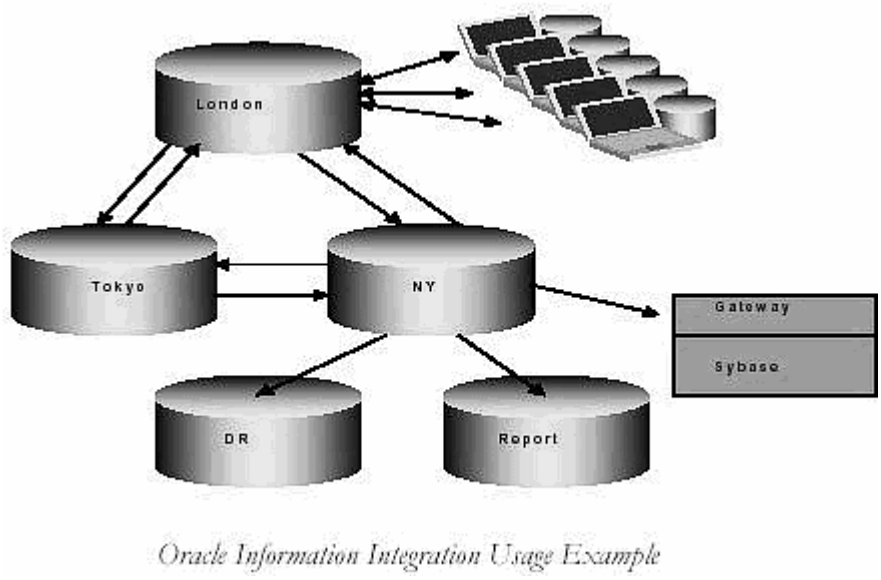
اصول و طراحی
پایگاه داده‌ها



Oracle Information Integration Usage Example

مدرس: محمد سوری

ترم اول ۸۸-۱۳۸۷

فهرست مطالب

فصل اول

«تاریخچه و مفاهیم پایگاه داده‌ها»

در این فصل، تاریخچه و سیر تکاملی پایگاه داده‌ها را بررسی کرده، مدل‌های پایگاه داده را مقایسه کرده و اجزا و روند کار سیستم‌های پایگاه داده را مورد بحث قرار می دهیم.

۱-۱ مفاهیم مربوط به پایگاه داده‌ها

قبل از تعریف پایگاه داده، لازم است تفاوت میان داده[۱] و اطلاع[۲] را درک کنید. داده، واقعیتی است خام که نمی‌تواند هیچ تأثیری در تصمیم‌گیری‌های ما داشته باشد. به عنوان مثال، موارد زیر سه داده در یک سیستم فروشگاه می‌باشند:

شماره فاکتور = ۱۰۰۰۱

تاریخ فاکتور = ۱۳۸۵/۱۰/۱۲

مبلغ کل فاکتور = ۵۰۰۰۰

اطلاع از پردازش داده ها به دست می‌آید. به عنوان مثال با انجام برخی محاسبات روی مبالغ کل فاکتورهای هر بخش فروشگاه، می‌توان میزان فروش کل هر یک از بخش ها را بدست آورده و از آن به عنوان معیاری برای مقایسه بازدهی بخش‌های مختلف استفاده کرد.

مدیریت داده[۳]

مبنای تصمیم گیری در سازمان‌ها، اطلاعات می‌باشند. از طرفی اطلاعات مفید و جامع، از داده های خوب به دست می آید. برای تولید، ذخیره و بازیابی صحیح داده ها، مکانیزم موثری مورد نیاز است که آن را مدیریت داده‌ها می‌نامیم.

پایگاه داده‌ها[۴]

یک ساختار کامپیوتری یک پارچه[۵]، شامل داده های مورد نیاز کاربران نهائی و فرا داده ها که توسط کلیه کاربران یک محیط عملیاتی (مثل دانشگاه، فروشگاه و ...) به صورت اشتراکی مورد استفاده قرار می گیرد.

تعریف دیگری از پایگاه داده‌ها[۶]

مجموعه ای است از داده های ذخیره شده و پایا، به صورت مجتمع (یک‌پارچه) (نه لزوماً به طور فیزیکی بلکه حداقل به طور منطقی)، به هم مرتبط، با کمترین افزونگی، تحت مدیریت یک سیستم

---

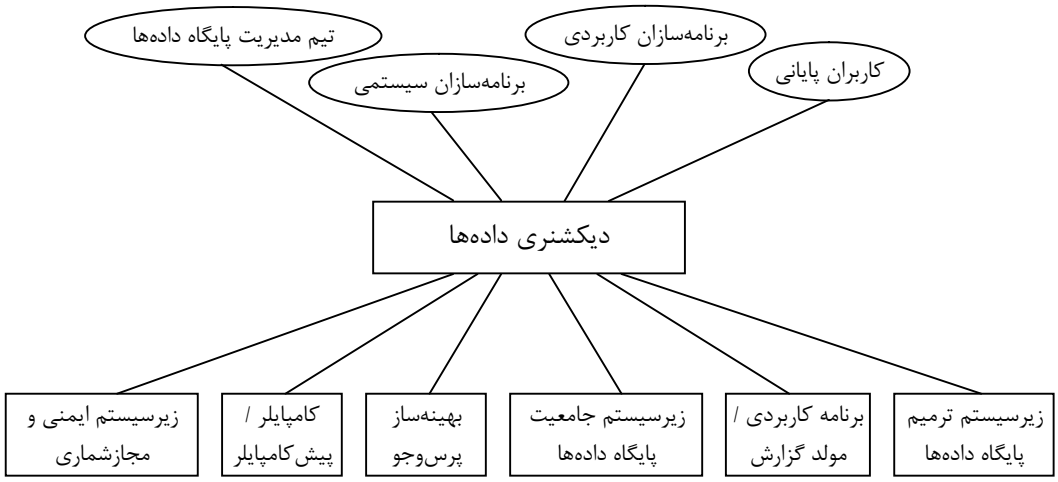
[1] Data
[2] Information
[3] Data Management
[4] Database
[5] Integrated
[6] تعریفی جامع از کتاب مفاهیم بنیادی پایگاه داده‌ها، تألیف سید محمد تقی روحانی رانکوهی، انتشارات جلوه

کنترل متمرکز، مورد استفاده یک یا چند کاربر، از یک (یا بیش از یک) سیستم کاربردی، بطور همزمان[1] و اشتراکی[2].

فرا داده‌ها[3]: کاتالوگ سیستم و دیکشنری داده‌ها

کاتالوگ سیستم، حاوی داده‌هایی است در مورد داده‌های ذخیره شده در پایگاه داده‌های کاربر و این داده‌ها، به فراداده‌ها موسومند. به کاتالوگ سیستم، گاه دیکشنری داده‌ها هم گفته می‌شود. اما در واقع دیکشنری داده‌ها، حاوی اطلاعات بیشتری است. فراداده‌ها معمولاً از دید کاربر سطح خارجی نهان‌اند اما مسئول سیستم و یا کاربر مجاز، می‌تواند تا حدی از محتوای کاتالوگ آگاه شود.

دیکشنری داده‌ها معمولاً جزیی از خود سیستم است و به دو صورت فعال[4] و غیرفعال[5] تولید می‌شود. دیکشنری فعال آن است که هر بار که پایگاه داده‌ها مورد دستیابی قرار می‌گیرد، واحدهایی از سیستم، بسته به نوع درخواست کاربر، آن را وارسی می‌کنند و بر اساس اطلاعات موجود در آن، درخواست کاربر، نهایتاً انجام می‌شود. اما دیکشنری غیرفعال فقط توسط طراح و کاربران مجاز پایگاه داده‌ها و نیز تیم مدیریت پایگاه داده‌ها استفاده می‌شود تا اطلاعاتی از آن به دست آورند و خود سیستم از آن استفاده نمی‌کند. در شکل ۱، استفاده کنندگان از دیکشنری داده‌ها نشان داده شده است.



شکل۱: دیکشنری داده‌ها و استفاده‌کنندگان آن

ساختار و محتوای کاتالوگ و دیکشنری داده‌ها در سیستم‌های مختلف یکسان نیست اما به طور کلی، اطلاعات زیر در آن نگهداری می‌شود:

- شِماهای خارجی[6]
- شِمای ادراکی[7]
- شِمای داخلی[8]

---

[1] Concurrent
[2] Shared
[3] Metadata
[4] Active Dictionary
[5] Passive Dictionary
[6] External Schema
[7] Conceptual Schema
[8] Internal Schema

- رویه‌های مربوط به تبدیلات بین سه سطح معماری (قسمت ۱-۷ را ببینید)
- شرح سازمان فیزیکی داده‌های ذخیره شده (فایل‌ها، رسانه‌ها، ...)
- مشخصات کاربران و حقوق دستیابی[1] آن‌ها به داده‌های ذخیره شده
- مشخصات برنامه‌های کاربردی تولید شده و ارتباط آن‌ها با درخواست‌های کاربران
- مشخصات پایانه‌های متصل به سیستم
- ارتباط بین برنامه‌های کاربردی و داده‌های ذخیره شده
- قواعد مربوط به کنترل صحت و دقت داده‌های ذخیره شده در پایگاه داده‌ها (موسوم به قواعد جامعیت)
- ضوابط کنترل ایمنی داده‌ها
- مشخصات پیکربندی سخت‌افزاری سیستم و رسانه‌های ذخیره‌سازی
- اطلاعات متنوع آماری در مورد پایگاه داده‌ها و کاربران
- توابع تعریف شده توسط کاربران

و برخی اطلاعات دیگر

نکته: از دیکشنری داده‌ها برای مستندسازی فرایند طراحی پایگاه داده‌ها نیز استفاده می‌شود به این ترتیب که مستندات نتایج هر مرحله از طراحی و تصمیمات اتخاذ شده در هر مرحله، در دیکشنری داده‌ها ذخیره می‌شود. این کار، به خودکارسازی فرایند طراحی کمک می‌کند.

نکته: فراداده را می‌توان «داده در مورد داده» نامید.

تفاوت میان داده و فرا داده

در یک محیط عملیاتی مثل دانشگاه، داده‌هایی برای کاربران نهائی اهمیت دارند که یکی از موجودیت‌های محیط عملیاتی (مانند دانشجو، استاد، وام و ...) و یا یکی از روابط محیط عملیاتی (مانند ثبت‌نام یک دانشجو در یک درس، اخذ وام توسط دانشجو و ...) را تشریح می‌کنند. علاوه بر این نوع داده‌ها، دسته دیگری از داده‌ها وجود دارند که مربوط به هیچ موجودیت یا رابطه عملیاتی نیستند. این داده ها خود ساختار داده های اصلی را توصیف می‌کنند که آنها را فراداده می نامیم. مثلاً فراداده ای که ساختار جدول دانشجو را تشریح می کند شامل نام، نوع و طول فیلدهای جدول دانشجو است.

۱-۲ تاریخچه پایگاه داده‌ها

۱-۲-۱ سیستم فایل (مشی فایلینگ)

قبل از اختراع کامپیوتر سازمانها از سیستم‌های دستی برای نگهداری اطلاعات خود استفاده می‌کردند که هر دسته از اطلاعات در پوشه مجزایی نگه داری می شد. مثلاً در یک دانشگاه، اطلاعات مربوط به دانشجویان در یک پوشه، اطلاعات مربوط به اساتید در یک پوشه، اطلاعات مربوط به ثبت نام در یک پوشه و ... طبقه بندی می شد.

[1] Access Right

با اختراع کامپیوتر، سیستم‌های فایل جایگزین سیستم‌های دستی شدند. به این ترتیب در هر سازمان، اطلاعات هر پوشه در یک فایل کامپیوتری ذخیره شده و برنامه‌نویسان با استفاده از زبانهای نسل سوم مانند Cobol و Basic، برنامه‌هایی برای دستکاری اطلاعات موجود در فایل‌ها و نیز تهیه گزارشات می نوشتند.
مهمترین مشخصه و در واقع عیب سیستم‌های فایل، عدم یکپارچگی آنها بود چرا که هر قسمت از سازمان، سیستم فایل مجزایی داشت. مثلاً بخش آموزش و بخش مالی در یک دانشگاه، فایلهای مجزایی داشتند.

۱-۲-۲ مفاهیم مربوط به سیستم‌های فایل

۱- *داده*[1]: عبارت است از حقیقتی خام که هیچ مفهوم خاصی ندارد. مثلاً وقتی از عدد ۵ صحبت می کنیم، مشخص نمی‌شود این ۵، تعداد فرزندان یک کارمند است یا وزن یک جسم یا ...
نکته: مفهوم داده در سیستم‌های فایل با مفهوم داده در حالت کلی تفاوت دارد.

۲- *فیلد*[2]: گروهی از کاراکترها که خصوصیتی از یک موجودیت را توصیف می کند مثل name، age و weight. هنگامی که داده ای در داخل یک فیلد ریخته می‌شود، مفهوم و معنا پیدا می کند. مثلاً اگر عدد ۵ را داخل فیلد weight بریزیم، وزن یک موجودیت را مشخص می کند.

۳- *رکورد*[3]: مجموعه ای از فیلد ها که مربوط به یک موجودیت خاص هستند. مثلاً رکورد یک دانشجو می‌تواند به شکل زیر باشد:

StNo	Name	Course
۷۸۰۱	آرش راد	پایگاه داده

۴- *فایل*[4]: مجموعه ای از رکوردها که مربوط به یک نوع موجودیت هستند مثلاً فایل دانشجویان یک دانشگاه می تواند به شکل زیر باشد:

StNo	Name	Course
۷۸۰۱	علی راد	پایگاه داده
۷۸۰۹	علی تقوی	ریاضی
۷۹۰۱	مینا رسولی	ادبیات فارسی

۵- *سیستم فایل*[5]: مجموعه ای از فایل ها و برنامه‌های لازم برای کار با آنها. مثلاً سیستم فایل یک دانشگاه شامل موارد زیر می‌باشد:
الف) فایل دانشجویان و برنامه‌های مدیریت این فایل (برنامه‌های برای درج، حذف و یا تغیر اطلاعات دانشجویان) و برنامه‌های مورد استفاده برای تهیه گزارشات (لیستی از دانشجویان ممتاز، مشروط و اخراجی)
ب) فایل دروس و برنامه‌های مربوط به مدیریت این فایل و نیز تهیه گزارشات
ج) فایل اساتید و برنامه‌های مدیریت و گزارش‌گیری مربوط به این فایل

۱-۲-۳ معایب سیستم‌های فایل

۱- نیاز به برنامه‌نویسی زیاد و پیچیده.

---

[1] Data
[2] Field
[3] Record
[4] File
[5] File System

۲- *وابستگی داده ای*[1]: وابستگی داده ای به «لزوم تغییر کد برنامه‌های مربوط به یک فایل پس از اعمال یک تغییر در مشخصات یکی از فیلدهای فایل (تغییر نوع یک فیلد از Integer به String و یا تغییر طول یک فیلد)» اطلاق می‌شود.

۳- *وابستگی ساختاری*[2]: وابستگی ساختاری به «لزوم تغییر کد برنامه‌های مربوط به یک فایل پس از اعمال هر تغییر در ساختار فایل (اضافه کردن یا حذف فیلد)» اطلاق می‌شود.

۴- *افزونگی داده ها*[3]: افزونگی داده عبارت است از تکرار یک قلم داده[4] در چندین جای مختلف.

۵- *عدم وجود امکانات لازم برای تأمین امنیت داده ها*[5]: سیستم‌های فایل فاقد امکانات لازم برای تعیین حدود اختیارات هر کاربر بودند. به عبارتی دیگر، در این سیستم‌ها هر کاربر با هر سمت و اختیاراتی، می توانست به کلیه‌ی داده های ذخیره شده در فایل ها دسترسی داشته باشد.

۱-۲-۴ مشکلات حاصل از افزونگی در سیستم فایل

۱- *هدر رفتن فضا و نیروی کار*

۲- *ناسازگاری داده‌ها*[6]: ناسازگاری داده‌ها به «وجود ۲ یا چند مقدار متفاوت برای یک قلم داده» اطلاق می‌شود. وجود افزونگی ذاتاً عامل مهمی برای بروز ناسازگاری داده ها می‌باشد.

۳- *بروز انواع ناهنجاری*: افزونگی داده، باعث بروز سه نوع ناهنجاری می‌شود. برای درک انواع ناهنجاری، سیستم یک دانشگاه را در نظر بگیرید. قسمت امور مالی این دانشگاه، فایلی به نام Tutor دارد که حاوی اطلاعات اساتید دانشگاه است:

Tuor	Tname	Ttel
۱۰۰	آرش راد	۶۲۴۴۴
۱۰۱	مینا رضوی	۶۱۵۲۵
۱۰۲	عسل رسولی	۲۱۹۳۹

از طرف دیگر، قسمت آموزش از فایلی به نام Project برای درج اطلاعات پروژه های فارغ التحصیلی دانشجویان استفاده می کند و چون به فایل های موجود در قسمت امور مالی دسترسی ندارد، مجبور است اطلاعات استاد راهنمای پروژه را نیز در این فایل نگهداری کند.

