های غیرمجاز است. محدودیت‌های امنیتی از دسترسی **کاربران غیرمجاز** به پایگاه داده محافظت می‌کند. دقت کنید که در محدودیت‌های امنیتی تاکید روی **کاربران مجاز** است. برای مثال کاربر Ali به صورت فقط خواندنی حق دسترسی به جدول S دارد. محدودیت‌های امنیتی از سه بخش **who** چه کسی؟ Ali، **what** چه چیزی؟ جدول S و **how** چگونه؟ به صورت فقط خواندنی تشکیل شده است.

توجه: گارد یا سپردفاعی **محدودیت‌های امنیتی** (Security constraints) در لایه اول حفاظتی جلوی End user قرار دارد.



توجه: دستورات کنترل داده‌ها (DCL: Data Control Language) در SQL- Server کارهای مربوط به تعریف **محدودیت‌های امنیتی** جداول پایگاه داده از قبیل ایجاد سطوح دسترسی برای کاربران را انجام می‌دهد. دستورات DCL به دستورات Authorization نیز موسوم هستند.

VI. **کنترل امنیت منطقی با تهیه** Backup در رسانه پایدار (RAID)

VII. **کنترل محدودیت‌های جامعیتی (Integrity constraints) داخلی**: کنترل مقادیر ورودی برای نوع داده هر ستون برای **تضمین جامعیت دامنه‌ای**، کنترل مقادیر ورودی کلید اصلی برای **تضمین جامعیت درون رابطه‌ای**، کنترل مقادیر کلید خارجی برای **تضمین جامعیت ارجاعی**، کنترل Not NULL بودن ستون کلید اصلی برای **تضمین جامعیت موجودیت**

VIII. **کنترل محدودیت‌های جامعیتی (Integrity constraints) خارجی**: کنترل قواعد کسب و کار یا Business rules مثل کنترل درج نمره 0 تا 20 برای دانشجویان، کنترل اینکه تاریخ رفت پرواز قبل از تاریخ برگشت باشد، موجودی حساب منفی نباشد.

محدویت‌های جامعیتی داخلی و خارجی (Integrity constraints)

محدودیت‌های جامعیتی داخلی و خارجی تضمین می‌کنند که **عملیات** و تغییرات که بر روی پایگاه داده انجام می‌شود، منجر به از دست رفتن **سازگاری داده‌ها** نشود. دقت کنید که در **محدودیت‌های جامعیتی** تاکید روی **عملیات مجاز** است. هدف از محدودیت‌های جامعیتی جلوگیری از ورود داده‌های نادرست به پایگاه داده است.

توجه: گارد یا سپر دفاعی **محدودیت‌های جامعیتی** (Integrity constraints) در لایه دوم حفاظتی جلوی End user قرار دارد. برای مثال کاربر Ali مجاز است به جدول دانشجویان دسترسی داشته باشد، این محدودیت امنیتی (**افراد مجاز**) است، اما دیگر نگفتیم که برو داخل جدول دانشجویان هر کاری دلت خواست انجام بدی و نمره 22 وارد کنی، این محدودیت جامعیتی (**عملیات مجاز**) است.

توجه: در نهایت ترکیب **محدودیت‌های امنیتی** و **محدودیت‌های جامعیتی** منجر به این می‌شود که **کاربران مجاز**، **عملیات مجاز** روی پایگاه داده انجام دهند.

IX. **تضمین خاصیت Atomicity برای اجرای کامل و اتمیک تراکنش‌ها:** با استفاده از Commit و Rollback از دستورات TCL

X. **امکان دسترسی هم‌روند و استفاده اشتراکی کاربران از داده‌ها**

دسترسی همزمان و اشتراکی: با توجه به ماهیت سیستم‌های بانک اطلاعات و مبتنی بر شبکه بودن، کاربران مختلف می‌توانند به صورت همزمان و اشتراکی با بانک کار کنند، یعنی هر کاربر بدون ایجاد محدودیت برای کاربر دیگر در هر لحظه می‌تواند با بانک کار کند. البته هر کاربر دید خاص و حق دسترسی مختص به خود را نسبت به داده‌ها دارد، به دلیل تعریف سطوح دسترسی برای کاربران توسط DBMS.

XI. **تضمین خاصیت Isolation برای مدیریت تراکنش‌های هم‌روند:** با استفاده از عایق‌سازی تراکنش‌ها بدین معنی که در هم‌روندی روی هم اثر مخرب نگذارند.

XII. **تضمین خاصیت Durabiltiy برای ذخیره نتایج نهایی در دیسک، جهت پایایی تراکنش‌ها:** با استفاده از Commit و Rollback از دستورات TCL

XIII. **تضمین خاصیت Consistency برای حفظ سازگاری پایگاه داده**

سازگاری داده‌ای: در سیستم‌های بانکی سازگاری داده‌ای توسط DBMS تضمین می‌شود. ناسازگاری دلایل مختلف دارد که در ادامه فصل بررسی می‌شود. **سازگاری یعنی وجود داده‌های درست در پایگاه داده.**

XIV. **افزایش سرعت پرس‌وجوها با تعریف شاخص:** با استفاده از Create Index برای

ایجاد شاخص از دستورات DDL که منجر به کاهش زمان اجرای برنامه کاربردی می‌شود.

زمان اجرای برنامه کاربردی: بستگی به شرایط مختلف دارد، البته در حالت کلی سیستم بانکی مکانیزم ویژه‌ای (مثل استفاده از شاخص و درخت جستجو) برای کاهش زمان اجرای برنامه کاربردی دارد. اما در شرایط خاص در پرس‌وجوهای تودرتو با حجم زیاد داده‌ها و به تبع تحمیل سربار حاصل از الحاق جداول، سیستم بانکی زمان اجرای بیشتری نسبت به سیستم پرونده‌ای خواهد داشت.

همانطور که پیش‌تر گفتیم، یک سیستم پایگاه داده از ۴ مولفه اساسی تشکیل شده است که هر یک مسئولیتی را بر عهده گرفته است، این مولفه‌ها شامل (الف) **مولفه ساختار داده‌ای** (ب) **مولفه نرم‌افزار**، (ج) **مولفه کابران** و (د) **مولفه سخت‌افزار** است که در ادامه به شرح مولفه کاربران و مولفه سخت‌افزار می‌پردازیم:

ج) مولفه کاربران

در محیط بانک اطلاعاتی چهار نوع کاربر وجود دارد:

۱) مدیر داده‌ها (DA : Data Administrator)

انجام مدل تحلیل یا همان طراحی ادراکی(مفهومی) بر عهده DA است، مدیر داده‌ها یک شخص مدیریتی است و نه یک شخص فنی. DA شخصی است که مسئولیت کنترل اصلی بر روی داده‌ها را بر عهده دارد. وظیفه مدیر داده‌ها آن است که در مرحله اول تصمیم بگیرد که چه داده‌هایی باید در بانک اطلاعاتی ذخیره شود و سپس هنگامی که داده‌ها ذخیره شدند، سیاست‌گذاری‌های لازم را جهت نگهداری و کار با آن‌ها تعیین کند. برای مثال اینکه چه کسی می‌تواند چه اعمالی را بر روی چه داده‌هایی انجام دهد، به معنی تعیین سطوح دسترسی به داده‌ها. **DA رهبر پایگاه داده جهت تعیین سیاست‌های کلی نظام است.**

وظایف مدیر داده‌ها (DA)

- انجام تحلیل موجودیت‌ها توسط مدل ER.
- انجام تحلیل پرس‌وجوها توسط حساب رابطه‌ای.
- انجام تحلیل محدودیت‌های امنیتی و جامعیتی

۲) مدیر بانک اطلاعات (DBA : Data Base Administrator)

انجام مدل طراحی یا همان طراحی منطقی بر عهده DBA است، مدیر بانک اطلاعات فردی است فنی که پشتیبانی‌های تکنیکی لازم را برای پیاده‌سازی تصمیمات مدیر داده‌ها (DA) فراهم می‌سازد.

وظیفه DBA ایجاد بانک اطلاعات و پیاده‌سازی محدودیت‌های جامعیتی و امنیتی است که سیاست‌گذاری‌های مدیر داده‌ها (DA) را اعمال کند مثل تهیه رویه‌ها و استراتژی تهیه Back up و نحوه احیای پایگاه داده‌ها. همچنین DBA مسئول نظارت بر عملکرد پایگاه داده‌ها (Performance Monitoring) و نظارت بر کارایی و پاسخ به تغییر نیازهاست، بدین معنی که DBA مسئول سازماندهی سیستم برای بدست آوردن بهترین کارایی برای سازمان می‌باشد و همزمان با تغییر نیازهای سازمان، تنظیمات متناسب با نظارت DA اعمال می‌گردد. همچنین DBA مجموعه‌ای از برنامه‌نویسان و سایر افراد فنی را همچون DBPها جهت پیاده‌سازی کارها در اختیار دارد. DBA رئیس جمهور پایگاه داده است. برای مثال رهبر (DA) گفته است مردم درس بخوانند حال اینکه چگونه بخوانند و در چه مقاطعی و با چه نمراتی بر عهده DBA است.

وظایف مدیر بانک اطلاعات (DBA)

- انجام طراحی جداول توسط مدل رابطه‌ای. (مثل تهیه Schema)
- انجام طراحی پرس‌وجوها توسط جبر رابطه‌ای.
- انجام طراحی محدودیت‌های امنیتی (میزان دسترسی کابران) و جامعیتی (مثل تعیین نمره 0 تا 20 برای دانشجویان یا تعیین استراتژی تهیه Back up)
- نظارت بر عملکرد پایگاه داده‌ها (Performance Monitoring)

توجه: تهیه Back up از پایگاه داده و نحوه ترمیم پایگاه داده توسط DBMS انجام می‌شود، اما تهیه رویه‌ها و استراتژی تهیه Back up و نحوه احیای پایگاه داده‌ها بر عهده DBA است. در واقع DBA فقط نقش نظارتی و راهبردی را در این مورد بر عهده دارد.

۳) برنامه‌نویسان بانک اطلاعات (DBP : Data Base Programming)

انجام فعالیت پیاده‌سازی بر عهده DBP است، برنامه‌نویسانی هستند که از طریق دستورات DDL، DML، DCL و TCL در SQL با بانک اطلاعات برای ایجاد ساختار، عملیات درج، حذف و بروزرسانی، عملیات امنیتی و جامعیتی در ارتباط هستند. این افراد دستورات DML را درون یک زبان سطح بالا (زبان میزبان) قرار می‌دهند و برنامه‌های کاربردی را می‌سازند تا نیازهای کاربران نهایی را برای استفاده از داده‌های درون بانک اطلاعاتی بر طرف سازند.

وظایف برنامه‌نویسان بانک اطلاعات (DBP)

- انجام پیاده‌سازی ساختار جدول، دید و شاخص توسط دستورات DDL در SQL.
- انجام پیاده‌سازی پرس‌وجوها توسط دستورات DML در SQL.
- انجام پیاده‌سازی کنترل‌های امنیتی و جامعیتی توسط دستورات DCL و TCL در SQL.

توجه: به طور کلی DA و DBA از جنس تعیین‌کننده (define) هستند و DBP از جنس مجری

است.

۴) کاربر نهایی (End User)

فردی است که هیچ تخصصی در حوزه پایگاه داده ندارد و به عنوان اپراتور صرفا از برنامه‌های نوشته شده استفاده می‌کند. این افراد با سیستم مدیریت پایگاه داده‌ها از طریق **برنامه کاربردی** که توسط DBP ایجاد گردیده در تعامل هستند.

توجه: تنها مدیران (DA و DBA) و برنامه‌سازان (DBP) می‌توانند بدون دخالت DBMS به داده‌ها دسترسی داشته باشند و تمام کاربران نهایی فقط از طریق DBMS به داده‌های داخل بانک اطلاعات دسترسی دارند.

د) مولفه سخت افزار

در محیط بانک اطلاعات سه دسته سخت‌افزار وجود دارد:

الف) سخت‌افزار ذخیره‌سازی اطلاعات: منظور همان رسانه ذخیره‌سازی خارجی است. همانند دیسک که رسانه اصلی ذخیره‌سازی می‌باشد.

ب) سخت‌افزار پردازنده مرکزی: منظور همان کامپیوتر است.

ج) سخت‌افزار ارتباطی: منظور سخت‌افزارهای مورد نیاز جهت ارتباط بین کامپیوتر و دستگاه‌های جانبی و نیز سایر کامپیوترها می‌باشد.

دیکشنری داده‌ها یا کاتالوگ سیستم (Data dictionary or System catalog)

در یک سیستم بانک اطلاعات، اسامی زیادی مورد استفاده قرار می‌گیرد، از آنجایی که افراد زیاد و متفاوتی در یک مجموعه بانک اطلاعات درگیر هستند، مرجعی برای ایجاد یکنواختی و هماهنگی در نام داده‌ها و معنای آنها ضروری است. این مرجع، دیکشنری داده یا کاتالوگ نام دارد.

در اختصار کاتالوگ سیستم یا دیکشنری داده‌ها شامل تمامی اطلاعات سیستمی مربوط به پایگاه داده‌ها همچون مشخصات سیستمی مربوط به اسکیمای (ساختار) جداول و سطوح دسترسی کاربران می‌باشد که به طور خودکار توسط DBMS بروزرسانی می‌گردد. به بیان کامل‌تر هرگونه تغییرات حاصل از دستورات DDL در پایگاه داده همچون ایجاد جداول، حذف جداول، ایجاد شاخص، حذف شاخص، ایجاد View، حذف View و یا هرگونه تغییرات حاصل از دستورات DML در پایگاه داده همچون درج، حذف و بروزرسانی رکوردها که منجر به تغییر تعداد رکوردها و به تبع آن تغییر اندازه جداول پایگاه داده‌ها می‌شود و یا هر گونه تغییرات حاصل از ایجاد و تغییر سطوح دسترسی توسط دستورات DCL در پایگاه داده همچون تخصیص سطوح دسترسی به کاربران و هر آنچه که مربوط به مشخصات سیستمی پایگاه داده باشد در کاتالوگ سیستم نگهداری

می‌شود. به عبارت ساده‌تر، دیکشنری داده‌ها یا کاتالوگ سیستم، **شناسنامه** یک پایگاه داده است که ساختار و محتوای آن را توصیف می‌کند.

توجه: در هر پایگاه داده ۲ طبقه اطلاعات ذخیره می‌شود. **اطلاعات اصلی** و **اطلاعات سیستمی** (کنترلی)، دیکشنری داده در جایگاه خود، پایگاه داده‌ای سیستمی شامل اطلاعات سیستمی در مورد پایگاه داده یک محیط عملیاتی است که حاوی data about data (داده‌هایی درباره داده‌ها) است که گاهی اوقات به نام MetaData (فراداده یا دادگان) معرفی می‌گردد.

توجه: کاتالوگ سیستم به واسطه اجرای تمامی دستورات DDL و DCL و برخی دستورات DML مانند Insert و Delete و نه Select و Update دستخوش تغییر می‌گردد.

محتویات کاتالوگ سیستم

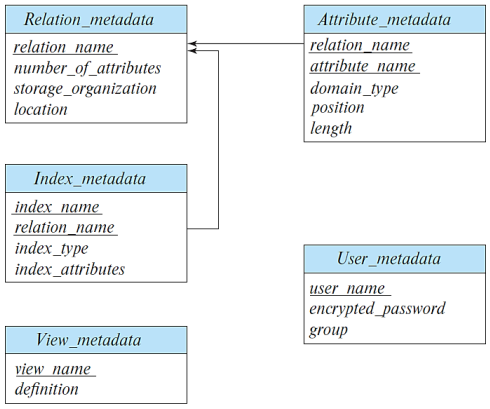
- مشخصات سیستمی جداول (مانند نام جداول، نام و نوع ستون‌های جداول و تعداد و اندازه رکوردها و اندازه جداول)
- مشخصات سیستمی ارتباطات جداول
- مشخصات سیستمی دیدهای بانک اطلاعات (viewها)
- مشخصات سیستمی شاخص‌های بانک اطلاعات (Indexها)
- نام جدولی که ایندکس روی آن تعریف شده است.
- مشخصات سیستمی روال‌های ذخیره شده (Stored Procedure)
- مشخصات سیستمی تراکنش‌های بانک اطلاعات
- مشخصات سیستمی تاریخ ایجاد و بروزرسانی جداول داده‌ای
- مشخصات سیستمی نام کاربران و چگونگی حق دستیابی آنها به داده‌ها مثل رمزهای عبور یا اطلاعات دیگر برای تأیید هویت کاربران و اطلاعات مربوط به مجوزها و محدوده مجاز عملیات آنها برای هر کاربر
- مشخصات محدودیت‌های امنیتی و جامعیتی
- مشخصات سیستمی پایانه‌های متصل به بانک

توجه: تمام این اطلاعات **متادیتا**، در واقع، یک **پایگاه داده کوچک** را تشکیل می‌دهد. برخی از سیستم‌های پایگاه داده چنین متادیتاهایی را با استفاده از ساختارهای داده مخصوص ذخیره می‌کنند. اما معمولا ترجیح داده می‌شود که داده‌های مربوط به پایگاه داده را به عنوان جداول در خود پایگاه داده ذخیره کرد.

توجه: پایگاه داده محیط عملیاتی، محل ذخیره‌سازی **اطلاعات اصلی** و کاتالوگ سیستم محل ذخیره‌سازی **اطلاعات سیستمی** است، بنابراین اطلاعات سیستمی از کاتالوگ سیستم استخراج می‌شود و نیاز به استخراج اطلاعات سیستمی از پایگاه داده اصلی نیست. پس استفاده از کاتالوگ

سیستم باعث افزایش استقلال داده‌های سیستمی از داده‌های اصلی پایگاه داده (Data Independence) می‌شود.

شکل زیر در مورد ساختار کاتالوگ سیستم گویای مطلب است:



Relational schema representing part of the system metadata.

توجه: هرگاه سیستم پایگاه داده نیاز به بازیابی رکوردهای داده‌ای از یک جدول داشته باشد، ابتدا باید به جدول سیستمی Relation metadata در خود کاتالوگ سیستم مراجعه کند تا مکان جدول داده‌ای را پیدا کند و سپس رکوردهای داده‌ای را با استفاده از این اطلاعات بازیابی کند که این مربوط به زمان ترجمه است. اما سرعت ذخیره و بازیابی اطلاعات به سرعت خود حافظه و دیسک و تعریف شاخص وابسته است. کاتالوگ سیستم پس از هر ذخیره و بازیابی اطلاعات مربوط به بانک اطلاعات بروزرسانی می‌گردد. پس کاتالوگ سیستم دخالتی در افزایش یا کاهش سرعت ذخیره و بازیابی اطلاعات نخواهد داشت.

توجه: در برخی کتب غیرمرجع، دیکشنری داده‌ها زیرمجموعه کاتالوگ سیستم در نظر گرفته شده است. اما در کتب مرجع اصلی آبراهام سیلبرشاتز، راما کریشنان، رامزالمصری، سی جی دیت، توماس کانلی و کارولین بگ دیکشنری داده‌ها معادل کاتالوگ سیستم در نظر گرفته شده است.