St#	Project-Title	Tutor	Tname	Ttel
۷۸۰۱	طراحی و پیاده‌سازی یک سایت خرید و فروش اینترنتی	۱۰۰	آرش راد	۶۲۴۴۴
۷۹۰۲	طراحی و پیاده‌سازی یک سیستم ثبت‌نام دانشگاه	۱۰۲	عسل رسولی	۲۱۹۳۹
۸۰۰۱	بهینه سازی پرس و جوها در پایگاه داده	۱۰۰	آرش راد	۶۲۴۴۴

[1] Data Dependence
[2] Structural Dependence
[3] Data Redundancy
[4] Item
[5] Data Security
[6] Data Inconsistency

سازماندهی فایل ها به صورت فوق می تواند باعث بروز ناهنجاری های زیر شود:

الف) ناهنجاری اصلاح[1]:

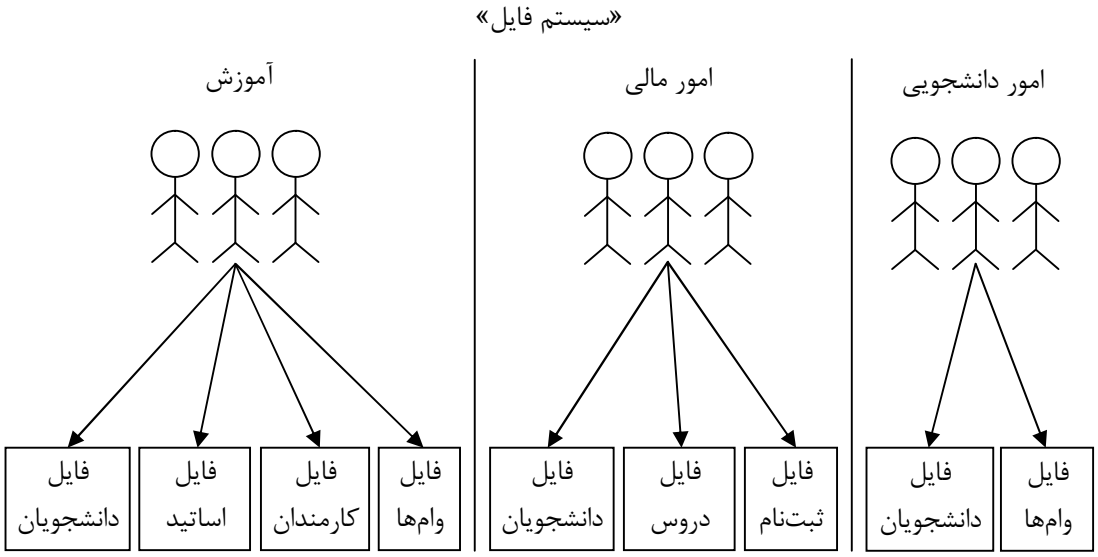
فرض کنید آرش راد شماره تلفن خود را به ۶۲۷۷۷ تغییر دهد. در این صورت، برای جلوگیری از بروز ناسازگاری لازم است در هر دو قسمت امور مالی و آموزش، این تغییر اعمال شود. بدیهی است قسمت آموزش برای اعمال این تغییر مجبور است کلیه‌ی رکوردهای مربوط به پروژه های اخذ شده با آقای آرش راد را تغییر دهد. این نوع اصلاح، به اصلاح منتشر شونده[2] معروف است.

ب) ناهنجاری درج[3]:

برای درج یک پروژه جدید در فایل Project لازم است اطلاعات استاد راهنمای مربوطه نیز درج شود حتی اگر قبلاً اطلاعات وی در پروژه های دیگر درج شده باشد. این مسأله علاوه بر هدر دادن فضا و نیروی کار، خطر بروز ناسازگاری را نیز افزایش می دهد چرا که در ورود دوباره اطلاعات استاد توسط اپراتور، امکان خطا وجود دارد.

ج) ناهنجاری حذف[4]:

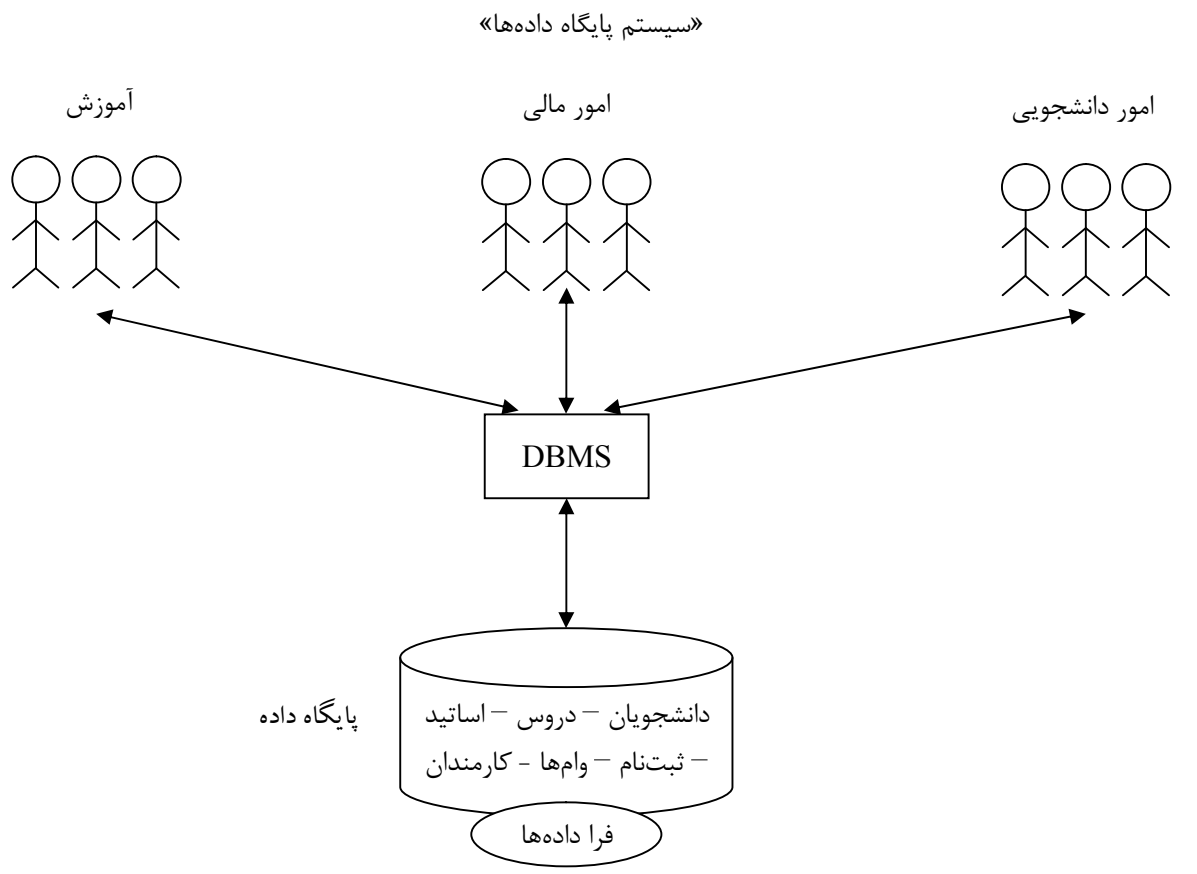
فرض کنید قسمت امور مالی، اطلاعات مربوط به آقای آرش راد را از فایل Tutor حذف کند. در این صورت در فایل Project برخی از دانشجویان، استاد راهنمایی خواهند داشت که دیگر وجود ندارد. پس برای جلوگیری از بروز ناسازگاری داده ها، قسمت آموزش باید کلیه‌ی پروژه هایی را که با این استاد اخذ شده اند حذف کرده یا اطلاعات فیلدهای مربوط به استاد را در این رکوردها خالی گذاشته و یا با اطلاعات استاد دیگری، جایگزین کند. از طرف دیگر، فرض کنید دانشجوی ۷۹۰۲ تنها دانشجویی باشد که با عسل رسولی پروژه گرفته است. در این صورت با حذف این دانشجو، قسمت آموزش، اطلاعات مربوط به عسل رسولی را نیز از دست می دهد. در واقع، «با حذف یک قلم اطلاعاتی، به طور ناخواسته اطلاعات دیگری نیز حذف می‌شود».



[1] Modification Anomaly
[2] Propagating Modification
[3] Insertion Anomaly
[4] Deletion Anomaly

۱-۳ سیستم‌های پایگاه داده‌ها[1] (مشی پایگاهی)

در این سیستم‌ها بر خلاف سیستم‌های فایل، کلیه داده ها به صورت یکپارچه و تنها در یک محل ذخیره می‌شوند و کلیه کاربران می‌توانند به صورت اشتراکی و همزمان از این داده ها استفاده کنند. در سیستم‌های پایگاه داده، بر خلاف سیستم‌های فایل هیچ یک از کاربران به صورت مستقیم به داده ها دسترسی ندارند بلکه درخواست های خود را در قالب یک دستور سطح بالا به یک نرم‌افزار از پیش ساخته شده به نام نرم‌افزار مدیریت پایگاه داده یا DBMS[2] تحویل می دهند. DBMS نقش واسط و مترجم میان کاربران (یا برنامه‌های کاربردی[3] آنها) و پایگاه داده (شامل داده های مورد نیاز کاربران و فراداده ها) را بازی می‌کند و بسیاری از وظایفی را که در سیستم‌های فایل بر عهده کاربران بود، خود به تنهایی به دوش می‌کشد. مثلاً با اجرای دستور Select Name From Student Where Avg>17 می‌توان اسامی تمام دانشجویانی که معدل آنها بیشتر از ۱۷ بوده را از DBMS درخواست کرد.



۱-۴ اجزای یک سیستم پایگاه داده‌ها

محیط پایگاه داده‌ها، مثل هر محیط ذخیره و بازیابی اطلاعات، از چهار عنصر تشکیل شده است:

◆ سخت‌افزار ◆ نرم‌افزار ◆ کاربر[4] ◆ داده

[1] Database Systems
[2] Database Management System
[3] Applications
[4] User

۱-۴-۱ سخت‌افزار

در محیط پایگاه داده‌ها هم مثل هر محیط ذخیره‌سازی اطلاعات، سه دسته سخت‌افزار وجود دارد که عبارتند از *سخت‌افزار ذخیره‌سازی داده‌ها* (مثل دیسک مغناطیسی)، *سخت‌افزار پردازش‌گر* (منظور، خود کامپیوتر است) و *سخت‌افزار هم‌رسانش*[1] *(ارتباط)*. سخت‌افزار هم‌رسانش عبارت است از هر سخت‌افزار ارتباطی بین کامپیوتر و دستگاه‌های جنبی و نیز بین کامپیوترها. این سخت‌افزارها به دو رده کلی تقسیم می‌شوند که عبارتند از سخت‌افزارهای محلی و سخت‌افزارهای شبکه‌ای. سخت‌افزارهای محلی برای ایجاد ارتباط بین کامپیوتر و دستگاه‌های جنبی آن در یک مانه (سایت[2]) به کار می‌روند و سخت‌افزارهای شبکه‌ای در ایجاد سیستم پایگاه داده‌ها با معماری نامتمرکز[3] به کار می‌روند.

۱-۴-۲ نرم‌افزار

الف) سیستم عامل

ب) DBMS: همانگونه که عنوان شد، DBMS نرم‌افزاری از پیش تهیه شده و پیچیده است که برنامه‌نویسان و طراحان هیچ‌گونه دخل و تصرفی در عملکرد آن ندارد و فقط نسخه‌های مختلف آن را خریداری کرده و نصب می‌کنند. از جمله معروف‌ترین DBMSهای کنونی، Microsoft SQL Server، Oracle، Informix، DB2 و MySQL می‌باشد.

ج) برنامه‌های کاربردی: برنامه‌هایی که توسط برنامه‌نویسان و مخصوص یک محیط عملیاتی خاص نوشته می‌شوند مثل برنامه‌های حسابداری، حقوق و دستمزد و ... که این‌گونه برنامه‌ها از طریق DBMS با پایگاه داده‌ها ارتباط برقرار می‌کنند.

د) رویه‌های ذخیره شده[4] (روال‌ها): شامل دستوراتی هستند که برای اجرای قوانین حاکم بر سیستم نوشته می‌شوند. مثلاً در یک سیستم فروشگاه به محض صدور فاکتور برای مشتری باید موجودی انبار به روز رسانده شود که برای انجام این کار، معمولاً از یک روال استفاده می‌شود.

نکته: برخی از نویسندگان، روال‌ها را به عنوان بخش مجزایی از اجزای یک سیستم پایگاه داده‌ها در نظر می‌گیرند.

۱-۴-۳ کاربر

الف) مدیران سیستم[5]: این کاربران بر عملکرد کلی سیستم پایگاه داده نظارت می‌کنند.

ب) مدیر پایگاه داده‌ها یا DBA[6]: این فرد، خط مشی و سیاست‌های کلی استفاده و کار با پایگاه‌داده را مشخص می‌کند مثلاً تعین می‌کند از چه DBMSی استفاده شود و یا چگونه نسخه‌های جدید DBMS نصب شوند و یا برای بهینه سازی کارآیی پایگاه داده، از چه استانداردها و مکانیزم‌هایی استفاده شود. این مدیر معمولاً همراه با یک تیم تخصصی کار می‌کند که به آن، تیم مدیریت پایگاه داده‌ها می‌گویند. هر یک از اعضا این تیم، مسئولیت خاصی دارد و در حیطه اختیارات و وظایفش، می‌تواند سرپرست یک تیم اجرایی باشد.

---

[1] Communication
[2] Site
[3] Decentralized Architecture
[4] Stored Procedures
[5] System Administrators
[6] Database Administrator

ج) طراحان پایگاه داده[1]: این دسته از کاربران، در واقع معماران پایگاه داده هستند و پایگاه داده را مطابق با خواسته های مدیران پایگاه داده و بر اساس نیازهای کلیه کاربران نهائی طراحی می‌کنند.
د) برنامه‌نویسان: این گروه از افراد، برنامه‌های کاربردی لازم برای کار با پایگاه داده را مطابق با نیازمندیهای کاربران طراحی و پیاده‌سازی می‌کنند. در واقع وظیفه آنها طراحی و ایجاد صفحات ورود داده و گزارشات مورد نیاز کاربران نهائی است.
ه) کاربران نهائی[2]: کاربران استفاده کننده از سیستم (اپراتورها، مدیران سازمان‌ها و ...) می‌باشند. این دسته از کاربران، اطلاعات زیادی در مورد پایگاه داده و یا برنامه‌نویسی ندارند.

۱-۴-۴ داده

داده در سیستم پایگاه داده‌ها، شامل داده‌های مورد نیاز کاربران نهایی و فراداده‌ها است.

۱-۵ انواع سیستم پایگاه داده‌ها

DBMSها را بر اساس تعداد کاربران، تمرکز و یا توزیع شدگی داده ها و نوع کاربرد می توان به چند دسته تقسیم کرد. اگر در هر لحظه تنها یک کاربر اجازه کار با پایگاه داده را داشته باشد، DBMS را تک کاربره[3] می نامند. به عبارتی اگر کاربر A در حال کار با پایگاه داده باشد، کاربران B و C باید منتظر بمانند تا کار کاربر A به اتمام برسد. چنانچه DBMS تک کاربره روی یک کامپیوتر شخصی (PC) اجرا شود آن را پایگاه داده رومیزی[4] نامند. در مقابل، DBMS های چند کاربره، همزمان اجازه استفاده چندین کاربر را می‌دهند. اگر تعداد کاربران از ۵۰ کاربر در هر لحظه کمتر باشد، آن را پایگاه داده گروه کاری[5] و چنانچه توسط کل بخشهای یک سازمان مورد استفاده قرار گیرد، آن را پایگاه داده شرکتی[6] می نامند.
نحوه توزیع داده‌ها نیز یکی از معیارهای طبقه بندی پایگاه داده است. DBMSهایی که اجازه نمی دهند داده ها روی چند سایت پخش شوند، DBMSهای متمرکز[7] و DBMSهایی که امکان توزیع داده ها روی چند سایت را فراهم می‌کنند، DBMSهای توزیع شده[8] نامیده می‌شوند.
موارد کاربرد پایگاه داده نیز یکی از معیارهای گروه بندی پایگاه داده می‌باشد. DBMSها را می توان بر اساس کاربرد آنها به ۲ دسته تراکنش گرا[9] و تصمیم گیرنده[10] تقسیم کرد. در سیستم‌های مبتنی بر تراکنش مانند سیستم‌های بانک، فروش و ... سیستم لازم است در واکنش به عملی از طرف کاربر به سرعت پاسخی تولید نموده و یا عملیاتی را انجام دهد. مثلاً در سیستم فروش به محض صدور فاکتور موجودی انبار باید به روز رسانده شود. در DBMS تصمیم گیرنده، هدف، تولید اطلاعاتی است که مدیران رده بالای سازمان را در اتخاذ تصمیم های استراتژیک یاری نماید. مثلاً یک DBMS تصمیم گیرنده ممکن است داده های فراوانی

[1] Database Designers
[2] End Users
[3] Single-User
[4] Desktop Database
[5] Work-group Database
[6] Enterprise
[7] Centralized DBMS
[8] Distributed DBMS
[9] Transactional
[10] Decision Support

را از منابع مختلف دریافت کرده و فرمول مناسب برای تضمین قیمت یک محصول را استخراج کند و یا مثلاً وضعیت آب و هوا را پیش بینی کند.

نکته: در سیستم‌های تراکنش گرا، سرعت پاسخگویی و واکنش DBMS مهم است اما در پایگاه داده تصمیم گیرنده، دقت و صحت اطلاعات تولید شده توسط DBMS.

۱-۶ وظایف DBMS

۱- *تأمین استقلال داده ها و ساختاری*: DBMS به طور اتوماتیک یک دیکشنری داده برای پایگاه داده تهیه می کند و مشخصات کلیه داده ها (اعم از داده های مربوط به موجودیت‌ها و روابط میان موجودیت‌ها) را در این دیکشنری ذخیره می کند. قبل از رجوع به هر قلم داده، DBMS به دیکشنری داده رجوع کرده و از مشخصات ساختاری آن آگاه می‌شود. اگر طراح پایگاه داده فیلد جدیدی به جدول اضافه کند، این تغییر در دیکشنری داده ها منعکس می‌شود. به این ترتیب نیازی به تغییر کد برنامه‌هایی که با جدول مورد نظر سر و کار دارند نمی‌باشد. در واقع، DBMS با ایجاد دیکشنری داده، غالباً باعث مصونیت کدهای برنامه‌نویسی از تغییرات انجام شده در ساختارهای پایگاه داده می‌شود.

۲- مدیریت ذخیره *سازی داده*[1]: DBMS ساختارهای پیچیده لازم برای ذخیره داده ها را خود ایجاد کرده و طراحان پایگاه داده ار از وظیفه سنگین تعریف خصوصیات فیزیکی داده ها، معاف می کند. اکثر DBMSهای کنونی نه تنها ذخیره سازی داده ها را خود مدیریت می‌کنند، بلکه ابزارهایی برای طراحی فرم‌های ورود داده، تعریف گزارشات، تعیین قوانین اعتبارسنجی[2] محیط عملیاتی (مثلاً در محیط دانشگاه، بازه‌ی قابل قبول برای نمره دانشجو عددی بین ۰ تا ۲۰ است)، و ورود تصاویر و ... به پایگاه داده، در اختیار کاربر قرار می دهد.