تراکنش

به دنباله‌ای از عملیات مرتبط به هم در محیط بانک اطلاعات برای انجام یک کار منطقی، **تراکنش** گفته می‌شود. مانند عملیات برداشت از حساب مبدا و واریز به حساب مقصد. تراکنش مجموعه‌ای از یک یا چند دستور SQL است که به عنوان یک واحد منطقی کار در محیط DBMS در نظر گرفته می‌شود. به طور کلی هر عملیاتی در پایگاه داده در قالب یک تراکنش تعریف و اجرا می‌شود. هر تراکنش شامل یک یا چند دستور است. تفاوت اصلی یک تراکنش با یک برنامه

معمولی در محیط غیربانکی این است که تراکنش همواره به DBMS تسلیم می‌شود و DBMS در اعمال هر گونه کنترل و حتی به تعویق انداختن و ساقط کردن آن آزادی عمل دارد. هدف اصلی از اینگونه کنترل‌ها، حذف‌ها و تعویق‌ها، **حفظ سازگاری** بانک اطلاعات است. سیستم پایگاه داده قبل از شروع هر تراکنشی در حالت **سازگار** (Consistent) است، یعنی محتوای داخل جداول پایگاه داده با **محدودیت‌های جامعیت داخلی و خارجی (Integrity Constraints)** تطابق دارد. حال اگر یک تراکنش انجام شود، نتیجه اجرای تراکنش باید سیستم پایگاه داده را از حالت **سازگار قدیم** به حالت **سازگار جدید** انتقال دهد. و اگر اینچنین نشود همه چیز باید به عقب بازگردد.

خواص تراکنش

چهار کنترل زیر لازم است روی تمامی تراکنش‌ها در بانک اطلاعات اعمال گردد تا انجام **کامل** و **درست** آن تضمین شود. این کنترل‌ها به خواص ACID معروفند:

۱- تجزیه‌ناپذیری (Atomicity)

اگر یک تراکنش شروع به **پیشروی** کرد باید همه دستورات آن به طور کامل انجام شود. اما اگر در حین مسیر و میانه راه دچار اختلال شد و به دلایلی شکست خورد و از ادامه انجام دستورات باقی‌مانده ناتوان شد و به تبع اجرای تراکنش ناقص ماند، آنگاه هرگونه تغییری که تراکنش ایجاد کرده است، باید ابطال و لغو شود تا وضعیت پایگاه داده به آخرین وضعیت قبل از اجرای تراکنش بازگردد. بنابراین هر تراکنش باید تجزیه‌ناپذیری را رعایت کند تا در نهایت پایگاه داده را از حالت **سازگار قدیم** به حالت **سازگار جدید** منتقل کند، در غیر اینصورت اجرا و نتایج به دست آمده از آن قابل قبول نیست و همه چیز باید به قبل بازگردد.

این خطاها می‌تواند از انواع مختلفی باشد، مانند:

- خطاهای منطقی در خود تراکنش (مثلاً تقسیم بر صفر)
- خطاهای سخت‌افزاری در سیستم و قطعی برق.
- خطاهای شبکه‌ای که ارتباط بین تراکنش و پایگاه داده را مختل می‌کند.
- قفل‌شدگی تراکنش‌ها (deadlock) که باعث می‌شود دو یا چند تراکنش منتظر یکدیگر شوند و هیچ‌کدام نتوانند پیشروی کنند.

تضمین خاصیت تجزیه‌ناپذیری تراکنش بر عهده DBMS است. که تضمین می‌کند یا همه دستورات اجرا شود و یا هیچکدام از آن دستورات اجرا نشود. به طور کلی هر پرس‌وجویی که **عملیات تغییر** انجام می‌دهد (درج، حذف و آپدیت) در قالب یک تراکنش تعریف و اجرا می‌شود. در SQL برای نمایش ابتدای یک تراکنش از دستور Begin Transaction و برای نمایش خاتمه یک تراکنش از دستور End Transaction استفاده می‌شود. تراکنش با اجرای Begin Transaction

شروع می‌شود و در صورت اجرای موفق commit و در صورت عدم موفقیت یعنی عدم اجرای همه بخش‌های مختلف تراکنش با اجرای دستور Rollback خاتمه می‌یابد. با اجرای این دستور کلیه تغییراتی که تراکنش روی پایگاه داده اعمال نموده است، ابطال می‌شود و وضعیت پایگاه داده به آخرین وضعیت قبل از اجرای تراکنش برگردانده می‌شود. انگار که تراکنش هرگز اجرا نشده است.

توجه: اجرا و نتایج یک تراکنش فقط و فقط زمانی معتبر است که هر چهار شرط ACID باهم برقرار باشد.

مثال: درج یک رکورد

```
BEGIN TRANSACTION;

INSERT INTO Customers (CustomerID, CustomerName, City)
VALUES (1001, 'Ali', 'Tehran');

END TRANSACTION;

COMMIT;
```

توجه: در SQL برای پرس‌وجوهای تک خطی BEGIN و END به طور خودکار درج می‌شود.

مثال: فرض کنید T_i تراکنشی باشد که مبلغ 50 واحد پولی از حساب A به حساب B منتقل می‌کند. فرض کنید درست قبل از اجرای تراکنش T_i ، مقادیر حساب‌های A و B به ترتیب 1000 و 2000 واحد پولی باشد. این تراکنش را می‌توان به صورت زیر تعریف کرد:

T_i : read (A);
A := A − 50;
write (A);
read (B);
B := B + 50;
write (B).

این عملیات شامل دو مرحله است:

الف) بخش اول (برداشت وجه)

پول را از حساب A برداشت می‌کند.

ب) بخش دوم (واریز وجه)

همان پول را به حساب B واریز می‌کند.

اگر این دو عملیات به صورت جداگانه اجرا شوند، ممکن است در اثر مشکلی مانند قطعی برق یا

خطای سیستمی، تنها یکی از آن‌ها انجام شود و در نتیجه حساب‌ها در حالت **ناسازگار** قرار بگیرد.
برای جلوگیری از این مشکل، این دو عملیات را درون یک تراکنش قرار می‌دهیم:

```
BEGIN TRANSACTION;

    UPDATE accounts SET balance = balance - 50 WHERE user_id = 'A';
    UPDATE accounts SET balance = balance + 50 WHERE user_id = 'B';

COMMIT TRANSACTION;

COMMIT;
```

دستور BEGIN TRANSACTION یک نقطه شروع برای تراکنش تعیین می‌کند.
دو دستور UPDATE اجرا می‌شود. تا زمانی که دستور COMMIT اجرا نشود، تغییرات در پایگاه داده **تثبیت** نمی‌شود.
اگر هر دو دستور UPDATE با موفقیت اجرا شود، دستور COMMIT فراخوانی شده و تغییرات به صورت دائمی روی دیسک ذخیره می‌شود. که در این حالت مقدار A برابر 950 و مقدار B برابر 2050 خواهد بود.
اگر در حین اجرای تراکنش مشکلی پیش آید (مثلاً به دلیل نقض محدودیت‌های جامعیتی یا خطای سیستمی بخش اول تراکنش انجام شود اما بخش دوم انجام نشود که در این حالت مقدار A برابر 950 و مقدار B برابر همان 2000 خواهد بود)، آنگاه دستور ROLLBACK **به طور خودکار** فراخوانی شده و تمام تغییرات لغو می‌شود.
توجه: در شروع و پایان یک تراکنش سیستم باید سازگار باشد ولی در اثنای اجرای تراکنش ممکن است موقتاً نیاز به ناسازگاری باشد. برای مثال هنگام واریز پول از حساب A به B، پس از برداشت پول از حساب A، سیستم به **طور موقت ناسازگار** است و پس از واریز آن به حساب B دوباره سیستم سازگار می‌شود. لذا برنامه برداشت از حساب A یا واریز به حساب B به تنهایی تراکنش نیست.
توجه: دستور شروع تراکنش BEGIN TRANSACTION یا START TRANSACTION معمولاً اولین دستوری است که برای آغاز یک تراکنش صادر می‌شود. در هنگام اجرای این دستور، DBMS به طور خودکار یک شناسه منحصر به فرد Transaction ID یا TID به آن تراکنش اختصاص می‌دهد. DBMS با اختصاص یک شناسه منحصر به فرد به هر تراکنش، امکان مدیریت و کنترل مؤثر بر روی تراکنش‌ها را فراهم می‌کند.

۲- سازگاری (Consistency)

اگر یک تراکنش شروع به پیشروی کرد، در حین و میانه مسیر نباید **محدودیت‌های جامعیتی**

داخلی و خارجی (Integrity Constraints) را نقض کند. اما اگر نقض کرد، آنگاه هرگونه تغییری که تراکنش ایجاد کرده است، باید ابطال و لغو شود تا وضعیت پایگاه داده به آخرین وضعیت قبل از اجرای تراکنش بازگردد. بنابراین هر تراکنش به طور **منفرد** باید **محدودیت‌های جامعیتی** را رعایت کند تا در نهایت پایگاه داده را از حالت **سازگار قدیم** به حالت **سازگار جدید** منتقل کند، در غیر اینصورت اجرا و نتایج به دست آمده از آن قابل قبول نیست و همه چیز باید به قبل بازگردد.

توجه: دقت کنید که سازگاری فقط رعایت محدودیت‌های جامعیتی داخی و خارجی نیست، بلکه نقض خاصیت Atomicity، نقض خاصیت Isolation و نقض خاصیت Durabiltiy منجر به نقض خاصیت Consistency نیز می‌شود چون یک حالت سازگار را به حالت ناسازگار منتقل کرده است.