۳- *تبدیل فرمت داده ها میان محیط فیزیکی و محیط منطقی*: DBMS وظیفه دارد داده های وارد شده توسط کاربران را به فرمت قابل قبول برای ذخیره سازی روی محیط فیزیکی تبدیل کند و از طرفی هنگام بازیابی داده ها از محیط فیزیکی آنها را به فرمت قابل فهم توسط کاربران تبدیل کند.

۴- مدیریت *امنیت*: امنیت در پایگاه داده ها به این معناست که هر کاربر تنها در حوضه اختیارات و وظایف خود به داده ها دسترسی داشته باشد (مثلاً در محیط دانشگاه، یک کارمند امور مالی نباید به نمرات دانشجویان دسترسی داشته باشد). وظیفه تعیین اختیارات هر کاربر در سازمان بر عهده مدیر پایگاه داده می‌باشد که مدیر پایگاه داده از طریق مصاحبه با مدیر سازمان، از حدود اختیارات هر کاربر آگاه می‌شود.

۵- *تأمین امکان دسترسی مشترک چندین کاربر به پایگاه داده*

۶- مدیریت *تهیه نسخه پشتیبان*[3] *و ترمیم*[4] *پایگاه داده*: DBA می تواند برنامه زمانی مورد نظر برای تهیه نسخه پشتیبان و محل ذخیره شدن آن را برای DBMS مشخص کند. DBMS وظیفه دارد طبق برنامه

[1] Data Storage
[2] Validation Rules
[3] Backup
[4] Recovery

دیکته شده، عمل تهیه نسخه پشتیبان را انجام دهد. این برنامه زمانبندی می تواند مثلاً روزانه، هفتگی و یا ... باشد.

۷- *تأمین جامعیت داده ای:* مفهوم جامعیت داده ای شامل ۲ مفهوم اعتبار داده[1] و سازگاری داده[2] می‌باشد. اعتبار داده ها به این معناست که هیچ مقدار نا معتبری[3] در پایگاه داده وارد نشود، مثلاً نمره ۲۲ در سیستمی که بازه نمره قابل قبول در آن بین ۰ تا ۲۰ است، یک داده نامعتبر محسوب می‌شود. سازگاری داده ها به این معناست که در صورت ذخیره یک داده در چندین محل مقدار همه آنها یکسان باشد.

۸- تأمین زبان پرس و جو[4] و ابزارهای مدیریت پایگاه داده: زبان پرس و جو، زبانی بسیار ساده است که به کاربران اجازه می دهد دستوری را روی پایگاه داده صادر کنند بدون آنکه از چند و چون نحوه انجام آن مطلع باشند.

۹- تأمین امکان دسترسی به پایگاه داده از طرق مختلف: DBMSهای کنونی امکان دسترسی به پایگاه داده از روش های مختلف از جمله از طریق Web را در اختیار کاربران قرار می دهند.

۱-۷ معماری سیستم‌های پایگاه داده‌ها

برای اینکه جزئیات ذخیره و بازیابی داده ها کاملاً از دید کاربر پوشیده باشد ANSI[5] یک معماری ۳ لایه برای سیستم‌های پایگاه داده ارائه کرده است.

۱- *لایه فیزیکی یا داخلی*[6]: در این لایه، داده های فیزیکی همان‌گونه که روی محیط فیزیکی ذخیره شده اند نمایش داده می‌شود.

۲- *لایه ادراکه یا انتزاعی*[7]: دیدی است که طراح پایگاه داده نسبت به کلیه موجودیت‌ها و ارتباطات میان آنها دارد. این دید یک دید جامع است که دید کلیه کاربران نهائی از روی آن استخراج می‌شود. وظیفه تهیه این دید جامع، بر عهده طراح پایگاه داده می‌باشد.

۳- *لایه خارجی*[8]: شامل دید خارجی کلیه کاربران است و دیدی است که هر کاربر نسبت به اطلاعات ذخیره شده دارد. این دید، لزوماً با دید ادراکی یکسان نیست و هر کاربر می تواند از نقطه نظر متفاوتی به داده ها نگاه کند. همچنین ممکن است هر کاربر از چند دید خارجی متفاوت استفاده کند.

مثال: برای درک تفاوت میان دید ادراکی و دید خارجی، محیط یک دانشگاه را در نظر بگیرید. دید طراح باید شامل نیازمندی‌های کلیه بخش های دانشگاه شامل آموزش، امور مالی، امور دانشجویی و ... باشد. فرض کنید این دید شامل جداول زیر باشد:

دانشجو (شماره دانشجویی، نام، نام خانوادگی، رشته، سال ورود، ش.ش، محل صدور، ت.ت، نوع بیمه و ...)

درس (شماره درس، نام درس، تعداد واحد، نوع درس و ...)

---

[1] Data Validation
[2] Data Consistency
[3] Invalid
[4] Query Language
[5] American National Standards Institute
[6] Physical or Internal Layer
[7] Conceptual Layer
[8] External Layer

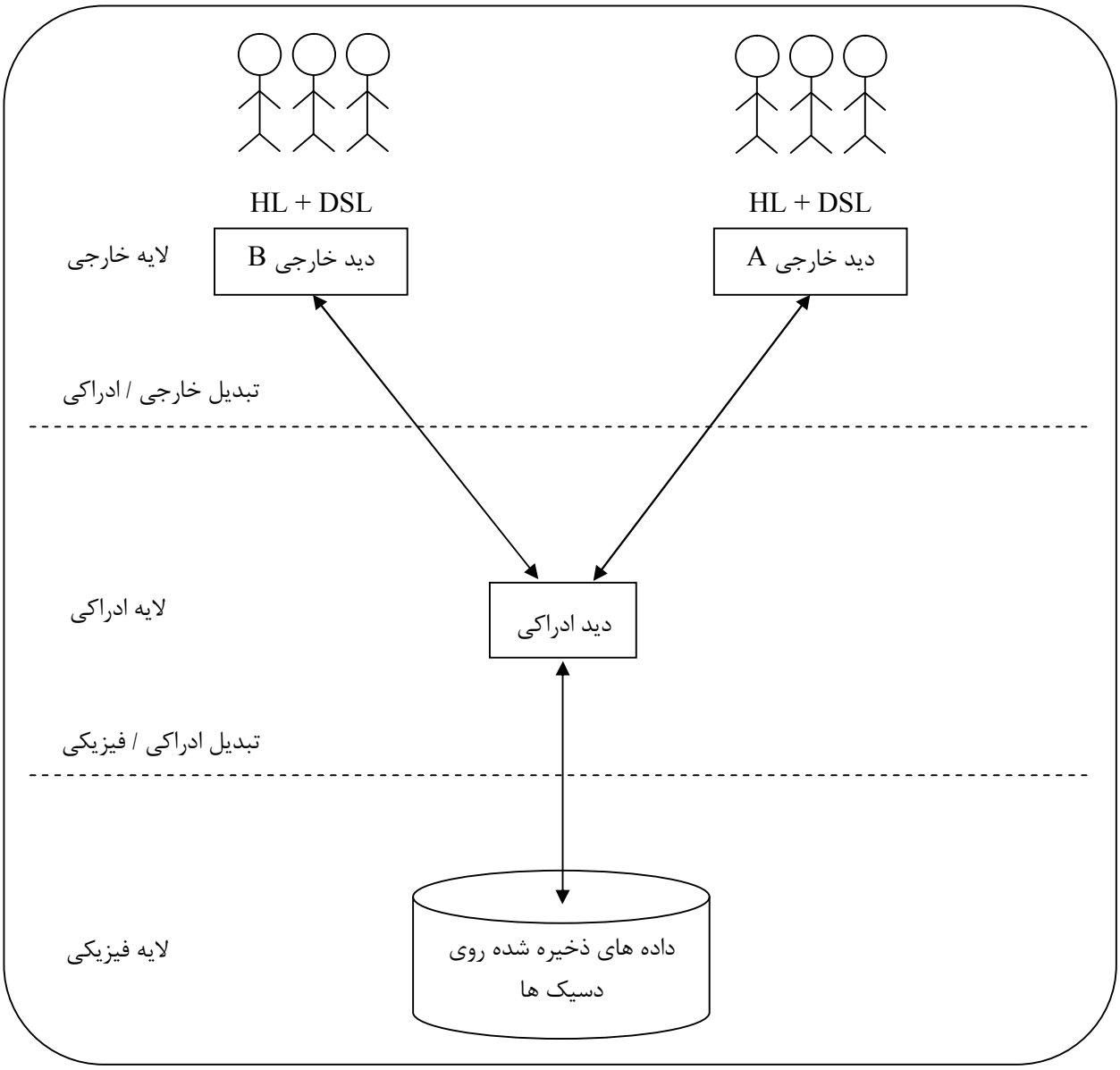
استاد (شماره استاد، نام، نام خانوادگی، مدرک و ...)

نمرات (شماره دانشجویی، شماره درس، ترم، نمره)

بسیاری از اطلاعات فوق ممکن است خارج از محدوده کاری یک بخش باشند، مثلاً برای کارمند آموزش، شماره شناسنامه و یا نوع بیمه یک دانشجو اهمیتی ندارد؛ پس دلیلی ندارد که این فیلدها را ببینند. از طرف دیگر، ممکن است در بسیاری از گزارشات قسمت آموزش، معدل کل دانشجو مورد استفاده باشد. در هیچ‌یک از جداولی که طراحی در نظر گرفته است، ویژگی معدل وجود ندارد ولی می توان مقدار این ویژگی را بر اساس نمرات دانشجو و تعداد واحد هر درس محاسبه کرد. پس DBA می تواند یک دید خارجی مخصوص کارمندان آموزش تعریف کند که به صورت زیر باشد:

دانشجو از دید آموزش (شماره دانشجویی، نام، نام خانوادگی، رشته، سال ورود، معدل کل)

دانشجو از دید مالی (شماره دانشجویی، نام، نام خانوادگی، مبلغ وام گرفته، تعداد اقساط وام، مبلغ هر قسط، تاریخ سررسید هر قسط)



معماری ۳ لایه ای ANSI

در هر برنامه کاربردی که با پایگاه داده‌ها سر و کار دارد، از دو دسته زبان استفاده می‌شود:

۱- *زبان فرعی داده ای یا DSL*[1]

۲- *زبان میزبان یا HL*[2]

زبان فرعی داده ای

از دستورات زبان فرعی داده‌ای برای کار با داده ها استفاده می‌شود که به ۳ دسته تقسیم می‌شوند:

الف) زبان تعریف داده ها یا DDL[3]: از این دستورات برای تعریف ساختار جداول و ایندکس ها و غیره استفاده می‌شود.

```
create table course(c# int, cname char(20), unit int, primary key(c#))
```

course

c#	cname	unit

ب) *زبان دستکاری داده یا DML*[4]: از این دستورات برای بازیابی، درج، حذف و اصلاح اطلاعات جدول استفاده می‌شود.

```
insert into course (c#, canme, unit) values (1500, "ریاضی", 3)
```

course

c#	cname	unit
1500	ریاضی	3

ج) *زبان کنترل داده ها یا DCL*[5]: از این دستورات برای مدیریت مجوزهای دسترسی به پایگاه داده استفاده می‌شود.

```
grant update (cname) on course to Ali
```

نکته: معروف ترین زبان فرعی داده ای، زبان SQL[6] است.

[1] Data Sub Language
[2] Host Language
[3] Data Definition Language
[4] Data Manipulation Language
[5] Data Control Language
[6] Structured Query Language

زبان میزبان

زبانهای فرعی داده ای معمولاً فاقد امکانات لازم برای تعریف متغییرها، کنترل ظاهر برنامه، ایجاد حلقه‌ها، تست شرط ها و غیره هستند. به همین دلیل، از دستورات یک زبان برنامه‌نویسی مانند دلفی، ویژوال C و ... برای کارهایی مثل طراحی منو، ایجاد حلقه، تست شرط و ... استفاده می‌شود.

۱-۸ مدل‌های پایگاه داده‌ها

مدل پایگاه داده مجموعه ای است از ساختارهای منطقی که ساختار داده ها و روابط میان داده ها را نمایش می دهد. مدلهای پایگاه داده را می توان به ۲ گروه اساسی تقسیم کرد: مدلهای انتزاعی[۱] و مدل‌های پیاده‌سازی[۲].

✵**مدل‌های انتزاعی:** ساختار منطقی داده ها و روابط منطقی میان آنها را نشان می دهند. این مدل‌ها تنها مشخص می‌کنند در پایگاه داده چه چیزی باید ارائه شود و به چگونه پیاده‌سازی نمی پردازند. مدل‌های انتزاعی شامل: مدل موجودیت – رابطه یا ERM[3] و مدل شی گرا یا OOM[4] می‌باشد.

✵**مدلهای پیاده‌سازی:** مشخص می‌کنند آنچه در مدلهای انتزاعی در نظر گرفته شده است چگونه بایستی پیاده‌سازی شود. این مدلها شامل مدل سلسله مراتبی[۵]، مدل شبکه ای[۶] و مدل رابطه ای[۷] می‌باشند.

نکته: علاوه بر مدل‌های عنوان شده، مدل‌های دیگری هم وجود دارند که عبارتند از[۸]:

۱) Object-Relational Database Model: این مدل پایگاه داده، همان مدل رابطه ای است با این تفاوت که به کاربر اجازه می دهد نوع داده و متدهای تعریف شده مربوط به خود را داشته باشد. هدف اصلی این روش، این است که به توسعه دهندگان نرم‌افزار اجازه داده شود به سطح انتزاع بالاتری برسند.

۲) Associative

۳) Concept-oriented

۴) Multi-dimensional

۵) Star schema

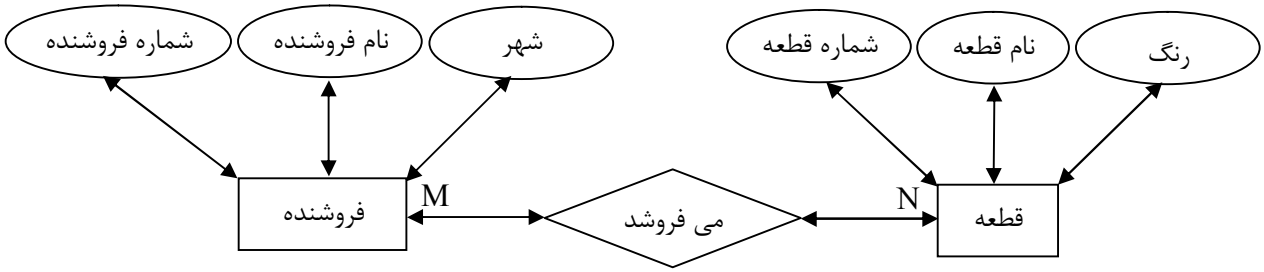
۶) XML database

برای درک مدل‌های پیاده‌سازی یک سیستم فروش قطعه توسط فروشندگان مختلف را در نظر بگیرید. موجودیت‌های اساسی این سیستم عبارتند از: فروشندگان و قطعات. رابطه ای که بین این دو موجودیت وجود دارد فروش است. طبق قوانین این سیستم، هر فروشنده ممکن است چندین نوع قطعه را بفروشد و هر

[1] Conceptual Models
[2] Implementation Models
[3] Entity-Relationship Model
[4] Object-Oriented Model
[5] Hierarchical Database Model
[6] Network Database Model
[7] Relational Database Model

[8] برگرفته از سایت http://en.wikipedia.org

نوع قطعه ممکن است توسط چندین فروشنده مختلف فروخته شود. بنابراین، این رابطه یک رابطه چند به چند یا M:N است. مدل ER این سیستم به شکل زیر می‌باشد:



اطلاعات فروشندگان:

شماره فروشنده = S1	نام فروشنده = تهران قطعه	شهر = تهران
شماره فروشنده = S2	نام فروشنده = خودیاران	شهر = تهران
شماره فروشنده = S3	نام فروشنده = یزد قطعه	شهر = یزد
شماره فروشنده = S4	نام فروشنده = البرز	شهر = اصفهان

اطلاعات قطعه:

شماره قطعه = P1	نام قطعه = تیر آهن	رنگ = مشکی
شماره قطعه = P2	نام قطعه = آرماتور	رنگ = مشکی
شماره قطعه = P3	نام قطعه = سیمان	رنگ = طوسی
شماره قطعه = P4	نام قطعه = آلومینیوم	رنگ = سفید

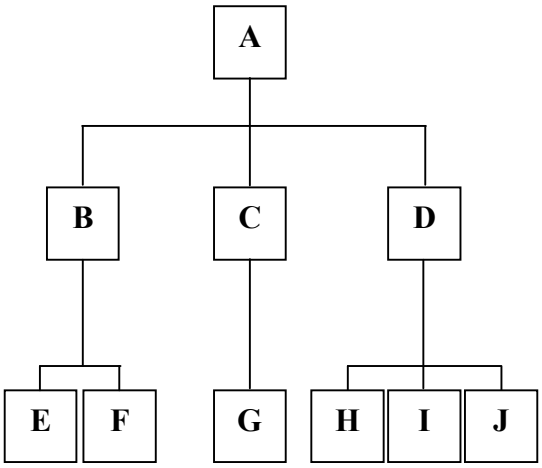
اطلاعات مربوط به فروش:

فروشنده S1، ۱۰،۰۰۰ کیلوگرم از قطعه P2 فروخته است.
فروشنده S1، ۳،۰۰۰ کیلوگرم از قطعه P4 فروخته است.
فروشنده S2، ۲،۰۰۰ کیلوگرم از قطعه P1 فروخته است.
فروشنده S2، ۴،۰۰۰ کیلوگرم از قطعه P2 فروخته است.
فروشنده S2، ۳،۰۰۰ کیلوگرم از قطعه P4 فروخته است.
فروشنده S3، ۳،۰۰۰ کیلوگرم از قطعه P4 فروخته است.

۱-۸-۱ مدل سلسله مراتبی

در این مدل داده ها در گره های یک درخت‌واره (گراف)، ذخیره می‌شوند. درخت‌واره، یک گراف غیرچرخشی و متصل است که یک ریشه دارد. در درخت‌واره، هر گره می تواند چند فرزند و تنها یک پدر

داشته باشد. به همین دلیل مدل سلسله مراتبی تنها برای پیاده‌سازی روابط یک به چند مناسب است و برای پیاده‌سازی روابط چند به چند مناسب نیست.



چون در مدل سلسله مراتبی امکان پیاده‌سازی روابط چند به چند وجود ندارد برای پیاده‌سازی مثال مورد نظر ۲ امکان وجود دارد:

۱- طرف فروشنده را یک و طرف قطعه را چند در نظر بگیریم

شماره فروشنده	نام فروشنده	شهر

1

M

شماره قطعه	نام قطعه	رنگ قطعه	مقدار فروخته شده

۲- طرف قطعه را یک و طرف فروشنده را چند در نظر بگیریم

شماره قطعه	نام قطعه	رنگ قطعه

1

M

شماره فروشنده	نام فروشنده	شهر	مقدار فروخته شده

برای این مثال، از روش دوم استفاده می کنیم.

P1	تیرآهن	مشکی

S2	خودیاران	تهران	2000

P2	آرماتور	مشکی

S1	تهران قطعه	تهران	10000

S2	خودیاران	تهران	4000

P3	سیمان	طوسی

P4	آلومینیم	سفید

S1	تهران قطعه	تهران	3000

S2	خودیاران	تهران	3000

S3	یزد قطعه	یزد	3000

عملیات بازیابی در مدل سلسله مراتبی:

۲ پرس و جوی قرینه زیر را داریم:

۱- نام فروشندگانی را بیابید که قطعه P2 را فروخته اند.

کافی است در قطعات، قطعه شماره P2 را یافته ونام کلیه فروشندگان زیر این قطعه را واکشی کنیم.

۲- نام قطعاتی را بیابید که توسط فروشنده S2 فروخته شده اند.

برای یافتن جواب باید کلیه قطعات را پیمایش کرده، برای هر قطعه جستجوی عمقی انجام دهیم، اگر S2 جزو فروشندگانی باشد که زیر قطعه مورد نظر قرار دارند، باید نام آن را در لیست جواب ها قرار داد. مشاهده می‌کنید با این که پرس وجو ها کاملا قرینه یکدیگرند، عملیات صورت گرفته برای به دست آمدن نتایج آنها کاملاً با یکدیگر متفاوت است.

عملیات درج در مدل سلسله مراتبی:

در این مدل، اطلاعات فروشنده S4 در نظر گرفته نشده است زیرا S4 هنوز قطعه ای نفروخته است. به عبارتی نمی توان یک گره بدون پدر در این مدل ایجاد کرد. بنابراین، مدل سلسله مراتبی در عملیات درج ناهنجاری دارد.

عملیات حذف در مدل سلسله مراتبی:

اگر بخواهیم اطلاعات قطعه P4 را از پایگاه داده حذف کنیم، با حذف این قطعه کلیه گره های زیر این قطعه یعنی گره های مربوط به S1، S2 و S3 نیز به طور خودکار حذف می‌شوند. از آنجا که اطلاعات مربوط به S1 و S2 در گره های دیگر نیز وجود دارند، حذف آنها مشکلی را ایجاد نخواهد کرد. ولی حذف گره S3 باعث بروز مشکلاتی خواهد شد چرا که اطلاعات این قطعه تنها در همین گره ثبت شده است. به عبارتی، با حذف اطلاعات مربوط به قطعه P4، اطلاعات فروشنده S3 را نیز از دست می دهیم و بنابراین این مدل ناهنجاری حذف دارد. از طرف دیگر اگر بخواهیم اطلاعات S1 را حذف کنیم، از آنجایی که این اطلاعات در چندین جا ذخیره شده اند، باید گره های متعددی را حذف کنیم.

عملیات اصلاح در مدل سلسله مراتبی:

اگر بخواهیم رنگ یک قطعه را تغییر دهیم فقط یک تغییر در گره مورد نظر نیاز است ولی اگر بخواهیم شهر یک فروشنده را تغییر دهیم، باید گره های متعددی را تغییر دهیم. به عبارتی برای حفظ سازگاری داده لازم است اصلاح منتشر شونده صورت گیرد.

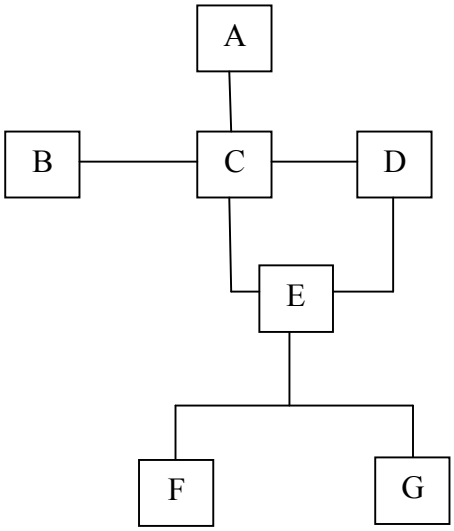
معایب مدل سلسله مراتبی:

۱- برای طراحی و پیاده‌سازی روابط چند به چند مناسب نیست.
۲- تهیه برخی گزارشات در این مدل بسیار وقت‌گیر است.
۳- این مدل در عملیات درج، حذف و اصلاح، ناهنجاری دارد.

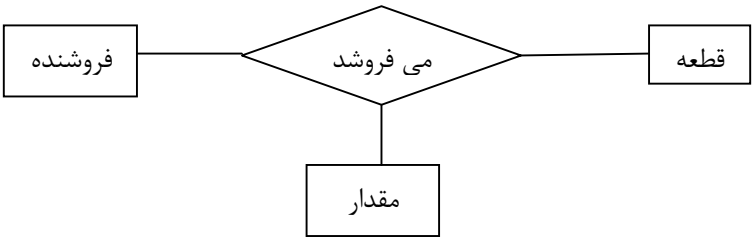
۴- به دلیل استفاده از اشاره گر ها و وابستگی به آدرس های فیزیکی ذخیره سازی، مدل کاملاً به محیط فیزیکی ذخیره سازی وابسته است و با اعمال هر گونه تغییر در ساختار داده ها، طراح پایگاه داده وبرنامه‌نویسان به زحمت می افتند.

۱-۸-۲ مدل شبکه ای

در مدل شبکه ای اطلاعات در گره های یک گراف دل‌خواه ذخیره می شوتد. در این گراف، هر گره می تواند چند پدر و چند فرزند داشته باشد. بنابراین، مدل شبکه‌ای برای پیاده‌سازی روابط چند به چند مناسب است.



در مثال مورد نظر ما، از مقدار فروش برای اتصال گره‌های فروشندگان و قطعات استفاده می کنیم:



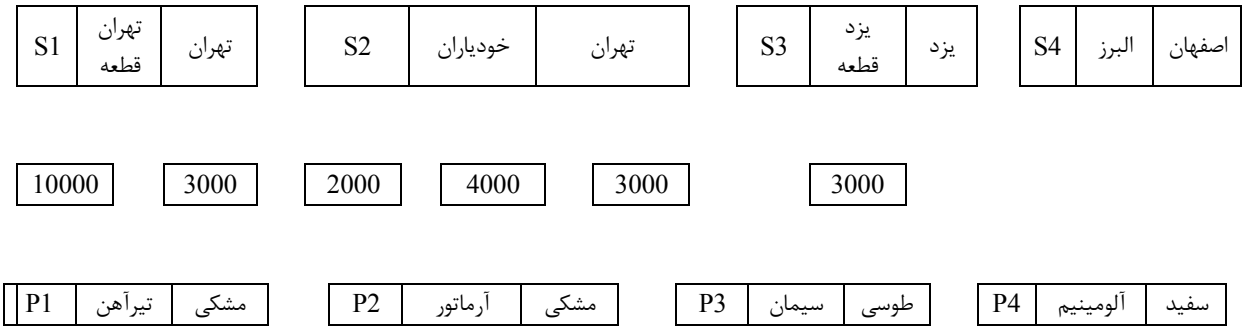
قدم ۱: برای هر فروشنده، مقدار فروش کلیه قطعات را در نظر گرفته و کلیه این گره ها را از طریق اشاره‌گر به هم متصل می کنیم.

S1	تهران قطعه	تهران	S2	خودیاران	تهران	S3	یزد قطعه	یزد	S4	البرز	اصفهان

10000	3000	2000	4000	3000	3000

قدم ۲: برای هر قطعه، مقدار فروخته شده توسط یک فروشنده را در نظر گرفته، کلیه این گره ها را توسط اشاره گر به هم متصل می کنیم. مثلاً قطعه P2 توسط فروشنده S1 به مقدار 10000 کیلوگرم و توسط

فروشنده S2 به مقدار 4000 کیلوگرم فروخته شده است. بنابراین، از قطعه P2 یک اشاره گر به 10000یی که در طرف دیگر به فروشنده S1 وصل است و از 10000 یک اشاره گر به 4000یی که در طرف دیگر به فروشنده S2 وصل است و از 4000 یک اشاره گر به خود P2 رسم می کنیم. این کار را برای کلیه قطعات، تکرار می کنیم.



عملیات بازیابی در مدل شبکه ای:

۲ پرس و جوی قرینه زیر را داریم:

۱- نام فروشندگانی را بیابید که قطعه P2 را فروخته اند.

برای دریافت جواب باید در میان قطعات، قطعه P2 را یافته، اتصالاتی را که به این قطعه متصل هستند پیمایش کرده و نام فروشندگانی را که در سمت دیگر این اتصالات قرار دارند به دست آوریم.

۲- نام قطعاتی را بیابید که توسط فروشنده S2 فروخته شده اند.

برای یافتن جواب باید در میان فروشندگان، فروشنده S2 را یافته، اتصالاتی را که به این فروشنده متصل هستند پیمایش کرده و نام قطعاتی را که در سمت دیگر این اتصالات قرار دارند به دست آوریم.

مشاهده می کنید که برای پرس و جو های قرینه، عملیات کاملاً مشابه ای باید صورت گیرد.

عملیات درج در مدل شبکه ای:

این مدل در عملیات درج هیچ گونه ناهنجاری ندارد.

عملیات حذف در مدل شبکه ای:

این مدل در عملیات حذف هیچ گونه ناهنجاری ندارد.

عملیات اصلاح در مدل شبکه ای:

این مدل در عملیات اصلاح هیچ گونه ناهنجاری ندارد و نیازی به اصلاح منتشر شونده نیست.

محاسن مدل شبکه ای نسبت به مدل سلسله مراتبی:

۱- برای پیاده‌سازی روابط چند به چند مناسب است.

۲- در مدل شبکه ای، ناهنجاری درج، حذف و اصلاح وجود ندارد.

معایب مدل شبکه ای نسبت به مدل سلسله مراتبی:

۱- طراحی و پیاده‌سازی و برنامه‌نویسی در مدل شبکه ای بسیار پیچیده است.

۲- این مدل به دلیل استفاده از اشاره گر ها، کاملاً به محیط فیزیکی ذخیره سازی داده ها وابسته است.

۱-۸-۳ مدل رابطه ای

در سال ۱۹۷۰ یک نشریه کامپیوتری به نام COMMUNICATIONS OF ACM مقاله ای با عنوان «مدل رابطه ای داده‌ها برای بانک های اطلاعاتی اشتراکی بزرگ» منتشر ساخت.

این مقاله نوشته دکتر ای. اف. کاد عضو آزمایشگاه تحقیقاتی سن جوز IBM بود. این مقاله، تئوری ریاضی پایگاه داده ای رابطه ای، در زمینه اینکه چگونه می توان داده ها را با استفاده از یک ساختار جدولی ذخیره و مدیریت نمود، پایه گذاری کرد. در زمینه علوم کامپیوتر به ندرت می توان به مقاله دیگری اشاره نمود که به اندازه مقاله دکتر ای. اف. کاد چنان تأثیر شگرفی بر تولیدکنندگان بانک های اطلاعاتی رابطه ای گذاشته باشد.

دکتر کاد در مقاله خود، عناصر پایگاه داده رابطه ای همچون رابطه ها، صفات، دامنه ها و عملگرهای رابطه‌ای را تشریح نمود.

مقاله کاد به تشریح نوعی سیستم ذخیره سازی اطلاعات پرداخت که دارای ۳ ویژگی مورد نیاز آن زمان بود:

- استقلال منطقی داده ها: این خصیصه ضروری به معنی آن است که تغییر در یک صفت (ستون) مانند افزایش یا کاهش طول آن هیچ گونه تأثیر محسوسی بر سایر صفات (ستون ها) در همان رابطه (جدول) نداشته باشد. این ویژگی به علت کاهش قابل ملاحظه هزینه نگهداری نرم‌افزار، مورد توجه قرار گرفت.
- جامعیت داده ها: بر خلاف سایر سیستم‌های پایگاه داده، پایگاه داده رابطه ای برنامه کاربردی را از قید محدودیت های جامعیت[1] رها می سازد. کاد به تشریح دو ویژگی موجود در پایگاه داده رابطه ای یعنی جامعیت داده ها و جامعیت ارتباطی پرداخت.
- پرس و جوی ویژه: این ویژگی جستجو داده های مورد نظر، جستجو در پایگاه داده ای را بدون برنامه‌نویسی و دانستن چگونگی انجام اعمال توسط کاربر، میسر می سازد. درک محدودیت های موجود در سیستم‌های پایگاه داده در آن زمان حائز اهمیت بود. کاربر عادی که برنامه‌نویسی نمی دانست قادر نبود داده های مورد نیاز خود را بدون نیاز به برنامه‌نویسان، تجزیه و تحلیل کند.

در مدل رابطه ای، داده ها در داخل جداول ذخیره می‌شوند. در این مدل برای هر یک از موجودیت‌ها و یا روابط میان موجودیت‌ها یک جدول مجزا در نظر گرفته می‌شود. شکل زیر، مدل رابطه ای مثال کوچک ما را نشان می دهد:

[1] Enforcing

Supplier

S#	Sname	City
S1	تهران قطعه	تهران
S2	خودیاران	تهران
S3	یزد قطعه	یزد
S4	البرز	اصفهان

Part

P#	Pname	Color
P1	تیرآهن	مشکی
P2	آرماتور	مشکی
P3	سیمان	طوسی
P4	آلومینیم	سفید

SP

S#	P#	qty
S1	P2	10000
S1	P4	3000
S2	P1	2000
S2	P2	4000
S2	P4	3000
S3	P4	3000

عملیات بازیابی در مدل رابطه ای:

دو پرس و جوی قرینه قسمت قبل را در نظر بگیرید:

۱- فروشندگانی را بیابید که قطعه P2 را فروخته اند.

در جدول SP به راحتی می توان شماره فروشندگانی را که قطعه P2 را فروخته اند به دست آورد. برای به دست آوردن نام این فروشندگان کافی است به جدول Supplier مراجعه کنیم.

۲- نام قطعاتی را بیابید که توسط فروشنده S2 فروخته شده اند.

در جدول SP به راحتی می توان شماره قطعاتی را که توسط S2 فروخته شده اند به دست آورد. برای به دست آوردن نام این قطعات، کافی است به جدول Part مراجعه کنیم.

مشاهده می کنید که برای به دست آوردن نتایج این دو پرس و جو، عملیات مشابه و بسیار ساده ای باید انجام شود.

عملیات درج در مدل رابطه ای:

در جدول Supplier به راحتی می توان اطلاعات فروشنده ای مانند S4 را که هنوز هیچ قطعه ای نفروخته است و در جدول Part، اطلاعات P3 را که هنوز توسط هیچ فروشنده ای فروخته نشده است، وارد کرد. بنابراین، مدل رابطه ای در عملیات درج، ناهنجاری ندارد.

عملیات حذف در مدل رابطه ای:

می توان اطلاعات فروشنده ای مانند S1 را در جدول Supplier حذف کرد. در این صورت باید کلیه فروش‌های مربوط به S1 در جدول SP نیز حذف شوند ولی اطلاعات هیچ قطعه ای از دست نمی رود. به

همین ترتیب می توان اطلاعات قطعه ای مانند P4 را در جدول Part حذف کرد. در این صورت، برای حفظ سازگاری داده ها، بایستی کلیه فروش های مربوط به P4 در جدول SP نیز حذف شود ولی اطلاعات هیچ فروشنده ای از دست نمی رود. پس این مدل در عملیات حذف، ناهنجاری ندارد.

عملیات اصلاح در مدل رابطه ای:

برای آنکه رنگ قطعه P3 را از طوسی به مشکی تغییر دهیم، کافی است این تغییر را تنها در یکی از سطرهای جدول Part اعمال کنیم. به همین ترتیب، برای تغییر شهر فروشنده S1 از تهران به ساری، کافی است یکی از سطرهای جدول Student را تغییر دهیم و نیازی به اصلاح منتشر شونده نیست پس این مدل در عملیات اصلاح، ناهنجاری ندارد.

محاسن مدل رابطهای نسبت به مدل شبکه ای:

در این مدل مشروط بر آنکه جداول به درستی طراحی شده باشند، همانند مدل شبکه ای، ناهنجاری درج و اصلاح و حذف ندارد. مدل رابطه ای برای پیاده‌سازی روابط یک به چند و چند به چند کاملاً مناسب است. علاوه بر این موارد، مدل رابطه ای نسبت به مدل شبکه ای دو حسن عمده دارد:

۱- طراحی و پیاده‌سازی آن بسیار ساده است.

۲- به علت عدم وابستگی به آدرس های فیزیکی، طراحان پایگاه داده و برنامه‌نویسان را از درگیر شدن با جزئیات ذخیره سازی فیزیکی داده ها معاف می کند.