توجه: به وضعیت سازگار بانک اطلاعات، **وضعیت معتبر** (Valid State) نیز گفته می‌شود.

توجه: به تفاوت محدودیت‌های امنیتی و جامعیتی در ادامه توجه نمایید.

محدویت‌های امنیتی (Security constraints)

محدودیت‌های امنیتی از دسترسی **کاربران** غیرمجاز به پایگاه داده محافظت می‌کند. دقت کنید که در محدودیت‌های امنیتی تاکید روی **کاربران مجاز** است.

محدویت‌های جامعیتی (Integrity constraints)

محدودیت‌های جامعیتی تضمین می‌کنند که **عملیات** و تغییراتی که بر روی پایگاه داده انجام می‌شود، منجر به از دست رفتن سازگاری داده‌ها نشود. دقت کنید که در محدودیت‌های جامعیتی تاکید روی **عملیات مجاز** است. هدف از محدودیت‌های جامعیتی جلوگیری از ورود داده‌های نادرست به پایگاه داده است.

توجه: در نهایت ترکیب **محدودیت‌های امنیتی** و **محدودیت‌های جامعیتی** منجر به این می‌شود که **کاربران مجاز**، **عملیات مجاز** روی پایگاه داده انجام دهند.

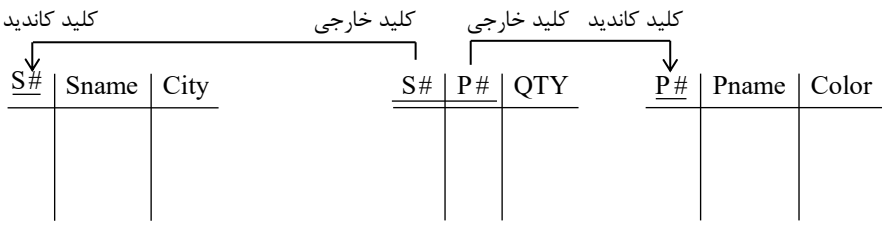
به طور کلی **محدودیت‌های جامعیتی** (Integrity Constraints) در سیستم‌های بانکی به دو طبقه‌ی **جامعیت داخلی** و **جامعیت خارجی (قواعد کسب و کار یا Business rules)** تقسیم می‌گردد. به حفظ قوانین مطرح شده از سوی مدل رابطه‌ای و DBMS در سطح پیاده‌سازی، **جامعیت داخلی** و به حفظ قوانین مطرح شده از سوی کسب‌وکار و اجرا شده توسط طراحان و برنامه‌نویسان بانک در سطح پیاده‌سازی، **جامعیت خارجی** گفته می‌شود. DBMS مسئول کنترل محدودیت‌های جامعیتی داخلی و خارجی در بانک است و هرگونه عاملی که باعث نقض محدودیت‌های جامعیتی داخلی و خارجی بانک گردد را رد می‌کند. قوانین داخلی بانک شامل **قانون جامعیت درون**

رابطه‌ای، قانون جامعیت موجودیت، قانون جامعیت ارجاعی و **قانون جامعیت دامنه‌ای** است، این موارد در فصل مدل رابطه‌ای بررسی خواهد شد. قوانین خارجی بانک هم شامل هر قانونی است که مربوط به قواعد کسب‌وکار است و توسط طراحان و برنامه‌نویسان بانک پیاده‌سازی می‌شود، مثل تعریف بازه 0 تا 20 برای نمرات یک دانشگاه، تعریف بازه 0 تا بینهایت برای حساب‌های بانکی به معنی عدم وجود موجودی منفی در حساب‌های بانکی و همچنین در برنامه‌های حساب بانکی جمع حساب مبدا (A) و مقصد (B) قبل از انجام تراکنش با بعد از تراکنش باید برابر باشد، در سیستم انبارداری موجودی کالا، نتیجه تفاضل تعداد کالای ورودی و خروجی است، در سیستم رزرواسیون هتل، تعداد اتاق‌های رزرو شده نباید از تعداد کل اتاق‌های هتل بیشتر باشد. چگونگی و انجام محاسبات این کار بر عهده برنامه‌نویس است که تراکنشی را کدگذاری می‌کند.

توجه: در سیستم‌های بانکی کنترل محدودیت‌های جامعیتی داخلی و خارجی به صورت خودکار توسط مکانیزم‌های موجود در DBMS انجام می‌گردد. اما طراحی و پیاده‌سازی محدودیت‌های جامعیتی داخلی و خارجی پایگاه داده برعهده برنامه‌نویسان است.

حال یکبار دیگر کد تعریف جدول SP و روابط بین جداول را در نظر بگیرید:

```
Create Table SP
        (
        S# char (5),
        P# char (5),
        QTY numeric (10),
        Primary key (S#, P#),
        Foreign key (S#) References S(S#)
          on delete cascade
          on update cascade,
        Foreign key (P#) References P(P#)
          on delete cascade
          on update cascade,
        Check (QTY>1 AND QTY<1000)
        )
```



جدول P *جدول SP* *جدول S*

کارکرد قطعه کد زیر از کد تعریف جدول SP فوق به صورت زیر است:

```
Foreing key (S#) References S(S#)
on delete cascade
on update casecade
```

کاربرد دستور Cascade : اگر سطرهای جدول اصلی (S) حذف (on delete cascade) یا بروزرسانی (on update casecade) شود، آنگاه سطرهای جدول فرعی (SP) که توسط کلید خارجی به آن ارجاع کرده است نیز حذف یا بروزرسانی می‌شود. برای مثال اگر مقدار S1 در جدول S به S11 بروزرسانی شود و این مقدار در جدول SP نیز موجود باشد، آنگاه مطابق رابطه Cascade مقدار S1 در جدول SP به مقدار S11 نیز بروزرسانی می‌شود. برای مثالی دیگر اگر سطر S1 از جدول S حذف شود و این مقدار در جدول SP نیز موجود باشد، آنگاه مطابق رابطه Cascade سطر S1 از جدول SP نیز حذف می‌شود.

بنابراین قطعه کد فوق، سبب می‌گردد تا به طور خودکار هرگونه تغییری در ستون #S در جدول S به ستون #S در جدول SP نیز اعمال گردد. بنابراین **جامعیت داخلی** از نوع **جامعیت ارجاعی** نقض نمی‌گردد.

یا به طور مشابه، کارکرد قطعه کد زیر از کد تعریف جدول SP فوق به صورت زیر است:

```
Foreing key (P#) References P(P#)
on delete cascade
on update cascade
```

این قطعه کد، سبب می‌گردد تا به طور خودکار هرگونه تغییری در ستون #P در جدول P به ستون #P در جدول SP نیز اعمال گردد. بنابراین **جامعیت داخلی** از نوع **جامعیت ارجاعی** نقض نمی‌گردد.

کارکرد قطعه کد زیر از کد تعریف جدول SP فوق به صورت زیر است:

```
Check (QTY>1 AND QTY<1000)
```

این قطعه کد، سبب می‌گردد تا به طور خودکار هرگونه مقداردهی در ستون QTY از جدول SP در بازه 1 تا 1000 باشد، بنابراین **جامعیت خارجی** نقض نمی‌گردد.

در ادامه به بررسی قوانین **جامعیت داخلی** می‌پردازیم:

۱- قانون جامعیت درون رابطه‌ای (Intra-Relational Integrity Rule)

مطابق قانون جامعیت درون رابطه‌ای، هر رابطه (جدول) باید به تنهایی درست باشد. یعنی صفات رابطه اتومیک باشد (چند مقداری و مرکب نباشد) و همچنین هر رابطه باید حتماً حداقل دارای

یک کلید کاندید باشد.

۲- قانون جامعیت موجودیت (Entity Integrity Rule)

مطابق قانون جامعیت موجودیت، هیچگاه نباید تمام یا بخشی از کلید کاندید مقدار NULL داشته باشد.

۳- قانون جامعیت دامنه‌ای (Domain Integrity Rule)

مطابق قانون جامعیت دامنه‌ای، صفات رابطه می‌بایست دارای نوع (دامنه) باشد و مقادیر صفات در محدوده همان دامنه باشد.

۴- قانون جامعیت ارجاعی (Referential Integrity Rule)

مطابق قانون جامعیت ارجاعی، هیچگاه نباید کلید خارجی، دچار ارجاع NULL گردد. برای این منظور باید تعریف ساختاری و محتوایی کلید خارجی برقرار باشد. همانطور که گفتیم کلید خارجی برای ارتباط میان جداول مورد استفاده قرار می‌گیرد. کلید خارجی در دو دیگاه ساختاری و محتوایی مورد بررسی قرار می‌گیرد:

دیدگاه ساختاری کلید خارجی

تعریف: اگر صفت(هایی) در یک جدول به عنوان کلید خارجی تعریف شود،

اول اینکه: این صفت(ها) در جدول خودش شرط خاصی ندارد یعنی الزاما خاصیت کلیدی ندارد.

و دوم اینکه: این صفت(ها) در همان جدول یا جدول دیگری، باید کلید کاندید (اصلی یا فرعی) باشد.

دیدگاه محتوایی کلید خارجی

به ازای هر مقدار موجود در یک کلید خارجی، باید دقیقاً یک مقدار متناظر در کلید کاندید متناظر آن وجود داشته باشد، در غیر این صورت می‌گوییم، کلید خارجی دارای ارجاع NULL است. به بیان دیگر، مقادیر کلید خارجی همواره باید زیرمجموعه مقادیر کلید کاندید باشد.

۳- انزوا یا جداسازی (Isolation)

اگر یک تراکنش شروع به پیشروی کرد، در حین مسیر فقط باید قادر باشد درصورت لزوم از **نتایج نهایی** تراکنش‌های هم‌روند دیگر استفاده کند و نباید قادر باشد از **نتایج میانی** تراکنش‌های هم‌روند دیگر استفاده کند. اما اگر استفاده کرد، آنگاه هرگونه تغییری که تراکنش ایجاد کرده است، باید ابطال و لغو شود تا وضعیت پایگاه داده به آخرین وضعیت قبل از اجرای تراکنش بازگردد. بنابراین هر تراکنش باید **انزوا** را رعایت کند تا در نهایت پایگاه داده را از حالت **سازگار قدیم** به

حالت **سازگار جدید** منتقل کند، در غیر اینصورت اجرا و نتایج به دست آمده از آن قابل قبول نیست و همه چیز باید به قبل بازگردد. راه انزوای تراکنش‌ها این است که هر تراکنش خودش عایق شود تا نتایج میانی آن به تراکنش‌های هم‌روند دیگر درز نکند. راه عایق‌سازی یک تراکنش این است که داده‌های اشتراکی به عنوان **ناحیه بحرانی** تلقی شود و برای آن **انحصار متقابل** برقرار شود، بدین معنی که در هر لحظه به هر داده اشتراکی فقط و فقط یک تراکنش دسترسی داشته باشد.

توجه: هم‌روندی در دسترسی به داده‌ها موجب بهبود کارایی و کاهش متوسط زمان پاسخ کل سیستم می‌گردد، این امر تسریع عملکرد برنامه‌ها را در پی دارد. بسیاری از سیستم‌ها اجازه می‌دهند که چندین کاربر به صورت هم‌روند به داده‌ها دسترسی یابند و تغییرات مورد نظر خود را بر روی آنها اعمال نمایند. در چنین محیط‌هایی تغییرات هم‌روند ایجاد شده بر روی داده ممکن است منجر به ایجاد ناسازگاری در داده‌ها گردد.

توجه: تراکنش‌های هم‌روند، الزاما از نظر زمانی نقطه شروع و پایان یکسان ندارند، اما در مقطعی از زمان به صورت موازی اجرا می‌شوند.

مثال : فرض کنید کاربر A و B به ترتیب در تهران و شیراز همزمان قصد برداشت وجه از حساب آقای 6037 از طریق برگ چک را دارند. بنابراین روال‌های زیر را خواهیم داشت، از آنجا که برداشت وجه از یک رکورد مشترک (عامل مشترک) صورت می‌گیرد، به تبع وقوع پدیده هم‌روندی برای هر دو تراکنش رخ می‌دهد. روال کار بدین صورت است که هر دو تراکنش مبلغ موجودی حساب که برابر مقدار 500 هزارتومان می‌باشد را خوانده و با توجه به مبالغ برگ چک A و B به ترتیب مبالغ 100 و 50 هزارتومان را از حساب کسر می‌کنند و بر حسب اینکه کدام تراکنش آخرین بروزرسانی را انجام دهد مبلغ مانده حساب 400 تا 450 هزارتومان ذخیره می‌گردد. که در هر دو صورت اطلاعات نادرستی ذخیره شده است. در حالی که مبلغ 350 هزار تومان باید جهت مبلغ مانده حساب ذخیره می‌شد!

شماره مشتری	شماره حساب	نام	نام‌خانوادگی	موجودی
6037	0313	Ali	Alavi	500

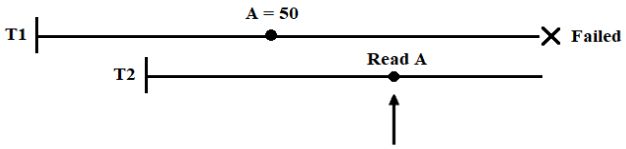
Read (A)	Read (B)
A ← 500	B ← 500
R=500-100	R=500-50
R ← 400	R ← 450
Write (R) ← 400	Write (R) ← 450

بدین‌ترتیب مقادیر نادرستی توسط برنامه‌ها ذخیره شده است که این نادرستی به دلیل تداخل

عملیات دو برنامه هم‌روند است. برای جلوگیری از ایجاد چنین نتایج نامطلوبی سیستم باید نـوعی نظارت بر عملکرد برنامه‌های هم‌روند داشته باشد.

برای برقراری خاصیت Isolation در مثال فوق، تراکنش B باید آپدیت A را پس از commit شدن A یا تراکنش A باید آپدیت B را پس از commit شدن B ببیند و استفاده کنـد. در اینصـورت بـه مقدار سازگار 350 می‌رسیم.

مثال : دو تراکنش T1 و T2 را به صورت هم‌روند در نظر بگیرید. فرض کنیـد خاصـیت Isolatin برای آنها برقرار شدن comm



هر **داده مشترک**

فرض کنید مقدار اولیه **داده مشترک** A برابر 100 است. و تراکنش T1 مقدار آن را بـه 50 آپـدیت می‌کند. سپس تراکنش T2 بدون commit شدن T1 مقدار داده مشترک A را استفاده می‌کند. فـرض کنید در ادامه تراکنش T1 به دلیل قطعی برق Failed شود، بنابراین چون T1 بـه طـور کامـل تمـام نشده است، مطابق خاصیت Atomicity تغییرات آن Rollback می‌شود، پس مقدار A بـه 100 بـاز می‌گردد. این درحالی است که T2 کارهای خود را با مقدار 50 انجام داده اسـت و نتـایج آن قابـل قبول نیست. در این مثال خاصیت Isolation نقض شده است، چون T2 توانسته از نتایج میانی T1 استفاده کند.

توجه: اگر در یک تراکنش **منفرد** اصل Atomicity برقرار باشد و همچین اصل Consistency نیز برقرار باشد، تضمینی نیست که همین تراکنش به طور هم‌روند در مقابل تراکنش دیگر اصل Isolation برای آن برقرار باشد. اگر به تنهایی عالی باشید تضمینی نیست که در همکاری با دیگران هم عالی باشید، تعامل با دیگران نیاز به مهارت‌های بیشتر دارد.

مثال: همانطور که قبلاً دیدیم، پایگاه داده در حین اجرای تراکنش انتقال وجـه از A بـه B، **بـه‌طور موقت ناسازگار** است، در این شرایط، مبلغ از حساب A برداشت شده، امـا بـه حسـاب B واریـز نشده است. حال اگر یک تراکنش دوم به‌طور هم‌روند در حال اجرا در این نقطـه میـانی، A و B را بخواند و A + B را محاسبه کند، مقدار ناسازگار را مشاهده خواهد کرد یعنی 2000 + 950 برابـر 2950 را در خروجی قرار خواهد داد. در حالی که پس از پایان تراکنش اول، جمع مقادیر A و B با مقدارهای 950 و 2050 برابر 3000 خواهد بود و این **ناسازگاری** در تراکنش دوم است.

توجه: یک راه برای جلوگیری از مشکل اجرای هم‌روندی تراکنش‌ها، اجرای سریالی تراکنش‌ها است، یعنی اجرای یکی پس از دیگری و به ترتیب. انجام سریالی تراکنش‌ها از نظر نتیجه، کار درستی است، اما از نظر بهره‌وری، کار نادرستی است. چون، اجرای هم‌روند تراکنش‌ها مزایای عملکردی قابل توجهی دارد که در صورت اجرای سریالی، سیستم از مزایای هم‌روندی محروم می‌شود:

بهبود توان عملیاتی و بهره‌وری (Improved throughput and resource utilization)

یک تراکنش شامل مراحل زیادی است. برخی شامل فعالیت ورودی وخروجی هستند؛ برخی دیگر شامل فعالیت CPU هستند. CPU و دیسک‌ها در یک سیستم کامپیوتری می‌توانند به‌طور موازی کار کنند. بنابراین، فعالیت ورودی و خروجی می‌تواند به‌طور موازی با پردازش در CPU انجام شود. همه اینها توان عملیاتی (تعداد تراکنش‌های اجرا شده در واحد زمان) را افزایش می‌دهند. متقابلاً، استفاده از CPU و دیسک نیز افزایش می‌یابد؛ به عبارت دیگر، CPU و دیسک زمان کمتری را بیکار می‌مانند.

کاهش زمان انتظار (Reduced waiting time)

ممکن است ترکیبی از تراکنش‌ها در یک سیستم در حال اجرا باشد، برخی کوتاه و برخی طولانی. اگر تراکنش‌ها به‌طور سریال اجرا شوند، یک تراکنش کوتاه ممکن است مجبور باشد تا تکمیل شدن یک تراکنش طولانی قبلی منتظر بماند (convoy effect یا اثر کاروان)، که می‌تواند منجر به تأخیر در اجرای یک تراکنش شود. اگر تراکنش‌ها در بخش‌های مختلف پایگاه داده کار می‌کنند، بهتر است اجازه داد به‌طور هم‌روند اجرا شوند و چرخه‌های CPU و دسترسی‌های دیسک را بین آن‌ها به اشتراک گذاشت. اجرای هم‌روند، انتظار در اجرای تراکنش‌ها را کاهش می‌دهد. علاوه بر این، متوسط زمان پاسخ را کاهش می‌دهد. در یک قاعده کلی هرچه قدر به اجرای تراکنش‌های با زمان اجرای کوتاه‌تر اولویت اجرا دهیم، میانگین زمان پاسخ تراکنش‌ها به کمترین مقدار ممکن می‌رسد، پیش‌بینی زمان اجرای فرآیندها امکان پذیر نیست، بنابراین در عمل با الگوریتم اشتراک زمانی نوبت چرخشی Round Robin خود به خود تراکنش‌های با زمان اجرای کمتر زودتر پردازنده می‌گیرند و از سیستم خارج می‌شود و این باعث بهبود توان عملیاتی (Improved throughput) می‌شود. بنابراین بهتر است راه‌حل‌هایی توسعه یابد که به اجرای هم‌روند چندین تراکنش در کنار هم و بدون اختلال اجازه می‌دهد.