۱-۹ تراکنش[1]

هرگونه برنامه‌ای که توسط کاربر در محیط بانک اطلاعات، اجرا می‌شود، تراکنش نام دارد. تفاوت اصلی تراکنش با یک برنامه معمولی در محیط غیر بانکی، این است که تراکنش همواره به سیستم مدیریت بانک اطلاعات (DBMS) تسلیم می‌شود و این سیستم در اعمال هرگونه کنترل و حتی به تعویق انداختن و ساقط کردن آن، آزادی عمل دارد. هدف اصلی از این‌گونه کنترل‌ها و حذف و تعویق‌ها، حفظ جامعیت و صحت بانک اطلاعات است. در بانک اطلاعات، آن‌چه در درجه اول اهمیت قرار دارد، «داده» است نه برنامه. داده‌های بانک اطلاعات را مانا[2] می‌نامند زیرا برنامه‌ها می‌آیند و می‌روند اما داده‌ها می‌مانند. مثلاً یک حساب بانکی، زمانی باز می‌شود و به طور متناوب به آن پول واریز و از آن برداشت می‌گردد. یک برنامه که پولی را به حساب می‌ریزد یا برداشت می‌کند آنقدرها مهم نیست. مهم این است که موجودی حساب‌ها اشتباه نباشد. بدیهی است که صحت داده‌ها از صحت برنامه‌ها نشأت می‌گیرد.

از همان ابتدای انقلاب بانک اطلاعات، این سؤال ذهن متخصصین را به خود مشغول داشته بود که «چه کنترل‌هایی لازم است روی برنامه‌ها اعمال شود تا صحت و جامعیت بانک اطلاعات تضمین گردد؟». آقای

[1] Transaction
[2] Persistent

جیم گری[1]، در سال ۱۹۸۱ ثابت کرد که چهار کنترل زیر لازم است روی تمامی تراکنش‌ها در بانک اطلاعات اعمال گردد تا صحت و جامعیت آن تضمین شود (موسوم به خواص ACID):

الف) یکپارچگی[2]: این خاصیت، به «همه یا هیچ» موسوم است. منظور این است که یا تمامی دستورالعمل‌های یک تراکنش باید اجرا شود و یا هیچ‌کدام از آن‌ها اجرا نشوند. به عنوان مثال، تراکنشی را در نظر بگیرید که مبلغی پول را از حسابی به حساب دیگر (مثلاً در شهر یا کشوری دیگر) منتقل می‌کند. این تراکنش شامل دو بخش است. بخش اول، پول را از حساب اول برداشت می‌کند و بخش دوم، همان پول را به حساب دوم واریز می‌نماید. این دو بخش ممکن است روی دو کامپیوتر جداگانه اجرا شوند. فرض کنید بخش اول تراکنش اجرا شود اما ناگهان ارتباط با ماشین دوم قطع گردد بخش دوم قابل انجام نباشد. بدیهی است که باید پول را به همان حساب اول بازگرداند تا صحت و جامعیت بانک اطلاعات، حفظ شود. این عمل، معادل این است بگوییم هیچ دستورالعملی از تراکنش انجام نشده است.

ب) سازگاری[3] (هم‌خوانی): این خاصیت می‌گوید که هر تراکنش، باید تمامی قوانین جامعیت بانک اطلاعات را رعایت کند. علاوه بر این، فرض می‌شود که تراکنش، یک برنامه صحیح است. مثلاً در برنامه انتقال پول، اگر مبلغ برداشت شده با مبلغ واریز شده به حساب دیگر، برابر نباشد، تراکنش غلط است. غالباً چنین مواردی توسط سیستم مدیریت بانک اطلاعات به طور خودکار قابل کنترل نیست. بسیار اتفاق می‌افتد که کاربران، برنامه‌های غلط اجرا می‌کنند و نتیجه غلط می‌گیرند. نه کسی دیگر مقصر است و نه می‌توان جلوی چنین اشتباهاتی را گرفت. بنابراین، بخشی از خاصیت سازگاری، قابل کنترل نیست و باید «فرض» شود. نتیجتاً خاصیت سازگاری، به این صورت بیان می‌شود: «هر تراکنش، اگر به تنهایی اجرا شود، بانک اطلاعات را از حالتی صحیح (سازگار)، به حالت صحیح (سازگار) دیگری منتقل می‌کند».

تا به اینجا مشاهده شد که تراکنش ممکن است دو نوع پایان داشته باشد:

- پایان ناموفق که آن را «سقط[4] (سقوط)» می‌نامند.
- پایان موفق که آن را «انجام[5]» می‌نامند.

آیا دو خاصیت یکپارچگی و سازگاری، برای صحت تراکنش کافی نیست؟ بر طبق این دو خاصیت، هر تراکنش یا به طور صحیح به انجام می‌رسد و یا ساقط می‌شود و بانک اطلاعات به حالت اولیه بازمی‌گردد. آیا این کافی نیست؟

نکته اینجا است که ممکن است تراکنش‌های همروند[6] روی یکدیگر اثر مخرب داشته باشند و همچنین اثر تراکنش‌های انجام شده، به طور تصادفی (مثلاً خرابی دیسک)، از بین برود. این دو پدیده، دو خاصیت دیگر از خواص چهارگانه ACID را شامل می‌شود که در زیر می‌آید.

ج) انزوا[7]: بر طبق این خاصیت، اثر تراکنش‌های همروند روی یکدیگر چنان است که گویا هر کدام در انزوا انجام می‌شود. توجه به این نکته حائز اهمیت است که در بانک اطلاعات، تراکنش‌های همروند وجود دارند

[1] Jim Gray
[2] Atomicity
[3] Consistency
[4] Abort
[5] Commit
[6] Concurrent
[7] Isolation

ولی همروندی آن‌ها کنترل می‌شود تا اثر مخرب روی هم نداشته باشند. این عمل توسط بخشی از سیستم مدیریت بانک اطلاعات (DBMS) به نام واحد کنترل همروندی[1] انجام می‌شود.

د) پایایی[2]: بر اساس این خاصیت، تراکنش‌هایی که به مرحله انجام (Commit) برسند، اثرشان ماندنی است و هرگز به طور تصادفی از بین نمی‌روند. مثلاً اگر مبلغی به حسابی واریز شود و تراکنش مربوطه انجام یافته اعلام شود، حتی در صورت وقوع آتش‌سوزی در آن شعبه بانک، مشتری متضرر نخواهد شد یعنی عمل واریز، قبل از اعلام انجام موفق، در جای دیگری ثبت شده است.

دو عمل یکپارچگی و پایایی، توسط واحدی از سیستم مدیریت بانک اطلاعات به نام واحد مدیریت ترمیم[3]، انجام می‌گیرد.

۱-۱۰ تمرین‌های فصل

۱- یک سیستم پایگاه داده شامل اطلاعات دانشجویان (شماره دانشجویی، نام دانشجو، رشته تحصیلی دانشجو)، اطلاعات دروس (کد درس، نام درس، تعداد واحد درس) و نمرات دانشجویان در دروس مختلف می‌باشد. با فرض داشتن اطلاعات زیر، مدل سلسله مراتبی، مدل شبکه ای و مدل رابطه ای را برای این پایگاه داده رسم کنید.

اطلاعات دانشجویان:

- (۸۰۱۲۱۲ و علی و کامپیوتر)
- (۸۰۱۰۱۰ و زهرا و حسابداری)
- (۷۷۰۰۰۰ و حسن و کامپیوتر)

اطلاعات دروس:

- (۴۴۰۰ و پایگاه داده و ۳)
- (۴۴۰۲ و برنامه سازی و ۳)
- (۴۴۰۳ و سیستم عامل و ۲)
- (۴۴۰۴ و شبکه و ۳)
- (۱۲۰۰ و ادبیات و ۲)

اطلاعات مربوط به نمرات:

- علی در درس پایگاه داده، نمره ۱۸ گرفته است.
- زهرا در درس پایگاه داده، نمره ۱۲ گرفته است.
- حسن در درس پایگاه داده، نمره ۱۸ گرفته است.
- حسن در درس سیستم عامل، نمره ۱۰ گرفته است.
- زهرا در درس سیستم عامل، نمره ۸ گرفته است.
- زهرا در درس برنامه سازی، نمره ۱۰ گرفته است.

[1] Concurrency Control
[2] Durability
[3] Recovery Management

فصل دوم

«پایگاه داده‌های رابطه ای[1]»

۲-۱ مفاهیم پایگاه داده‌های رابطه‌ای

دامنه[2]: مجموعه تمام مقادیر ممکن صفت[3] است. قوانین حاکم بر سیستم، دامنه هر صفت را مشخص می‌کنند. مثلاً دامنه «عدد صحیح» یعنی مجموعه {... -2, -1, 0, 1, 2, ...} و صفتی که دامنه‌اش عدد صحیح است، می‌تواند هر یک از این مقادیر را تا حد گنجایش کامپیوتر مربوطه، در خود جای دهد.

رابطه[4]: زیرمجموعه‌ای از ضرب دکارتی چند دامنه است. مثلاً اگر داشته باشیم D1: String و D2: Integer، آن‌گاه هر مجموعه‌ای که عضوهایش زوج‌های مرتب (D1, D2) باشند، یک رابطه است. بهترین راه نمایش و پیاده‌سازی رابطه، به وسیله جدول است. جدول زیر، نمونه‌ای از رابطه فوق را نشان می‌دهد.

D1: String	D2: Integer
علی	۱۰
رضا	۲۰
.....	.....

یادآوری می‌شود که ضرب دکارتی D1×D2 معادل جدولی است که تمام ترکیب‌های ممکن این دامنه را دربر بگیرد (تعداد عضوهایش محدود است). پس رابطه، هر زیرمجموعه‌ای از آن و از جمله زیرمجموعه تهی (جدول خالی) را شامل می‌شود.

هر رابطه یا جدول، ساختاری دو بعدی شامل سطرها و ستون ها است. در پایگاه داده رابطه ای برای هر نوع موجودیت یا ارتباط میان موجودیت‌ها، یک جدول در نظر گرفته می‌شود. مثلاً برای موجودیت دانشجو یک جدول، برای موجودیت درس یک جدول و برای رابطه ثبت نام (رابطه میان درس و دانشجو) نیز یک جدول در نظر گرفته می‌شود. هر سطر جدول نمایانگر یک موجودیت یا رابطه خاص است.

student

st#	name	course
۷۸۰۱	آرش راد	کامپیوتر
۷۹۰۲	مینا رضایی	هنر
۸۰۰۱	علی راد	هنر
۸۰۰۲	فرزانه رسولی	هنر

[1] Relational Database
[2] Domain
[3] Attribute
[4] Relation

صفت (ویژگی): هر یک از ستون های یک جدول نمایانگر یکی از ویژگی های نوع موجودیت است. در مدل رابطه‌ای، صفت‌ها از دامنه‌های ساده (تجزیه‌ناپذیر) تعریف می‌شوند و دامنه‌های تودرتو، مجاز نیستند. مثلاً اگر صفتی از نوع «تاریخ» تعریف شود، اجزا آن (روز، ماه، سال)، قابل دستیابی نیستند.
مثلاً جدول دانشجو می تواند شامل ستون های st# (شماره دانشجویی)، name (نام دانشجو) و course (رشته تحصیلی دانشجو) باشد.

زوج مرتب (تاپل[1]): ارتباط مجموعه‌ای از مقادیر در یک رابطه است. هر سطر یک جدول، معادل یک تاپل است. مثلاً در جدول student، (هنر – مینا رضایی – ۷۹۰۲) یک تاپل است.

تذکر: در پایگاه داده رابطه ای، مفاهیم رابطه با جدول، مفاهیم تاپل با سطر و مفاهیم ویژگی با ستون کاملاً هم ارز هستند.

نکته: طبق تعریف مجموعه در ریاضیات، مجموعه، عضو تکراری ندارد بنابراین، در یک رابطه نیز عضو تکراری (تاپل یا رکورد تکراری) وجود ندارد. همچنین عناصر یک مجموعه، فاقد نظم هستند پس هیچ لزومی ندارد که تاپل‌های یک رابطه بر اساس یکی از صفت‌ها (مثلاً st#) مرتب باشند.

بدنه[2]: مجموعه تاپل های یک جدول را بدنه آن جدول نامند. به عنوان مثال:

Body (Student) = {<هنر، مینا رضایی، ۷۹۰۲> ,<کامپیوتر، آرش راد، ۷۸۰۱>,
<هنر، فرزانه رسولی، ۸۰۰۲> ,<هنر، علی راد، ۸۰۰۱>}

مجموعه عنوان[3]: مجموعه ویژگی های یک جدول را مجموعه عنوان آن نامند.

Header (student) = {st#, name, course}

نکته: طبق تعریف مجموعه در ریاضیات، مجموعه عنوان، فاقد نظم است بنابراین، student (st#, name, course) یا student (st#, course, name)، هر دو تعریف یک رابطه به نام student است.

درجه یک رابطه[4]: تعداد ویژگی های یک جدول را درجه آن جدول یا رابطه نامند. مثلاً درجه جدول student، ۳ است.

کاردینالیتی یک رابطه[5]: تعداد تاپل های یک جدول در هر لحظه از حیات آن را کاردینالیتی آن جدول یا رابطه نامند. کاردینالیتی یک رابطه، مرتباً در حال تغییر است چون دائماً تاپل هایی از جدول حذف و یا تاپل‌هایی در آن درج می‌شوند.

واژه شناسی بانک های اطلاعاتی:

نظریه پرداز	تحلیل گر	تولید کننده
رابطه (Relation)	موجودیت (Entity)	جدول (Table)
صفت (Attribute)	صفت (Attribute)	ستون (Column)
تاپل (Tuple)	ردیف (Row)	سطر / رکورد (Row / Record)

[1] Tuple
[2] Body
[3] Header
[4] Relation Degree
[5] Relation Cardinality

در جدول فوق، واژه های مورد استفاده هر دسته از افراد (نظریه پردازان، تحلیل گران و تولید کنندگان) که از نظر معنایی معادل می‌باشند، بیان شده است؛ مثلاً نظریه پردازان از اصطلاح رابطه استفاده می‌کنند درحالیکه تولید کنندگان از واژه جدول.

وابستگی تابعی[1]: در یک جدول، ویژگی B به ویژگی A وابستگی تابعی دارد اگر به ازاء هر مقدار برای ویژگی A، حداکثر یک مقدار برای ویژگی B وجود داشته باشد. این مفهوم را به صورت $A \rightarrow B$ نشان می‌دهیم. در این صورت، A تعیین کننده B نامیده می‌شود.

مثال: فرض کنید جدول employee که مربوط به اطلاعات کارمندان یک شرکت است، شامل ویژگی های زیر باشد:

Employee (emp#, name, family, ID, birthDate, birthPlace, address, SSN)

emp#: شماره کارمندی (هر کارمند، یک شماره منحصر به فرد دارد)

name: نام کارمند

family: نام خانوادگی کارمند

ID: شماره شناسنامه

birthDate: تاریخ تولد

birthPlace: محل تولد

address: آدرس محل سکونت

SSN: شماره ملی کارمند

در این جدول، روابط زیر صحیح هستند:

$emp\# \rightarrow name$

این رابطه صحیح است چون با داشتن یک شماره کارمندی تنها به یک نام می رسیم.

$emp\# + family \rightarrow ID$

این رابطه صحیح است چون با داشتن یک شماره کارمندی و یک نام خانوادگی تنها به یک شماره شناسنامه می رسیم. در واقع، شماره کارمندی به تنهایی ما را به یک شماره شناسنامه واحد می رساند.

$SSN \rightarrow emp\#,\ name,\ family,\ ID,\ \ldots$

این رابطه صحیح است چون هر شماره ملی نه تنها یک کارمند واحد در سازمان بلکه تنها یک شخص را در کل کشور مشخص می کند. بنابراین با داشتن یک شماره ملی، تنها به یک شماره کارمندی، تنها به یک نام، تنها به یک نام خانوادگی، تنها به یک شماره شناسنامه و ... می رسیم.

$ID + birthPlace + birthDate \rightarrow emp\#,\ name,\ family,\ldots$

این رابطه صحیح است چون ترکیب شماره شناسنامه، محل تولد و تاریخ تولد، تنها یک شخص را در کل کشور و مسلماً تنها یکی از کارمندان سازمان را مشخص می کند.

در جدول employee، روابط زیر نادرست می‌باشند:

$name + family \rightarrow ID$

این رابطه صحیح نیست چون با داشتن یک نام و نام خانوادگی، لزوماً تنها به یک شماره شناسنامه نمی‌رسیم.

$ID + birthPlace \rightarrow name$

[1] Functional Dependence

این رابطه صحیح نیست چون ممکن است چندین کارمند در سازمان وجود داشته باشند که محل تولد و شماره شناسنامه آنها یکسان باشد.

سوپر کلید[1]:

سوپر کلید مجموعه ای از یک یا چند ویژگی است که سایر ویژگی های جدول به آن وابستگی تابعی دارند.

مثال: چند نمونه از سوپر کلیدهای جدول employee عبارتند از:

emp#
emp#+name
emp#+family

ID+birthPlace+birthDate
ID+birthPlace+birthDate+emp#
ID+birthPlace+birthDate+name

SSN
SSN+emp#
SSN+name

وابستگی تابعی کامل[2]:

در یک جدول، ویژگی B به ویژگی A وابستگی تابعی کامل دارد ($A \xrightarrow{FFD} B$) اگر ویژگی B به کل ویژگی A وابستگی تابعی داشته باشد. به عبارت دیگر، ویژگی B به ویژگی A وابستگی تابعی کامل دارد اگر اولاً ویژگی B به ویژگی A وابستگی تابعی داشته باشد و ثانیاً به هیچ جزئی از آن وابستگی تابعی نداشته باشد.

مثال: در جدول employee روابط زیر صادقند:

$emp\# \xrightarrow{FFD} name,\ family,\ ID,...$

این رابطه صحیح است چون ویژگی های name و family و ID و ... به #emp وابستگی تابعی دارند و از آنجا که #emp تنها یک جزء دارد، بنابراین وابستگی جزئی نمی تواند وجود داشته باشد و وابستگی، کامل است.

$SSN \xrightarrow{FFD} name,\ family,\ ID,...$

این رابطه صحیح است چون ویژگی های name و family و ID و ... به SSN وابستگی تابعی دارند و چون SSN تنها یک جزء دارد، بدون شک این وابستگی از نوع کامل است و نیازی به بررسی ندارد.

$ID + birthPlace + birthDate \xrightarrow{FFD} emp\#,\ name,\ family,...$

این رابطه صحیح است چون ویژگی های #emp و name و family و ... به ترکیب ID+birthPlace+birthDate وابستگی تابعی دارند ولی به هیچ جزئی از آن (مثل ID یا ID+birthPlace یا birthPlace+birthDate) وابستگی تابعی ندارند، پس این وابستگی از نوع کامل است.

در جدول employee، روابط زیر صادق نیستند:

[1] Super Key
[2] Full Functional Dependence

$emp\#+name \xrightarrow{FFD} family$

این رابطه درست نیست زیرا:

$emp\# \rightarrow family$

به عبارت دیگر، family به #emp که قسمتی از emp#+name است، وابستگی تابعی دارد. پس این وابستگی تابعی، وابستگی تابعی جزئی است و کامل نیست.

$ID + birthPlace \xrightarrow{FFD} name$

این رابطه درست نیست چرا که اصولاً name به ID+birthPlace وابستگی تابعی ندارد پس مسلماً وابستگی تابعی کامل هم ندارد.

کلید کاندیدا[1]:

کلید کاندیدا سوپر کلیدی است که قابل خلاصه شدن نباشد.

و یا به عبارتی دیگر:

سوپر کلیدی است که هیچ جزئی از آن سوپر کلید نباشد.

ویا به عبارتی دیگر:

مجموعه ای از یک یا چند ویژگی است که سایر ویژگی های جدول به آن وابستگی تابعی کامل دارند.

مثال: کلیدهای کاندیدا برای جدول employee عبارتند از:

کلید کاندیدای ۱: #emp

کلید کاندیدای ۲: SSN

کلید کاندیدای ۳: Id+birthPlace+birthDate

کلید اصلی[2]:

طراح پایگاه داده ها با توجه به شرایط حاکم بر سیستم یکی از کلیدهای کاندیدا را به عنوان کلید اصلی انتخاب می کند. کلید اصلی، مجموعه ای از یک یا چند ویژگی است که برای شناسایی و تمایز میان موجودیت‌های یک دسته مورد استفاده قرار می گیرد.

مثال: در جدول employee سه کلید کاندیدا وجود دارد. کلید کاندیدای سوم یعنی Id+birthPlace+birthDate بسیار طولانی است و برای شناسایی کارمندان یک شرکت مناسب نیست. کلید کاندیدای دوم یعنی SSN در حال حاضر نمی تواند وسیله مناسبی برای شناسایی کارمندان یک شرکت باشد چون ممکن است بسیاری از آنها هنوز برای دریافت شماره ملی اقدام نکرده باشند. مناسب‌ترین کلید اصلی در این حالت #emp است چرا که خود شرکت می تواند به هر کارمند یک شماره کارمندی مناسب و منحصر به فرد دهد.

کلید ثانویه[3]:

کلید ثانویه مجموعه ای از یک یا چند ویژگی است که در صورت عدم دسترسی به مقدار کلید اصلی، از مقدار آن برای تسریع جستجوی اطلاعات یک موجودیت خاص استفاده می‌شود. اگر چه ممکن است مقادیر این ویژگی یک موجودیت منحصر به فرد را مشخص نکنند، ولی در یافتن آن کمک می‌کنند.

[1] Candidate Key
[2] Primary Key
[3] Secondary Key

مثال: در جدول employee موارد زیر می‌توانند به عنوان کلید ثانویه معرفی شوند:

name+family
SSN
ID+birthPlace+birthDate

کلید خارجی[1]: اگر ویژگی A بین جدول ۱ و جدول ۲ مشترک و در جدول ۱ کلید اصلی باشد، آنگاه ویژگی A در جدول ۲، کلید خارجی نسبت به جدول ۱ خواهد بود.

مثال: فرض کنید پایگاه داده یک دانشگاه شامل جداول زیر باشد:

city (city#, cityName)

city#: کد شهر (به هر شهر، یک کد منحصر به فرد نسبت داده شده است)

cityName: نام شهر

st (st#, sname, city#, field)

st#: شماره دانشجویی

sname: نام دانشجو

city#: کد شهری که دانشجو در آن ساکن است

field: رشته تحصیلی دانشجو (در این دانشگاه، هر دانشجو تنها در یک رشته تحصیل می کند)

course (crs#, cname, unit)

crs#: شماره درس (به هر درس، یک شماره منحصر به فرد داده شده است)

cname: نام درس

unit: تعداد واحد درس

enroll (st#, crs#, year, term, grade)

st#: شماره دانشجویی

crs#: شماره درس اخذ شده توسط دانشجو

year: سال اخذ درس توسط دانشجو

term: نیم سال اخذ درس توسط دانشجو (نیم سال اول یا دوم یا تابستان)

grade: نمره اخذ شده

برای درک بهتر مسأله، فرض کنید اطلاعات وارد شده در جداول به شرح زیر باشند:

city

city#	cityName
۰۰۱	تهران
۰۰۲	اصفهان
۰۰۳	شیراز
۰۰۴	رشت

st

st#	sname	city#	field
۷۸۰۱	آرش راد	۰۰۱	هنر
۷۸۰۹	علی رضایی	۰۰۱	کامپیوتر
۸۰۰۱	عسل رسولی	۰۰۲	کامپیوتر

[1] Foreign Key

enroll

st#	crs#	year	term	grade
۷۸۰۱	۱۴۰۰	۸۰	۱	۹،۵
۷۸۰۱	۱۴۰۰	۸۰	۲	۱۲
۷۸۰۱	۱۲۰۰	۸۰	۱	۲۰
۷۸۰۹	۱۴۰۰	۸۰	۱	۱۵
۷۸۰۹	۱۴۰۵	۸۱	۲	۱۸
۸۰۰۱	۱۴۰۰	۸۱	۱	۱۶

course

crs#	cname	unit
۱۴۰۰	پایگاه داده	۳
۱۴۰۵	مهندسی اینترنت	۳
۱۲۰۰	ادبیات فارسی	۳

کلیدهای اصلی:

در جدول city: #city

در جدول st: #st

در جدول course: #crs

در جدول enroll: st# + crs# + year + term

کلیدهای خارجی:

از آنجا که ویژگی #city بین دو جدول city و st مشترک و در جدول city کلید اصلی است، بنابراین ویژگی #city در جدول st کلید خارجی نسبت به جدول city است.

از آنجا که ویژگی #st بین دو جدول st و enroll مشترک و در جدول st کلید اصلی است، پس در جدول enroll کلید خارجی نسبت به جدول st است.

از آنجا که ویژگی #crs بین دو جدول course و enroll مشترک و در جدول course کلید اصلی است، پس در جدول enroll کلید خارجی نسبت به جدول course است.

۲-۲ قواعد جامعیت در مدل رابطه‌ای

در این قسمت به قواعد جامعیت (محدودیت‌های جامعیتی) در مدل رابطه‌ای به طور اجمال می‌پردازیم.

۲-۲-۱ تعریف

جامعیت پایگاه داده‌ها یعنی: صحت، دقت و سازگاری داده‌های ذخیره شده در پایگاه در تمام لحظات.

هر سیستم مدیریت پایگاه داده‌ها باید بتواند جامعیت پایگاه داده‌ها را کنترل و تضمین کند، زیرا همیشه ممکن است عواملی سبب نقض جامعیت شوند.

برخی از عوامل عبارتند از:

- اشتباه در برنامه‌های کاربردی
- اشتباه در وارد کردن داده‌ها
- وجود افزونگی کنترل نشده
- توارد تراکنشها به گونه‌ای که داده نامعتبر ایجاد شود
- خرابی‌های سخت‌افزاری و نرم‌افزاری

این عوامل، به طور مستقیم یا غیر مستقیم شرایطی را پدید می‌آورند که نهایتاً منجر به نقض جامعیت داده‌ها می‌شوند.

نکته: برای کنترل و تضمین جامعیت، قواعدی[1] لازم است تا سیستم مدیریت بتواند بر اساس آن‌ها عمل کند. به این قواعد، گاه محدودیت هم گفته می‌شود. این قواعد ماهیتاً از نوع قواعد سمانتیک[2] خرد جهان واقع هستند و هم از این رو از اهمیت خاصی برخوردارند زیرا اعمال این قواعد یا محدودیت‌ها سبب می‌شود تا محتوای پایگاه داده‌ها با واقعیات خرد جهان واقع انطباق داشته باشد.

۲-۲-۲ انواع قواعد جامعیت

قواعد جامعیت در مدل رابطه‌ای به دو رده کلی تقسیم می‌شوند:

- قواعد کاربری[3] (قواعد خاص)
- فراقواعد[4] (قواعد عام)

قواعد کاربری

قواعدی هستند که توسط کاربر تعریف می‌شوند. این قواعد وابسته به داده‌های خرد جهان واقع هستند، به این معنا که در مورد یک پایگاه داده‌های خاص (مورد نظر کاربر) مطرح می‌شوند و عمومیت ندارند. گاه به این قواعد، قواعد محیطی یا وابسته به داده و یا محدودیت‌های جامعیت معنایی[5] می‌گویند.

سیستم مدیریت پایگاه داده‌ها باید به کاربر امکان دهد تا قواعد جامعیت کاربری را تعریف کند و به سیستم بدهد. مشخص کردن قواعد یا محدودیت‌های جامعیتی یکی از وظایف مدل‌ساز و طراح پایگاه داده‌ها است. بخشی از این محدودیت‌ها در مرحله مدل‌سازی قابل اعمال هستند و بخش دیگر باید در مرحله طراحی منطقی پایگاه داده‌ها منظور شوند. این قواعد در اساس به دو صورت اعلانی[6] و رویه ذخیره شده[7] به سیستم داده می‌شوند. در حالت اعلانی، از احکام DDL استفاده می‌شود و قواعد، بخشی از شمای پایگاه داده‌ها هستند و ممکن است تا ۹۰ درصد تعریف پایگاه را تشکیل دهند. در حالت رویه ذخیره شده، از احکام DML یا زبان توصیف محدودیت[8] استفاده می‌شود و تعدادی رهانا[9] (راه‌انداز) نوشته و به سیستم داده می‌شود. روشن است که در حالت اعلانی دیگر لازم نیست که برنامه‌سازان برنامه‌های کاربردی (کاربردار)، خود به نوشتن رویه‌های ذخیره شده یا رهاناها بپردازد و در نتیجه میزان کار آن‌ها و نیز حجم برنامه‌های کاربردی کاهش می‌یابد.

یکی از مکانیسم‌ها برای اعمال قواعد جامعیت کاربری، استفاده از «رهانا» است. رهانا، قاعده یا قواعدی (به صورت تعدادی دستور) است که در پی بروز تغییراتی در پایگاه داده‌ها، باید به طور خودکار اعمال شوند.

برای طراحی یک رهانا باید رویداد، شرط یا شرایطی که در آن‌ها رهانا باید اجرا شود و نیز اقدامی که باید انجام شود، را مشخص کنیم. به رهانا، گاه قاعده فعال هم می‌گویند.

---

[1] Integrity Rule
[2] Semantic Rule
[3] User Defined Rule
[4] Meta Rule
[5] Semantic Integrity Constraint
[6] Declarative
[7] Stored Procedure
[8] Constraint Specification Language
[9] Trigger

بدیهی است که تعداد قواعد کاربری بستگی به محیطی دارد که پایگاه داده‌ها برای آن ایجاد می‌شود. توجه داشته باشید که مجموعه قواعد کاربری یک محیط عملیاتی باید رسماً به تأیید مدیر داده‌های سازمان (DA) برسد و سپس مدیر پایگاه داده‌ها آن‌ها را در مراحل طراحی و پیاده‌سازی پایگاه داده‌ها، منظور نماید. هر چه تعداد قواعد جامعیت کاربری بیشتر باشد، فزونکاری در سیستم برای اعمال آن‌ها بیشتر است.

توجه داریم که اگر یک قاعده جامعیتی جدید به سیستم داده شود، سیستم باید ابتدا مطمئن شود که این قاعده با وضعیت جاری پایگاه داده‌ها همخوانی دارد. اگر چنین نباشد، سیستم باید آن قاعده را رد کند. در غیر این صورت، قاعده پذیرفته می‌شود (در کاتالوگ سیستم وارد می‌شود) و از این لحظه به بعد، اعمال می‌شود.

و اما قواعد کاربری در مدل رابطه‌ای خود بر چهار دسته‌اند:

- **قاعده میدانی:** قاعده‌ای است ناظر به یک میدان و مقادیر مجاز آن را مشخص می‌کند، مثلاً مقادیر میدان GRADE (نمره) اعداد از صفر تا بیست است.
- **قاعده صفتی (ستونی):** قاعده‌ای است ناظر به یک صفت (ستون) و بیان‌کننده نوع آن صفت است، مثلاً صفت STID (شماره دانشجویی) از نوع کاراکتر است.
- **قاعده رابطه‌ای:** قاعده‌ای است ناظر به یک رابطه و مقادیر مجاز یک متغیر رابطه‌ای را مشخص می‌کند. مثلاً در رابطه COT (درس‌ها) درس عملی از گروه آموزشی D111 و D444 نمی‌تواند بیش از یک واحد داشته باشد.
- **قاعده پایگاهی:** قاعده‌ای است ناظر به دو یا بیش از دو متغیر رابطه‌ای که به نحوی با یکدیگر مرتبط هستند. مثلاً در رابطه‌های COT (درس‌ها) و PROF (اساتید) و STCOPR (رابطه میان درس و استاد) محدودیت زیر می‌تواند وجود داشته باشد:

  استاد با مرتبه دانشیار به بالا نباید درسی از دوره کاردانی را تدریس کند. (فرض می‌کنیم که در رابطه COT صفت COLVL به معنای سطح درس را هم داریم).

نکته: قاعده میدانی طبعاً در مورد صفت (صفاتی) که از یک میدان مقدار می‌گیرند، اعمال می‌شود. اما قاعده صفتی، قاعده‌ای است که قبل از هر چیز، محدودیتی است که نوع صفت را مشخص می‌کند.

مثال:

قاعده ۱: شماره دانشجو به صورت dddddddd است که در آن دو رقم اول عددی بزرگتر از ۶۵ است.

قاعده ۲: مقادیر میدان STDEGR: 'bs'، 'ms' و 'doc' است.

قاعده ۳: دانشجوی با معدل کمتر از ۱۲ در یک ترم، نمی‌تواند در ترم بعد بیش از ۱۴ واحد انتخاب کند.

قاعده ۴: مدیر گروه آموزشی حداکثر می‌تواند دو دوره دو ساله، مدیر گروه باشد.

قاعده ۵: هر درس حتماً باید یک منبع اصلی و دو منبع فرعی داشته باشد.

قاعده ۶: رعایت پیش نیاز (ها) در انتخاب درس، الزامی است.

قاعده ۷: در میدان مقادیر GRADE، نمره «ناتمام» وجود ندارد.

قاعده ۸: دانشجوی دوره کارشناسی باید ۱۴۲ واحد بگذراند تا فارغ التحصیل شود.
قاعده ۹: دانشجو نمی‌تواند درس آزمایشگاه را حذف کند.
قاعده ۱۰: تعداد دانشجویان درس‌های تخصصی در هر گروه نباید بیش از ۳۰ نفر باشد.
قاعده ۱۱: حداقل نمره قبولی برای دانشجوی دوره کارشناسی ارشد ۱۲ است.
قاعده ۱۲: تعداد واحدهای اخذ شده دانشجوی فعال نمی‌تواند «هیچمقدار» باشد.

فراقواعد

قواعدی هستند که باید توسط هر سیستم رابطه‌ای در هر پایگاه داده‌های رابطه‌ای اعمال شوند، ناوابسته به داده‌های خاص هستند و عمومیت دارند. این قواعد که به آن‌ها فرامحدودیت[1] نیز می‌گویند عبارتند از:

- **قاعده C_1:** قاعده جامعیت موجودیتی[2]
- **قاعده C_2:** قاعده جامعیت ارجاعی[3]

قاعده C_1: ناظر است به کلید اصلی و چنین است:
هیچ جزء تشکیل دهنده کلید اصلی نمی‌تواند هیچمقدار داشته باشد.
هیچمقدار عبارت است از هر مقدار ناشناخته، غیر قابل اعمال، تعریف نشده و در هر صورت همان مفهوم «اطلاع نهست». مثلاً در رابطه STCOT، در ابتدا و در طول ترم، هنوز نمره دانشجو در درس مشخص نیست.
دلیل توجیه کننده قاعده C_1 این است که هر مقدار کلید اصلی، شناسه یک تاپل است در رابطه و تاپل، خود نشان‌دهنده یک نمونه موجودیت در معنای عام. پس هر مقدار کلید اصلی، شناسه یک نمونه مشخص از یک نوع موجودیت است و عامل تمییز آن نمونه موجودیت از هر نمونه دیگر. عامل تمییز خود نمی‌تواند ناشناخته (ناموجود) باشد.
با توجه به محدودیت C_1 می‌توان گفت که در پایگاه داده‌های رابطه‌ای، هیچ‌گاه، شیئی را که نتوانیم بشناسیم، ذخیره نمی‌کنیم. عامل شناسایی هم کلید اصلی است. برای اعمال قاعده C_1، باید در تعریف رابطه، کلید اصلی آن را به سیستم معرفی کرد.
توجه داشته باشیم که اگر قاعده C_1 در یک سیستم پیاده‌سازی شده باشد، همان معرفی کلید اصلی آن برای اعمال این قاعده کفایت می‌کند و نیازی به تصریح NOTNULL برای صفت کلید اصلی نیست. ضمناً هر صفتی که هیچمقدار نپذیرد لزوماً کلید کاندید و بالطبع کلید اصلی نیست. پس از معرفی کلید اصلی، سیستم همه عملیات تاپلی را بر اساس کلید اصلی انجام می‌دهد، مثلاً درخواست درج تاپلی که کلید اصلی در آن «هیچمقدار» باشد را رد می‌کند و یا بازیابی تک تاپل حتماً از طریق مقدار کلید اصلی امکان پذیر می‌شود. اساساً همه عملیات تاپلی (و نه مجموعه ای) باید از طریق یک کلید کاندید انجام شود و طبعاً اولویت با همین کلید اصلی است.