کنترل هم‌روندی (control concurrency)

کنترل هم‌روندی جهت تضمین خاصیت Isolation **ادعا** دارد که هم اجازه می‌دهد تراکنش‌ها به

صورت هم‌روند اجرا شوند جهت بهره‌برداری از مزیت‌های هم‌روندی و هم اینکه طوری کنترل می‌کند که هیچ اختلالی به وجود نیاید. در واقع قاطعانه اجازه نمی‌دهد تراکنش‌های همکار از **نتایج میانی** هم استفاده کنند. در مباحث پایگاه داده، کنترل هم‌روندی تراکنش‌های هم‌کار به دو روش **زمان‌بندی ترتیبی** و **زمان‌بندی موازی** انجام می‌شود.

زمان‌بندی ترتیبی

مجدداً در نظر بگیرید T1 و T2 دو تراکنش هستند که وجه را از یک حساب به حساب دیگر منتقل می‌کنند. فرض کنید درست قبل از اجرای تراکنش T_i ، مقادیر حساب‌های A و B به ترتیب 1000 و 2000 واحد پولی است. تراکنش T1 ، 50 واحد پولی از حساب A به حساب B منتقل می‌کند:

```
T1 : read (A);
     A := A – 50;
     write (A);
     read (B);
     B := B + 50;
     write (B).
```

همچنین تراکنش T2 مقدار 10 درصد از حساب A را به B منتقل می‌کند.

```
T2 : read (A);
     temp := A * 0.1;
     A := A – temp;
     write (A);
     read (B);
     B := B + temp;
     write (B).
```

فرض کنید دو تراکنش به طور هم‌روند و به ترتیب T1 و سپس T2 اجرا شوند. این ترتیب اجرایی در شکل زیر نشان داده شده است:

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

Schedule 1—a serial schedule in which T_1 is followed by T_2.

در شکل فوق، ترتیب مراحل دستورات به صورت زمانی از بالا به پایین است، دستورات T1 در ستون سمت چپ و دستورات T2 در ستون سمت راست ظاهر می‌شوند.

مقادیر نهایی حساب‌های A و B، پس از اجرای تراکنش‌ها مطابق شکل فوق، به ترتیب 855 و 2145 هستند. بنابراین، کل مبلغ پول در حساب‌های A و B یعنی مجموع A + B برابر 3000 از اجرای هر دو تراکنش حفظ می‌شود. جمع دو حساب در پایگاه داده سازگار باید قبل و بعد از اجرای تراکنش، یکسان و برابر 3000 باشد که هست. و به معنی این است که نه مبلغی گم شده و نه مبلغی پیدا شده.

به طور مشابه، اگر تراکنش‌ها به طور هم‌روند و به ترتیب T2 و سپس T1 اجرا شوند، ترتیب اجرای مربوطه همان ترتیبی است که در شکل زیر نشان داده شده است.

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	

Schedule 2—a serial schedule in which T_2 is followed by T_1.

باز هم، همانطور که انتظار می‌رود، مجموع A + B حفظ می‌شود و مقادیر نهایی حساب‌های A و B به ترتیب 850 و 2150 هستند. یعنی مجموع A + B برابر 3000 از اجرای هر دو تراکنش حفظ می‌شود.

زمان‌بندی موازی

اگر دو تراکنش به طور هم‌روند در حال اجرا باشند، سیستم عامل ممکن است برای مدتی یک تراکنش را اجرا کند، سپس یک تعویض متن انجام دهد، تراکنش دوم را برای مدتی اجرا کند و سپس برای مدتی به تراکنش اول بازگردد و به همین ترتیب.

با تراکنش‌های چندگانه، زمان CPU بین تمام تراکنش‌ها به اشتراک گذاشته می‌شود. بنابراین چندین ترتیب اجرایی ممکن است، زیرا دستورالعمل‌های مختلف از هر دو تراکنش اکنون می‌توانند درهم آمیخته شوند. به طور کلی، پیش‌بینی اینکه چند دستورالعمل از یک تراکنش قبل از اینکه CPU به تراکنش دیگری تغییر کند، اجرا خواهد شد، امکان‌پذیر نیست. با بازگشت به مثال قبلی، فرض کنید دو تراکنش به طور هم‌روند اجرا شوند. یک زمان‌بندی احتمالی می‌تواند مطابق شکل زیر باشد:

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(B)
	$B := B + temp$
	write(B)
	commit

Schedule 3—a concurrent schedule equivalent to schedule 1.

پس از انجام این اجرا، به همان نتایجی می‌رسیم که در آن تراکنش‌ها به صورت سری و به ترتیب T1 و سپس T2 اجرا می‌شوند. مجموع A + B برابر 3000 در واقع حفظ می‌شود.

تمام اجرای‌های هم‌روند منجر به یک حالت صحیح نمی‌شوند. اما وظیفه DBMS اطمینان از این است که هر زمان‌بندی که اجرا می‌شود، پایگاه داده را در یک حالت سازگار قرار دهد. مؤلفه **کنترل هم‌روندی** DBMS توسط مکانیزم‌هایی که دارد این وظیفه را انجام می‌دهد.

توجه: خاصیت Isolation در تراکنش‌ها تضمین می‌کند که اجرای هم‌روند چند تراکنش، نتیجه‌ای معادل با حالتی داشته باشد که این تراکنش‌ها به‌صورت پشت‌سرهم و ترتیبی اجرا شده باشند. این کار توسط بخشی از DBMS به نام واحد **کنترل هم‌روندی** (Concurrency Control) انجام می‌شود.

۴- پایائی یا ماندگاری (Durabiltiy)

اگر یک تراکنش شروع به پیشروی کرد و به طور کامل تمام شد، باید در نهایت توسط دستور commit ذخیره و Save شود تا اطلاعات آن از حافظه اصلی به دیسک منتقل شود تا اثر آن ماندگار و دائمی شود. اما اگر commit نشد برای مثال برق قطع شد یا دیسک خراب یا پُر بود، آنگاه هرگونه تغییری که تراکنش ایجاد کرده است، باید ابطال و لغو شود تا وضعیت پایگاه داده به آخرین وضعیت قبل از اجرای تراکنش بازگردد. بنابراین هر تراکنش باید ماندگاری را رعایت کند تا در نهایت پایگاه داده را از حالت **سازگار قدیم** به حالت **سازگار جدید** منتقل کند، در غیر اینصورت اجرا و نتایج به دست آمده از آن قابل قبول نیست و همه چیز باید به قبل بازگردد. براساس این خاصیت، تراکنش‌هایی که به مرحله **تثبیت** (commit) برسند اثرشان ماندنی و دائمی است و هرگز از بین نمی‌رود. هنگامی‌که یک تراکنش دستور commit را اجرا می‌کند نتایج اجرای آن به نسخه‌ی اصلی پایگاه داده در دیسک منتقل می‌شود. بنابراین می‌توان گفت تا زمانی‌که یک تراکنش دستور commit را اجرا نکرده است یا به اصطلاح **تثبیت** نشده است، ویژگی ماندگاری در مورد آن تراکنش تضمین شده نیست، اما پس از اجرای دستور commit ماندگاری نتایج تراکنش توسط DBMS **تضمین** می‌شود.

خاصیت Durabiltiy تضمین می‌کند که پس از تکمیل موفقیت‌آمیز یک تراکنش، تمام به‌روزرسانی‌هایی که در پایگاه داده انجام داده است، حتی در صورت خرابی سیستم پس از تکمیل اجرای تراکنش، باقی می‌ماند. زیرا داده‌هایی که روی دیسک نوشته می‌شود هرگز از بین نمی‌رود. چون حفاظت در برابر از دست دادن داده‌ها روی دیسک توسط مکانیزم‌های پشتیبان‌گیری و **ذخیره‌سازی پایدار** امکان‌پذیر است. اطلاعات ذخیره‌شده در **ذخیره‌سازی پایدار** هرگز از بین نمی‌رود (البته ممکن است یک سیاهچاله زمین را در بر بگیرد و تمام داده‌ها را به‌طور کامل نابود کند، که این مورد خاص رو نادیده بگیرید چون بسیار بعید است.) برای پیاده‌سازی **ذخیره‌سازی پایدار**، اطلاعات در چندین رسانه ذخیره‌سازی **غیرفرار** (معمولاً دیسک) به صورت RAID تکثیر می‌شود. همچنین، برخی سیستم‌های پشتیبانی، پشتیبان باتری را نیز ارائه می‌دهند، بنابراین **حافظه اصلی** نیز می‌تواند از قطعی برق جان سالم به در ببرد.

توجه: یک تراکنش که اجرای خود را با موفقیت تکمیل می‌کند، گفته می‌شود که commit شده است. یک تراکنش commit شده که به‌روزرسانی‌هایی انجام داده است، پایگاه داده را به یک حالت سازگار جدید تبدیل می‌کند که باید حتی در صورت وجود خرابی سیستم نیز پایدار بماند.

توجه: پس از commit شدن یک تراکنش، نمی‌توانیم اثرات آن را با Undo کردن لغو کنیم. تنها راه لغو اثرات یک تراکنش commit شده، اجرای یک **تراکنش جبران‌کننده** است. برای مثال، اگر یک تراکنش 20 واحد پولی به یک حساب اضافه کرد، تراکنش جبران‌کننده 20 واحد پولی از حساب کم می‌کند.