[1] Meta – constraint
[2] Entity Integrity Rule
[3] Referential Integrity Rule

مشکلات هیچمقدار

مفهوم هیچمقدار توسط کاد در مدل رابطه‌ای وارد شده است و تأکید بر این است که این مفهوم، یک «مقدار» است. با این برداشت از این مفهوم، پیاده‌سازی آن در سیستم‌های رابطه‌ای مشکلاتی دارد، از آن میان:

۱- *نمایش این مقدار در پایگاه داده‌ها و بروز حافظه هرز (مصرف بیهوده حافظه).* توجه داریم که هیچمقدار، صفر یا بلانک نیست. برای این نمایش، طرح مشخصی لازم است. یکی از طرح‌های پیاده‌سازی شده در بعضی سیستم‌های رابطه‌ای از جمله سیستم DB/2، طرحی است موسوم به تکنیک متابایت[1]. در این تکنیک سیستم برای هر صفت رابطه (هر ستون جدول) یک بایت دیگر در نظر می‌گیرد که کاربر آن را نمی‌بیند. اگر مقدار صفتی در تاپلی از رابطه «هیچمقدار» باشد، سیستم در متابایت آن صفت از تاپل، مقدار صفر را می‌گذارد و اگر مقدار صفت، معلوم باشد، یک می‌گذارد.

Ai	متابایت
V	1
NV	0
V	1
V	1
NV	0

V: مقدار معلوم NV: مقدار همیچمقدار

۲- اگر هیچمقدار یک «مقدار» است، پس باید به عنوان عملوند[2] در عملیات محاسباتی و منطقی و نیز به عنوان قیاسوند[3] در عملیات مقایسه‌ای دخالت داده شود. در این صورت، باید قواعد عملیاتی مشخصی وجود داشته باشد. اگر هیچمقدار را با 'NV' نشان دهیم، حاصل اجرای هر یک از عملیات زیر، باز هم هیچمقدار است[4]:

NV + NV
NV - NV
NV × NV
NV ÷ NV
NV + CONST
NV - CONST
NV × CONST
NV ÷ CONST

در عمل مقایسه:

NV > NV ?
NV = NV ?

---

[1] Meta Byte
[2] Operand
[3] Comparand
[4] DATE C.J. "An Introduction to Database Systems". 5th ed. Addison Wesley. USA 1990.

NV < NV ?
NV <= NV ?
NV >= NV ?

نتیجه مقایسه ناشناخته است.

در عملیات منطقی، منطق دو ارزشی کافی نیست، به منطق سه ارزشی نیاز است، به صورت زیر:

AND	F	T	NV
F	F	F	F
T	F	T	NV
NV	F	NV	NV

OR	T	F	NV
F	F	T	NV
T	T	T	T
NV	NV	T	NV

NOT	
F	T
T	F
NV	NV

برای سایر عملکردهای منطقی نیز می‌توان جدول ارزش را مشخص کرد.

می‌بینیم که اگر در برنامه کاربردی، از توابع محاسباتی (مثل توابع SUM و...) استفاده شود، ممکن است اجرای آن‌ها با دشواری و یا فزونکاری در سیستم همراه باشد.

نکته: دکتر Codd در کتابش، همان‌طور که اشاره شد، بر «مقدار» بودن این مفهوم تأکید دارد. اما Date (که در حیطه دانش و تکنولوژی سیستم‌های رابطه‌ای، مرجع مسلم است) پس از سال‌ها دفاع از نظر کاد در این مورد، بالاخره به این نتیجه رسید که مفهوم هیچمقدار در مدل رابطه‌ای لازم نیست و حتی اعلام کرد که این مفهوم «مخرب» مدل رابطه‌ای است و به جای آن باید همان مفهوم کلاسیک «مقدار پیش نهاده[1]» را به کار برد تا نشان دهنده «اطلاع نهست» باشد. با این نظر، دیگر نیازی به تکنیک خاص برای نمایش هیچمقدار، شبیه تکنیک متابایت نیست. کافی است «مقدار پیش نهاده» به طور مناسب، در سیستم انتخاب شود.

سایر مؤلفین فقط مفهوم هیچمقدار را مطرح کرده‌اند و به بحث‌های نظری در این مورد نپرداخته‌اند.

نکته: محدودیت هیچمقدارناپذیری، فقط ناظر به کلید اصلی نیست بلکه طراح می‌تواند این محدودیت را در مورد هر صفت دیگر رابطه نیز اعمال کند و در این صورت، جزء قواعد کاربری محسوب می‌شود.

قاعده C2: ناظر است به کلید خارجی و چنین است:

اگر صفت خاصه A_i (ساده یا مرکب) در رابطه R_2 کلید خارجی باشد، در این صورت: A_i در R_2 می‌تواند هیچمقدار داشته باشد یا اینکه باید حتماً مقداری باشد که در رابطه مرجع (R_1) وجود دارد. به عبارت دیگر، مقدار کلید خارجی یک رابطه نمی‌تواند در رابطه مرجع وجود نداشته باشد.

[1] Default Value

دلیل توجیه کننده قاعده C_2 این است که کلید خارجی، عامل ارجاع است از یک نمونه موجودیت (از تاپل (هایی) در یک رابطه) به نمونه موجودیت دیگر (تاپلی از رابطه (هایی) دیگر یا همان رابطه) و نمی‌توان به نمونه موجودیت ناموجود ارجاع داد.

مثال:

برای اعمال قاعده C_2 در عملیات روی پایگاه داده‌ها، سیستم باید در هر عمل، متناسب با قاعده و طبق نظر طراح و مسئول پایگاه عمل کند:

- در عمل درج تاپل در رابطه رجوع کننده، سیستم باید وارسی کند آیا تاپل مرجع آن در رابطه (های) مرجع وجود دارد. اگر نه، درخواست رد می‌شود.
- در عمل حذف تاپل مرجع: پنج روش برای اعمال قاعده C_2 متصور است:

روش ۱: روش حذف تسلسلی[1] (منتشر شونده)

در این روش، با حذف تاپلی از رابطه مرجع، تمام تاپل‌های رجوع کننده به‌آن در رابطه (های) رجوع کننده، حذف می شوند.

روش ۲: روش حذف تعویقی[2] (مشروط)

در این روش، درخواست حذف تاپل مرجع، تا زمانی که تاپل (هایی) رجوع کننده به آن در رابطه (های) رجوع کننده، وجود داشته باشند، معوق می ماند. در واقع حذف به شرطی انجام می‌شود که تاپل رجوع کننده وجود نداشته باشد.

روش ۳: روش هیچقدار گذاری[3]

در این روش، با حذف تاپل مرجع، کلید خارجی در تاپل رجوع کننده، هیچمقدارگذاری می‌شود البته به شرط آن‌که کلید خارجی در رابطه رجوع کننده، جزء کلید اصلی رابطه نباشد (این روش دیگر مورد تأیید DATE نیست و در آخرین اثرش [DATE 2000] مطرح نشده است).

روش ۴: عدم اقدام[4]

در این روش، فقط همان عمل درخواست شده انجام می‌شود و اقدام دیگری صورت نمی گیرد.

روش ۵: مقدار گذاری با مقدار پیش نهاده

در این روش، با حذف تاپل مرجع، کلید خارجی در تاپل (های) رجوع کننده به آن با مقدار پیش نهاده، مقدارگذاری می‌شود.

طراح پایگاه داده‌ها باید در شِمای پایگاه، نظر خود را در عمل بهنگام سازی و عمل حذف با انتخاب گزیدار مناسب، به سیستم اعلام کند:

```
FOREIGN   KEY (Attribute(s)) REFERENCES Relation – name
      DELETE    Option
      UPDATE    Option
```

در اینجا Option یکی از پنج گزیدار زیر است:

---

[1] Cascade
[2] Restricted
[3] Nullifying
[4] No Action

- CASCADE
- RESTRICTED
- NULLIFIES
- NO ACTION
- SET TO DEFAULT

مثال:

در این مثال، رابطه را به صورتیکه در سیستم‌های رابطه‌ای موجود رایج است، تعریف می‌کنیم.

```
CREATE      TABLE     STCOT
(STID       CHAR(8)   NOTNULL ,
COID        CHAR(6)   NOTNULL ,
TR          CHAR(1),
YRYR        CHAR(5),
GRADE       DECIMAL(2,2)
PRIMARY     KEY (STID, COID)
FOREIGN     KEY (STID) REFERENCES STT
   DELETE   CASCADE
   UPDATE   CASCADE
FOREIGN     KEY (COID) REFERENCES COT
   DELETE   CASCADE
   UPDATE   CASCADE;
```

۲-۲-۳ راه‌های اعمال قواعد (محدودیت‌های) جامعیت

۱- معرفی کلید اصلی

۲- اعلام هیچمقدارناپذیری صفت

۳- معرفی کلید خارجی و گزیدارهای نشان‌دهنده طرز رفتار سیستم

۴- اعلان محدودیت‌های مورد نظر، در شِمای پایگاه داده‌ها

۵- نوشتن رهانا (راه‌انداز)

در اینجا کوتاهانه یادآور می‌شویم که رهانا مکانیسمی است برای راه اندازی اجرای یک عمل در پی اجرای یک عمل دیگر. به بیان دیگر، در صورت بروز یک رویداد، عملی باید انجام شود تا سازگاری پایگاه داده‌ها تأمین گردد. مثلاً در پی انجام یک عمل بهنگام‌سازی در تاپلی از یک رابطه، تاپل (هایی) از رابطه‌ای (هایی) دیگر هم بهنگام درمی‌آیند؛ یا در صورت حذف تاپلی از یک رابطه، تاپل (هایی) از رابطه‌ای (هایی) دیگر نیز حذف می شوند (مثلاً برای رعایت قاعده C_2).

۶- معرفی میدان و مقادیر آن.

۷- معرفی وابستگی های تابعی بین صفات به سیستم (که معمولاً به طور ضمنی از طریق طراحی رابطه‌های نرمال‌تر با توجه به این وابستگی‌ها، انجام می‌شود).

۲-۳ تمرین حل شده

فرض کنید سیستم پایگاه داده ها در یک دانشگاه شامل جداول زیر باشد:

field (field#, fieldName)

در این جدول اطلاعات مربوط به رشته های تحصیلی ذخیره می‌شود:

field#: کد رشته تحصیلی (برای هر رشته تحصیلی یک کد منحصر به فرد در نظر گرفته شده است)

fieldName: نام رشته تحصیلی

type (type#, typeName, fee)

در این جدول اطلاعات مربوط به نوع دروس ذخیره می‌شود:

type#: کد نوع درس (به هر نوع درس یک کد منحصر به فرد داده شده است)

typeName: نوع درس

fee: قیمت هر واحد

student (st#, sname, startYear, field#)

در این جدول اطلاعات مربوط به دانشجویان ذخیره می‌شود:

st#: شماره دانشجویی

sname: نام دانشجو

startYear: سال ورود به دانشگاه

field#: کد رشته تحصیلی دانشجو

course (crs#, cname, unit, type#)

در این جدول اطلاعات دروس ذخیره می‌شود:

crs#: شماره درس

cname: نام درس

unit: تعداد واحد درس

type#: کد نوع درس (نظری، عملی، ...)

CF (crs#, field#, kind)

در این جدول مشخص می‌شود هر درس مربوط به کدام رشته های تحصیلی است:

crs#: شماره درس

field#: کد رشته تحصیلی

kind: این ویژگی مشخص می کند درس مورد نظر برای رشته مورد نظر چه حالتی دارد ('P' برای پیش‌نیاز، 'A' برای پایه، 'T' برای تخصصی، 'O' برای عمومی و 'E' برای اختیاری)

grades (st#, crs#, term, grade)

در این جدول اطلاعات مربوط به نمرات دانشجویان ذخیره می‌شود:

st#: شماره دانشجویی

crs#: شماره درس

term: نیم سال اخذ درس توسط دانشجو (مثلاً ۸۳۱ به معنای ترم اول سال ۸۳ است)

grade: نمره اخذ شده

pre (crs#, pre#)

در این جدول پیش نیازهای هر درس مشخص می‌شوند:

#crs: شماره درس

#pre: شماره درس پیش نیاز

prof (prof#, pname, degree)

در این جدول اطلاعات مربوط به اساتید ذخیره می‌شود:

#prof: شماره استاد (به هر استاد یک شماره منحصر به فرد داده شده است)

pname: نام استاد

degree: آخرین مدرک تحصیلی استاد ('D' برای دکترا، 'F' برای فوق لیسانس و 'L' برای لیسانس)

PC (prof#, crs#, term)

در این جدول مشخص می‌شود هر استاد در هر نیم سال چه دروسی را تدریس کرده است:

#prof: شماره استاد

#crs: شماره درس

term: نیم سال تحصیلی

tuition (field#, startYear, constTuition)

در این جدول بر اساس سال ورود به دانشگاه، شهریه ثابت هر رشته تحصیلی مشخص می‌شود:

#field: کد رشته تحصیلی

startYear: سال ورود به دانشگاه

constTuition: شهریه ثابت

برای درک بهتر مسأله به نمونه هایی از اطلاعات ذخیره شده در پایگاه داده توجه کنید:

field

field#	fieldName
1	مهندسی کامپیوتر
2	مهندسی الکترونیک
3	ریاضی محض

type

type#	typeName	fee
1	نظری	5000
2	عملی	20000
3	آزمایشگاه	30000

tution

field#	startYear	constTuition
1	80	75000
1	81	80000
1	82	90000
2	81	80000
2	82	90000
3	82	75000

course

crs#	cname	unit	type#
1100	ادبیات	2	1
1105	تربیت بدنی	1	2
1400	برنامه سازی ۱	3	1
1402	برنامه سازی ۲	3	1
1403	ذخیره و بازیابی	3	1
1407	پایگاه داده	3	1
1500	ریاضی ۱	3	1
1600	زبان پیش نیاز	2	1

student

st#	sname	startYear	field#
8001	آرش راد	80	1
8002	عسل شاملو	80	3
8003	سیاوش آزاد	80	1
8101	ساغر راد	81	1
8102	علی نیکی	81	1
8111	فرامرز نیکی	81	1
8112	علی رضایی	81	2

prof

prof#	pname	degree
101	فرانک شایسته	L
102	علی پیامی	F
106	آزاده نیکوکار	F
107	سیامک فرزانه	D
108	علی نادرنژاد	F

grades

st#	crs#	term	grade
8001	1400	811	9
8001	1400	812	13
8001	1401	811	13
8101	1400	821	19
8101	1500	821	14
8111	1400	811	12
8111	1401	811	20

PC

prof#	crs#	term
101	1400	801
101	1400	802
101	1401	801
102	1400	801
108	1500	811

pre

crs#	pre#
1400	1500
1402	1400
1407	1402
1407	1403

CF

crs#	field#	kind
1100	1	O
1100	2	O
1100	3	O
1105	1	O
1105	2	O
1105	3	O
1400	1	T
1400	2	E
1400	3	E
1402	1	T
1403	1	T
1407	1	T

1500	1	P
1500	2	P
1500	3	P

الف) صحت روابط زیر را در جدول field بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$field\# \longrightarrow fieldName$

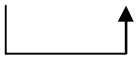
این رابطه صحیح است چون با داشتن یک کد رشته، تنها به یک نام رشته می رسیم.

$field\# \xrightarrow{FFD} fieldName$

این رابطه صحیح است چون با داشتن یک کد رشته، تنها به یک نام رشته می رسیم و #field قابل خلاصه شدن نیست.

نمودار وابستگی:

field (<u>field#</u>, fieldName)



تذکر: در نمودارهای وابستگی، برای مشخص کردن کلید اصلی از خط زیر استفاده می‌شود.

ب) صحت روابط زیر را در جدول type بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$type\# \xrightarrow{FFD} typeName,\ fee$

این رابطه صحیح است چون با داشتن یک کد نوع درس، تنها به یک نام نوع درس و تنها به یک قیمت واحد می رسیم.

$type\#+typeName \xrightarrow{FFD} fee$

این رابطه صحیح نیست چون $type\# \rightarrow fee$ صحت دارد. پس طرف چپ قابل خلاصه شدن است.

$type\# \xrightarrow{FFD} typeName,\ fee$

این رابطه صحیح است چون $type\# \rightarrow typeName,\ fee$ و از طرف دیگر، #type قابل خلاصه شدن نیست.

نمودار وابستگی:

type (<u>type#</u>, typeName, fee)

ج) صحت روابط زیر را در جدول tuition بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$field \rightarrow constTuition$

این رابطه صحیح نیست چون با داشتن یک کد رشته، لزوماً به یک شهریه ثابت واحد نمی رسیم، چون نرخ شهریه هر رشته برای ورودی های مختلف، متفاوت است. مثلاً برای کد رشته کامپیوتر ممکن است چندین نرخ شهریه وجود داشته باشد.

$startYear \rightarrow constTuition$

این رابطه صحیح نیست چون با داشتن یک سال ورود لزوماً به یک شهریه ثابت واحد نمی رسیم چون نرخ شهریه برای ورودی های هر سال در رشته های مختلف، متفاوت است. مثلاً برای ورودی های سال ۸۳ ممکن است چندین نرخ شهریه وجود داشته باشد.

$field\#+startYear \rightarrow constTuition$

این رابطه صحیح است چون با داشتن یک کد رشته و سال ورود، لزوماً به یک شهریه ثابت واحد می رسیم.

$field\#+startYear \xrightarrow{FFD} constTuition$

این رابطه صحیح است چون $field\#+startYear \rightarrow constTuition$ صحیح است و از طرف دیگر، field#+startYera، قابل خلاصه شدن نیست.

نمودار وابستگی:

tuition (<u>field#</u>, <u>startYear</u>, constTuition)

د) صحت روابط زیر را در جدول st بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$sname + stsrtYear \rightarrow field\#$

این رابطه صحیح نیست چون با داشتن یک نام دانشجو و سال ورود، لزوماً به کد رشته واحد نمی رسیم. مثلاً ممکن است دو دانشجوی ورودی ۸۰ با نام علی راد در دانشگاه وجود داشته باشند که یکی در رشته هنر و دیگری در رشته ریاضی تحصیل می کند.