توجه: دو خاصیت **Isolation** و **Durabiltiy** توسط واحدی از DBMS به نام واحد مدیریت بازیابی (Recovery management) کنترل می‌گردد.

خلاصه خواص ACID

به بیان ساده‌تر در خواص ACID **قدم اول** ماجرا همان Atomicity است. یعنی یک تراکنش، **کامل** اجرا شود، که اگر **ناقص** اجرا شود که هیچ اصلا کار به بررسی خواص بعدی ACID نمی‌رسد و همه چیز توسط اجرای دستور Rollback از Recovery management به نقطه شروع باز می‌گردد. در **قدم دوم** حال اگر یک تراکنش کامل اجرا شود در انتها توسط Consistency بررسی انجام می‌شود که آیا تمامی **محدودیت‌های جامعیتی داخلی و خارجی** (Integrity constraints) پس از انجام تراکنش رعایت شده است یا خیر؟ و در نهایت در **قدم سوم** خاصیت Durabiltiy اجرا می‌شود. مهمترین پرسشی که اینجا شکل می‌گیرد این است که پس جایگاه خاصیت Isolation کجاست؟ پاسخ این است که Isolation یک **قدم مکمل** در کنار قدم دوم Consistency است. اگر یک تراکنش **منفرد** بود که هیچ، اصلا نیاز به قدم مکمل Isolation نیست. اما اگر چند تراکنش هم‌روند وجود داشت. نیاز است در کنار خاصیت Consistency که وظیفه نظارت بر اجرای **معتبر** یک تراکنش را برعهده دارد، یک گارد محافظتی دیگر نیز به نام Isolation فعال شود که هر تراکنش را عایق کند. اصل Isolation را اینطور تصویرسازی کنید که انگار هر یک از تراکنش‌های هم‌روند به طور تک به تک در یک **محفظه ایزوله و عایق‌بندی شده** قرار می‌گیرند تا تراکنش‌ها به جای استفاده از نتایج میانی فقط از **نتایج پایانی** یکدیگر استفاده کنند. نتیجه اینکه Isolation کمک می‌کند تا در شرایط اجرای هم‌روند تراکنش‌ها، خاصیت Consistency حفظ شود.

دلایل ۶ گانه نقض خاصیت Consistency در سیستم‌های بانک اطلاعات

به طور کلی نقض خاصیت Consistency در سیستم‌های بانک اطلاعات به ۶ دلیل زیر است:

(۱) **خاصیت Atomicity نقض شده باشد:** در صورت اجرای ناقص تراکنش. برای مثال مبلغ 50 واحد پولی از حساب مبدا برداشت شده، اما به دلیل خرابی سیستم مبلغ 50 به حساب مقصد واریز

نشده است.

(۲) **خاصیت Isolation در تراکنش‌های هم‌روند نقض شده باشد:** استفاده از نتایج میانی

(۳) **محدودیت‌های جامعیتی داخلی نقض شده باشد:** برای مثال درج مقدار تکراری در ستون کلید اصلی نقض **جامعیت درون رابطه‌ای**، درج مقدار NULL در ستون کلید اصلی نقض **جامعیت موجودیت**، درج مقدار عددی در ستون با نوع کاراکتر نقض **جامعیت دامنه‌ای** و درج مقداری که زیر مجموعه کلید کاندید جدول اصلی نیست در ستون کلید خارجی جدول فرعی نقض **جامعیت ارجاعی**.

توجه: اگر پایگاه داده نرمال و بهینه نباشد و کلید خارجی تعریف نشود، تکرار اطلاعات منجر به **افزونگی طبیعی** و به تبع **ناسازگاری داده‌ای** می‌شود. اما اگر پایگاه داده نرمال و بهینه باشد و کلید خارجی تعریف شود، **جامعیت ارجاعی**، پایگاه داده را در وضعیت سازگار قرار می‌دهد.

(۴) **محدودیت‌های جامعیتی خارجی نقض شده باشد:** برای مثال درج مقدار منفی در حساب بانکی، یا درج نمره 22 خارج از بازه تعریف شده 0 تا 20، یا در سیستم حساب بانکی مقدار مربوط به برداشت و واریز وجه برابر نباشد برای مثال مبلغ 50 واحد پولی از حساب برداشت شده ولی مبلغ 70 واریز شده باشد. تاریخ برگشت پرواز قبل از تاریخ رفت باشد.

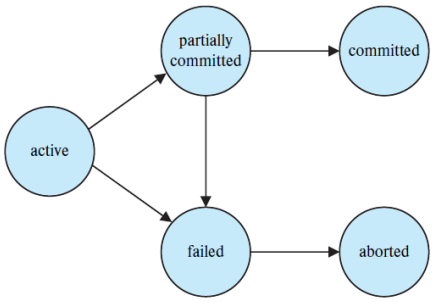
(۵) **خاصیت Durabiltiy در تثبیت تراکنش نقض شده باشد:** برای مثال قطعی برق یا خرابی دیسک.

(۶) **خطای انسانی در ورود داده:** برای مثال درج آدرس غلط برای مشتری که منجر به ورود داده ناصحیح به پایگاه داده شده است. DBMS امکان کنترل این نوع خطا را ندارد چون پدیده خطای انسانی است و نه فنی.

چرخه حیات تراکنش

همانطور که قبلاً اشاره کردیم، یک تراکنش ممکن است همیشه اجرای خود را با موفقیت تکمیل نکند. چنین تراکنشی **لغو شده** (aborted) نامیده می‌شود. اگر بخواهیم خاصیت Atomicity را تضمین کنیم، یک تراکنش لغو شده (aborted) نباید هیچ تأثیری بر حالت پایگاه داده داشته باشد. بنابراین، هر تغییری که تراکنش لغو شده در پایگاه داده ایجاد کرده است باید لغو شود. پس از لغو تغییرات ایجاد شده توسط یک تراکنش لغو شده، می‌گوییم تراکنش rolled back شده است که توسط سیستم مدیریت بازیابی (Recovery management) انجام می‌شود.

نمودار وضعیت یک تراکنش به صورت زیر است:



State diagram of a transaction.

۱- **فعال (Active) :** تراکنش در حال اجرا است و هنوز تکمیل نشده است. تراکنش در ابتدا در حالت فعال قرار دارد و تا زمانی که در حال اجرا است، در این حالت باقی می‌ماند.

۲- **نیمه تمام شده (Partially committed) :** پس از اجرای آخرین دستور، تراکنش در این حالت قرار می‌گیرد.

نیمه تمام شده: به این معنی است که تراکنش کار خود را تمام کرده و آخرین دستور آن اجرا شده است، اما هنوز commit نشده است. یعنی تغییرات ایجاد شده توسط تراکنش هنوز به طور کامل در **دیسک** ذخیره نشده است.

پس از اجرای آخرین دستور: یعنی زمانی که تراکنش تمام دستورات خود را اجرا کرده و به پایان رسیده است.

به عبارت ساده‌تر: یک تراکنش نیمه تمام شده، تراکنشی است که کار خود را انجام داده و نتایج نهایی داخل **حافظه اصلی** است، اما هنوز به طور کامل تضمین نشده است که تغییرات آن به طور **دائمی** در **دیسک** ذخیره شده باشد. مطابق نمودار این حالت واسطه‌ای بین حالت فعال **(Active)** و حالت تمام شده **(Committed)** است. یا واسطه‌ای بین حالت فعال **(Active)** و حالت ناکام **(Faild)** است.

۳- **ناکام (Failed) :** تراکنش به دلیل خطا یا شکست نتوانسته است به طور عادی اجرا شود.

این خطاها می‌توانند از انواع مختلفی باشد، مانند:

- خطاهای منطقی در خود تراکنش (مثلاً تقسیم بر صفر)
- خطاهای سخت‌افزاری در سیستم.
- خطاهای شبکه‌ای که ارتباط بین تراکنش و پایگاه داده را مختل می‌کند.

۴- **لغو شده (Aborted) :** تراکنش به دلیل خطا یا شکست لغو شده است و تغییرات آن نیز لغو شده‌اند. این حالت زمانی رخ می‌دهد که یک تراکنش ناکام (Failed) شده و سیستم تصمیم گرفته

است آن را لغو کند. لغو کردن تراکنش به معنای بازگرداندن پایگاه داده به حالت قبلی خود قبل از شروع تراکنش است. **یک تراکنش لغو شده نمی‌تواند به حالت دیگری منتقل شود.**

۵- **انجام شده (Committed) :** تراکنش با موفقیت اجرا شده است و تغییرات آن در دیسک **تثبیت** و **پایدار** شده‌است. این حالت زمانی رخ می‌دهد که یک تراکنش با موفقیت اجرا شده و تمام تغییراتی که ایجاد کرده است در پایگاه داده ذخیره شده‌اند. این به معنای آن است که تراکنش به طور کامل تکمیل شده و پایگاه داده به حالت سازگار جدیدی رسیده است. پس از انجام کامل یک تراکنش، تغییرات آن به طور دائمی در دیسک ذخیره و Save می‌شود و نمی‌توان آن‌ها را لغو کرد. **یک تراکنش commit شده نمی‌تواند به حالت دیگری منتقل شود.**

شرایط لازم برای Commit شدن یک تراکنش:

- تراکنش باید تمام دستورات خود را بدون خطا اجرا کرده باشد.
- تراکنش باید تمام تغییراتی که ایجاد کرده است در دیسک ذخیره کرده باشد.

زمانی که یک سری تغییرات بر روی پایگاه داده انجام می شود، این تغییرات به صورت موقت در **حافظه اصلی** ذخیره می‌شود. اجرای دستور commit باعث می‌شود که این تغییرات موقت به صورت دائمی در **دیسک** ذخیره و Save شوند و تراکنش به پایان برسد.

زبان کنترل تراکنش‌ها (TCL: Transaction Control Language)

زبان کنترل تراکنش‌ها Transaction control language یا به اختصار TCL زیر مجموعه زبان SQL است که تراکنش‌ها را در پایگاه داده مدیریت می‌کند. دستورات این زبان در واقع تغییرات اعمال شده توسط دستورات DML را اداره می‌کند.

SQL هیچ یک از دستورات DML را ماندگار نمی‌کند، مگر اینکه به طور کامل اجرا و Commit شود. اگر تراکنش به طور کامل انجام نشود، برای مثال حین آن برق برود یا سیستم دچار مشکل و از کار افتادگی شود، تمامی تغییرات انجام شده به حالت اولیه بازگشته و لغو می‌شود.

دستورات پرکاربرد TCL در SQL به صورت زیر است:

دستور COMMIT : تغییرات اعمال شده در طی تراکنش را بر روی دیسک دائمی می‌کند.

دستور Commit اغلب به طور خودکار توسط DBMS فراخوانی می‌شود، گاهی هم توسط برنامه‌نویسی داخل برنامه کاربردی فراخوانی می‌شود.

در SQL Server ، هر دستور SQL که به تنهایی اجرا می‌شود، به عنوان یک **تراکنش ضمنی** در نظر گرفته می‌شود. به محض اینکه این دستور با موفقیت اجرا شد، SQL Server به طور خودکار آن را commit می‌کند.

مثال: درج یک رکورد

```
INSERT INTO Customers (CustomerID, CustomerName)
VALUES (1, 'Ali');
```

زمانی که چندین دستور SQL را می‌خواهیم به عنوان یک **واحد کاری منطقی** در نظر بگیریم، باید یک **تراکنش صریح** ایجاد کنیم و با دستور COMMIT آن را پایان دهیم. اگر در حین اجرای این دستورات مشکلی پیش بیاید، دستور ROLLBACK تمام تغییرات را لغو می‌کند.

مثال: انتقال وجه بین دو حساب

```
BEGIN TRANSACTION;
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = 2;
COMMIT TRANSACTION;
```

دستور ROLLBACK : تمامی تغییرات یک تراکنش را به حالت اول برمی‌گرداند و لغو می‌کند.

دستور Rollback اغلب به طور خودکار توسط DBMS فراخوانی می‌شود، مانند زمانی که **محدودیت‌های جامعیتی داخلی و خارجی** نقض شود مثل درج مقدار تکراری در ستون کلید اصلی یا خطای سیستمی رخ دهد یا در نیمه تراکنش برق قطع شود. گاهی هم جهت کنترل خطای بهتر توسط برنامه کاربردی فراخوانی می‌شود.

مثال: در SQL Server

```
BEGIN TRANSACTION;
عملیات مختلفی که باید به صورت یک تراکنش انجام شود

IF ERROR_NUMBER() <> 0
BEGIN
   ROLLBACK TRANSACTION;
   PRINT  'تراکنش لغو شد.' ;
END
ELSE
BEGIN
   COMMIT TRANSACTION;
   PRINT  'تراکنش با موفقیت انجام شد.' ;
END
```

توجه: دریک عبارت ساده هرآنچه که Commit نشود در نهایت باید Rollback شود.

معماری سیستم‌های بانک اطلاعات

منظور از معماری بانک اطلاعات، پیکربندی اجزای سیستمی است که در آن حداقل یک بانک اطلاعات، یک سیستم مدیریت بانک اطلاعات (DBMS)، یک سیستم عامل، یک کامپیوتر با

دستگاه‌های جانبی و تعدادی کاربر (کاربران نهایی، مدیران و برنامه‌سازان) وجود دارد و خدماتی به کاربران نهایی ارائه می‌کند. نوع معماری به عواملی همچون سخت‌افزار، نرم‌افزار مدیریت بانک اطلاعات، موقعیت جغرافیایی کاربران، نیازهای کاربران ماهیت تراکنش‌ها، حجم داده‌ها و موقعیت مکانی داده‌ها و ارتباطات بین آنها بستگی دارد.

به طور کلی دو نوع معماری برای سیستم‌های بانکی وجود دارد :

۱- معماری متمرکز

۲- معماری نامتمرکز

- معماری مشتری ـ سرویس‌دهنده (خدمتگزار)
- معماری توزیع‌شده

معماری متمرکز

در این معماری یک پایگاه داده روی یک سیستم کامپیوتری و بدون ارتباط با سیستم کامپیوتری دیگر ایجاد می‌شود. و برای کاربردهای کوچک و با امکانات محدود است. سخت‌افزار این سیستم می‌تواند در حد یک کامپیوتر شخصی و یا بالاتر باشد. (مانند برنامه دفترچه تلفن شخصی)

معماری نامتمرکز

معماری مشتری - سرویس‌دهنده

هر معماری که در آن قسمتی از پردازش را یک برنامه، سیستم یا ماشین انجام دهد و انجام قسمت دیگری از پردازش را از برنامه، سیستم یا ماشین دیگری بخواهد، معماری مشتری - سرویس‌دهنده نامیده می‌شود.

در واقع وظایفی که باید سیستم انجام دهد به دو رده تقسیم می‌شوند: رده‌ای که انجام آنها بر عهده سرویس دهنده است و رده‌ای که توسط مشتری انجام می‌شود. بدین ترتیب یک معماری دو سطحی داریم که مشکل توزیع را تسهیل می‌کند. با این تعریف، در این معماری یک ماشین (یا سیستم یا برنامه) سرویسی را به ماشین (یا سیستم یا برنامه) دیگر ارائه می‌کند. بنابراین به این ماشین (یا سیستم یا برنامه) سرویس دهنده می‌گویند. خدماتی که سرویس‌دهنده ارائه می‌کند متنوع است، برای مثال خدمات چاپ، خدمات فایل، خدمات پست الکترونیک، خدمات اینترنت، خدمات بانک اطلاعات و بسته به نوع خدمت، سرویس‌دهنده را با نامی مناسب می‌نامند. برای نمونه سرویس‌دهنده چاپ و سرویس‌دهنده فایل، بنابراین منظور از معماری مشتری - سرویس‌دهنده آن نوع از معماری است که در آن مسئولیت‌ها به طور منطقی تقسیم شده است.

انواع معماری مشتری ـ سرویس‌دهنده از نظر تعداد مشتری و سرویس‌دهنده:

۱ـ چند مشتری ـ یک سرویس‌دهنده

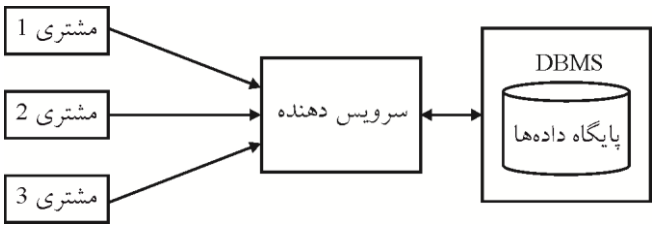
۲ـ یک مشتری ـ چند سرویس‌دهنده

۳ـ چند مشتری ـچند سرویس‌دهنده

انواع معماری مشتری ـ سرویس‌دهنده از نظر پیکربندی سخت‌افزاری

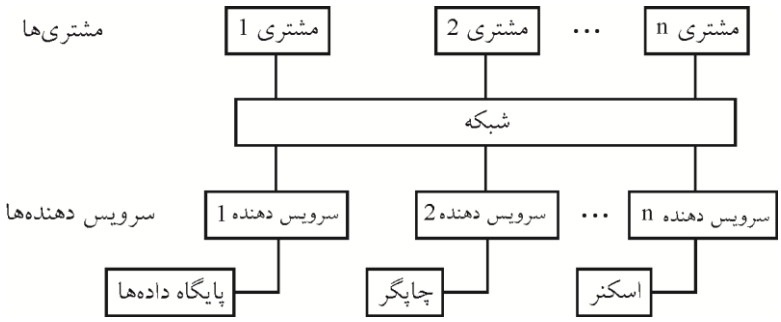
معماری حول سرویس‌دهنده

در این طرح، ماشین سرویس‌دهنده یک ابر کامپیوتر است و پایگاه داده‌ها روی همین کامپیوتر ایجاد و مدیریت می‌شود و تعدادی مشتری، از طریق شبکه از خدمات آن استفاده می‌کنند.



معماری حول شبکه

در این طرح، تعدادی کامپیوتر شخصی به عنوان سرویس‌دهنده و تعدادی دیگر به عنوان مشتری از طریق شبکه بهم مرتبط هستند.



معماری توزیع شده

اصطلاح پردازش توزیعی بدین مفهوم است که ماشین‌های مجزا می‌توانند در یک شبکه ارتباطی به یکدیگر متصل شوند. به گونه‌ای که یک عملکرد پردازش داده منفرد می‌تواند بر روی چندین ماشین شبکه پراکنده شود. در سیستم پایگاه‌های توزیع شده، پایگاه داده‌ها بر روی چند کامپیوتر ذخیره می‌شود. کامپیوترها در یک سیستم توزیع شده از طریق رسانه‌های مختلف ارتباطی نظیر شبکه‌های با سرعت بالا یا خطوط تلفن به یکدیگر مرتبط هستند.

می‌توان گفت که در این معماری تعدادی پایگاه داده‌های ذخیره شده روی کامپیوترهای مختلف داریم که از نظر کاربران، پایگاه داده واحدی هستند.