$st\# \rightarrow sname, startYear, field\#$

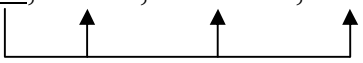
این رابطه صحیح است چون با داشتن یک شماره دانشجویی، تنها به یک نام، تنها به یک سال ورود و تنها به یک کد رشته می رسیم.

$st\# \xrightarrow{FFD} sname, startYear, field\#$

این رابطه صحیح است چون $st\# \rightarrow sname, startYear, field\#$ صحت دارد و از طرف دیگر، st# قابل خلاصه شدن نیست.

نمودار وابستگی:

student (<u>st#</u>, sname, startYear, field#)



ر) صحت روابط زیر را در جدول course بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$crs\# \rightarrow cname, unit, type\#$

این رابطه صحیح است چون با داشتن یک شماره درس، تنها به یک نام درس، تنها به یک تعداد واحد و تنها به یک کد نوع درس می رسیم.

$crs\#+cname \rightarrow unit, type\#$

این رابطه صحیح است چون با داشتن یک شماره درس و نام درس، تنها به یک تعداد واحد و تنها به یک کد نوع درس می رسیم.

$crs\# \xrightarrow{FFD} cname, unit, type\#$

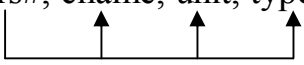
این رابطه صحیح است چون $crs\# \rightarrow cname, unit, type\#$ برقرار است و از طرف دیگر، crs# قابل خلاصه شدن نیست.

$crs\#+cname \xrightarrow{FFD} unit, type\#$

این رابطه صحیح نیست چون $crs\# \rightarrow unit, type\#$ صحیح است. پس سمت چپ قابل خلاصه شدن است.

نمودار وابستگی:

course (crs#, cname, unit, type#)



ز) صحت روابط زیر را در جدول CF بررسی کرده، نمودار وابستگی این جدول را رسم کنید.

$crs\# \rightarrow field\#$

این رابطه صحیح نیست چون با داشتن یک شماره درس، لزوماً تنها به یک کد رشته نمی رسیم. مثلاً ممکن است درس ریاضی ۱، هم مربوط به رشته ریاضی و هم مربوط به رشته کامپیوتر باشد.

$crs\# \rightarrow kind$

این رابطه صحیح نیست چون با داشتن یک شماره درس، لزوماً تنها به یک حالت درس نمی رسیم. مثلاً ممکن است درس برنامه سازی ۲، برای رشته کامپیوتر تخصصی و برای رشته ریاضی، اختیاری باشد.

$field\# \rightarrow kind$

این رابطه صحیح نیست چون با داشتن یک کد رشته، لزوماً تنها به یک حالت درس نمی رسیم چون مسلماً درس های زیادی در یک رشته وجود دارند که هر کدام از آنها، حالت خاص خود را دارد. مثلاً در رشته کامپیوتر هم دروس پایه و هم دروس تخصصی و هم دروس اختیاری وجود دارند.

$field\# \xrightarrow{FFD} kind$

این رابطه صحیح نیست چون رابطه $field\# \rightarrow kind$ صحیح نیست.

$crs\#+ field\# \rightarrow kind$

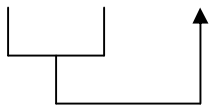
این رابطه صحیح است چون هر درس برای هر رشته، تنها یک حالت می تواند داشته باشد مثلاً درس برنامه‌سازی ۲، برای رشته کامپیوتر تنها تخصصی است.

$crs\#+ field\# \xrightarrow{FFD} kind$

این رابطه صحیح است چون اولاً $crs\#+ field\# \rightarrow kind$ برقرار است و ثانیاً crs#+field# قابل خلاصه شدن نیست.

نمودار وابستگی:

CF (<u>crs#</u>, <u>field#</u>, kind)



س) صحت روابط زیر را در جدول grades بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$st\# \rightarrow grade$

این رابطه صحیح نیست چون یک دانشجو ممکن است نمرات زیادی داشته باشد، پس داشتن یک شماره دانشجویی لزوماً ما را به یک نمره واحد نمی رساند.

$st\#+crs\# \rightarrow grade$

این رابطه صحیح نیست چون ممکن است یک دانشجو در یک درس چندین نمره داشته باشد. مثلاً ممکن است دانشجوی ۷۸۰۱ بار اول در درس ۱۴۰۰ نمره ۸ و بار دوم نمره ۱۱ گرفته باشد. پس با داشتن یک شماره دانشجویی و یک شماره درس، لزوماً به یک نمره واحد نمی رسیم.

$st\#+crs\# \rightarrow term$

این رابطه صحیح نیست چون ممکن است یک دانشجو یک درس را در چند ترم گرفته باشد، مثلاً ممکن است دانشجوی ۷۸۰۱ درس ۱۴۰۰ را هم در ترم ۸۲۱ و هم در ترم ۸۲۲ گرفته باشد. پس با داشتن یک شماره دانشجویی و یک شماره درس، لزوماً به یک ترم واحد نمی رسیم.

$st\#+crs\#+term \rightarrow grade$

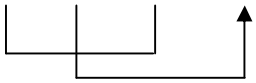
این رابطه صحیح است چون با داشتن یک شماره دانشجویی، یک شماره درس و یک ترم، تنها به یک نمره می رسیم؛ چون در هر درس در یک ترم برای هر دانشجو تنها یک نمره ثبت می‌شود.

$st\#+crs\#+term \xrightarrow{FFD} grade$

این رابطه صحیح است چون اولاً $st\#+crs\#+term \rightarrow grade$ صحیح است و ثانیاً طرف چپ، قابل خلاصه شدن نیست.

نمودار وابستگی:

grades (st#, crs#, term, grade)



ش) صحت روابط زیر را در جدول pre بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$crs\# \rightarrow preCrs\#$

این رابطه صحیح نیست چون با داشتن یک شماره درس، لزوماً به یک شماره درس پیش نیاز نمی رسیم. مثلاً ممکن است درس پایگاه داده بیش از یک پیش نیاز داشته باشد.

$preCrs\# \rightarrow crs\#$

این رابطه صحیح نیست چون با داشتن یک شماره درس پیش نیاز، لزوماً به یک شماره درس واحد نمی‌رسیم مثلاً ممکن است درس برنامه سازی ۱، پیش نیاز چند درس مختلف باشد.

$crs\#+preCrs\# \rightarrow crs\#, preCrs\#$

صحت این رابطه بدیهی است.

$crs\#+preCrs\# \xrightarrow{FFD} crs\#, preCrs\#$

این رابطه صحیح است چون اولاً $crs\#+preCrs\# \rightarrow crs\#, preCrs\#$ صحت دارد و ثانیاً سمت چپ، قابل خلاصه شدن نیست.

نمودار وابستگی:

pre (crs#, preCrs#)

و) صحت روابط زیر را در جدول prof بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$prof\# \rightarrow pname, degree$

این رابطه صحیح است چون با داشتن یک شماره استاد، تنها به یک نام استاد و تنها به یک آخرین مدرک تحصیلی استاد می رسیم.

$prof\# \rightarrow pname, degree$

این رابطه صحیح است چون اولاً $prof\# \rightarrow pname, degree$ برقرار است و ثانیاً سمت چپ رابطه، قابل خلاصه شدن نیست.

نمودار وابستگی:

prof (prof#, pname, degree)

هـ) صحت روابط زیر را در جدول PC بررسی کرده، نمودار وابستگی این جدول را رسم کنید:

$prof\# \rightarrow term$

این رابطه صحیح نیست چون با داشتن یک شماره استاد، لزوماً به یک ترم واحد نمی رسیم چون ممکن است یک استاد چندین ترم در دانشگاه تدریس کرده باشد.

$prof\#+crs\# \rightarrow term$

این رابطه صحیح نیست چون با داشتن یک شماره استاد و یک شماره درس، لزوماً به یک ترم واحد نمی‌رسیم چون ممکن است یک استاد، یک درس را چندین ترم در دانشگاه تدریس کرده باشد.

$prof\#+term \rightarrow crs\#$

این رابطه صحیح نیست چون با داشتن یک شماره استاد و یک ترم، لزوماً به یک شماره درس واحد نمی‌رسیم چون ممکن است یک استاد در طول یک ترم، چندین درس را تدریس کند.

$prof\#+crs\#+term \rightarrow prof\#, crs\#, term$

صحت این رابطه بدیهی است.

$prof\#+crs\#+term \xrightarrow{FFD} prof\#, crs\#, term$

این رابطه صحیح است چون اولاً $prof\#+crs\#+term \rightarrow prof\#, crs\#, term$ صحیح است و ثانیاً سمت چپ رابطه، قابل خلاصه شدن نیست.

نمودار وابستگی:

PC (prof#, crs#, term)

ی) کلیدهای خارجی کلیه جداول را مشخص کنید:

- جدول field:

کلید خارجی: ندارد

- جدول type:

کلید خارجی: ندارد

- جدول tuition:

کلید خارجی: #field نسبت به جدول field

- جدول student:

کلید خارجی: #field نسبت به جدول field

- جدول course:

کلید خارجی: #type نسبت به جدول type

- جدول CF:

کلیدهای خارجی: #crs نسبت به جدول course و #field نسبت به جدول field

- جدول grades:

کلیدهای خارجی: #st نسبت به جدول student و #crs نسبت به جدول course

- جدول pre:

کلیدهای خارجی: #crs نسبت به جدول course و #pre نسبت به جدول course (تشابه اسمی کلید اصلی و کلید خارجی اهمیتی ندارد)

- جدول prof:

کلید خارجی: ندارد

- جدول PC:

کلید خارجی: #prof نسبت به جدول prof و #crs نسبت به جدول course

۲-۴ تمرین فصل

فرض کنید پایگاه داده یک وب سایت اطلاع رسانی در مورد سینما از جداول زیر تشکیل شده باشد:

film (film#, fname, year, subject)

در این جدول اطلاعات فیلم ها ذخیره می‌شود:

film#: شماره فیلم

fname: نام فیلم

year: سال ساخت

subject: موضوع فیلم (اجتماعی، خانوادگی و ...)

people (ID, name, biography)

در این جدول مشخصات کلیه کارگردانان، بازیگران، نویسندگان و تهیه کنندگان، نگهداری می‌شود. به هر شخص یک شناسه منحصر به فرد داده شده است. یک شخص ممکن است مثلاً هم کارگردان و هم بازیگر و هم نویسنده باشد، ولی اطلاعات وی تنها یک بار ثبت می‌شود.

ID: شناسه شخص

name: نام

biography: زندگینامه شخص

film_director (film#, ID)

در این جدول مشخص می‌شود چه کسانی فیلم را کارگردانی کرده اند:

film#: شماره فیلم

ID: شماره شخصی که فیلم را کارگردانی کرده است (توجه کنید که یک فیلم ممکن است بیش از یک کارگردان داشته باشد)

film_writer (film#, ID)

در این جدول مشخص می‌شود چه کسانی فیلم فیلنامه فیلم را نوشته اند:

film#: شماره فیلم

ID: شماره شخصی که فیلمنامه را نوشته است (توجه کنید که یک فیلم ممکن است بیش از یک نویسنده داشته باشد)

film_actor (film#, ID, role)

در این جدول مشخص می‌شود چه کسانی در فیلم بازی کرده اند:

#film: شماره فیلم

ID: شماره شخصی که در فیلم بازی کرده است (توجه کنید که یک فیلم بیش از یک بازیگر دارد)

role: نوع نقش (نقش اول یا دوم یا ...)

film_producer (film#, ID)

در این جدول مشخص می‌شود چه کسانی فیلم را تهیه کرده اند:

#film: شماره فیلم

ID: شماره شخصی که فیلم را تهیه کرده است (توجه کنید که یک فیلم ممکن است بیش از یک تهیه‌کننده داشته باشد)

award (award#, aname)

در این جدول اطلاعات مربوط به انواع جایزه ها ذخیره می‌شود:

#award: کد جایزه (به هر نوع جایزه مانند بهترین کارگردانی، بهترین بازیگر نقش اول مرد و ... یک کد، اختصاص یافته است)

aname: نام جایزه

festival (fes#, fname, year, place)

در این جدول اطلاعات مربوط به جشنواره های مختلف ذخیره می‌شود:

#fes: شماره جشنواره

fname: نام جشنواره

year: سال برگزاری جشنواره

place: محل برگزاری جشنواره

fes_film (fes#, film#)

در این جدول مشخص می‌شود در هر جشنواره، چه فیلم هایی شرکت کرده اند. در هر جشنواره، فیلم های متعددی شرکت می‌کنند و هر فیلم ممکن است در جشنواره های مختلف شرکت کند.

#fes: شماره جشنواره

#film: شماره فیلم

grant (fes#, film#, award#, ID, kind, result)

در این جدول اطلاعات مربوط به مقام هایی که فیلم های مختلف در جشنواره های مختلف کسب کرده اند، ذخیره می‌شود. اطلاعات ذخیره شده باید نمایانگر اطلاعاتی به فرم زیر باشد:

در جشنواره جشن خانه سینما – دوره چهارم، سلیمه رنگزن برای بازی در فیلم عروس آتش، برنده جایزه بهترین بازیگر نقش دوم زن شد و دیپلم افتخار گرفت.

#fes: شماره جشنواره

#film: شماره فیلم

#award: کد نوع جایزه

ID: شماره شخصی که جایزه به وی اختصاص یافته است

kind: نوع جایزه (دیپلم افتخار، سیمرغ بلورین و ...)

result: نتیجه (برنده یا کاندیدا)

۱) برای هر جدول، کلیه وابستگی های تابعی و وابستگی های تابعی کامل را مشخص کنید.
۲) کلید اصلی هر جدول را مشخص کنید.
۳) برای هر جدول، کلیدهای خارجی را در صورت وجود، مشخص کنید.
۴) برای هر جدول، کلیدهای ثانویه را در صورت وجود، مشخص کنید.

فصل سوم

«زبان پرس و جوی ساخت یافته - SQL[1]»

همانگونه که قبلاً اشاره شد، هر کاربر یا برنامه کاربردی، برای ارتباط با پایگاه داده، از یک زبان فرعی داده‌ای استفاده می کند. SQL معروف ترین زبان فرعی داده ای است که توسط کلیه DBMSهای کنونی دنیا شامل Oracle, SQL Server, DB2, Informix و ... پشتیبانی می‌شود. به عبارت دیگر، برای نوشتن اغلب برنامه‌های کاربردی اعم از برنامه‌های حسابداری، حقوق و دستمزد، انبارداری، سایت های وب و ... که با پایگاه داده سر و کار دارند، فارغ از زبان برنامه‌نویسی و همچنین DBMS مورد استفاده، استفاده از زبان SQL الزامی است (البته بیشتر زبان های میزبان همچون Delphi, C#.NET و ...، درون خود دستوراتی برای کار با DBMS دارند ولی بهتر است از زبان SQL برای ارتباط با پایگاه داده استفاده شود).

۳-۱ تاریخچه زبان SQL

پس از آنکه دکتر Codd در سال ۱۹۷۰ مدل بانک اطلاعاتی رابطه ای را تعریف کرد، نظریات وی در زمینه تحقیق در باب بانک اطلاعاتی رابطه ای، غوغایی برپا نمود که از آن جمله پروژه تحقیقاتی در IBM بود. هدف پروژه که System/R نامیده می شد، این بود که ثابت کند مفهوم رابطه ای قابل اجرا بوده و نیز تجربیاتی عملی در زمینه ایجاد یک DBMS رابطه ای ارائه نماید. کار بر روی System/R در نیمه اول دهه ۱۹۷۰ در آزمایشگاه های IBM's Santa Teresa در San Jose کالیفرنیا آغاز گردید.

در ۱۹۷۴ و ۱۹۷۵ فاز اول پروژه System/R نمونه کوچکی از یک DBMS رابطه ای را تولید نمود. علاوه بر خود DBMS، پروژه System/R شامل کار بر روی زبان های پرسشی بانک اطلاعاتی نیز بود. یکی از این زبان ها به نام SEQUEL[2] بود. در ۱۹۷۶ و ۱۹۷۷ پروژه تحقیقاتی System/R بازنویسی گردید. در ساختار جدید پرسش های چند جدولی نیز پشتیبانی می‌شد و چندین کاربر می توانستند به طور اشتراکی از داده ها استفاده نمایند.

سیستم Syetem/R بین چندین مشتری IBM توزیع شد تا در ۱۹۷۸ و ۱۹۷۹ مورد ارزیابی قرار گیرد. این مشتریان مقداری تجربه عملی با System/R و زبان بانک اطلاعاتی آن که به SQL تغییر نام داده بود، کسب نمودند. علیرغم تغییر نام، تلفظ SEQUEL باقی ماند و تا امروز نیز ادامه دارد. در ۱۹۷۹ پروژه تحقیقاتی System/R به پایان رسید و IBM نتیجه گرفت که پیاده‌سازی بانک های اطلاعاتی رابطه ای نه تنها عملی است، بلکه می تواند پایه ای باشد برای یک محصول تجاری مفید. پروژه System/R و زبان بانک اطلاعاتی SQL آن، در مطبوعات فنی در دهه ۱۹۷۰ مورد توجه زیادی قرار گرفت. سمینارهایی در زمینه تکنولوژی بانک اطلاعاتی و مباحثاتی در مورد مزایای مدل رابطه ای جدید برگزار گردید. تا ۱۹۷۶ آشکار بود که IBM طرفدار جدی تکنولوژی بانک اطلاعاتی رابطه ای بوده، توجه زیادی نسبت به زبان SQL دارد.

---

[1] Structured Query Language
[2] Structured English Query Language

تبلیغات در زمینه System/R باعث جذب گروهی از مهندسین در Menlo Park در کالیفرنیا گردید، و این گروه به این نتیجه رسیدند که تحقیقات IBM منجر به یک بازار تجاری برای بانک های اطلاعاتی رابطه‌ای خواهد گردید. در ۱۹۷۷ این گروه شرکتی به نام Relational Software, Inc. تأسیس نمودند تا یک DBMS رابطه ای بر اساس SQL بسازند. محصولی به نام Oracle در ۱۹۷۹ عرضه گردید، و اولین DBMS رابطه ای به وجود آمد. به این ترتیب محصول Oracle باعث گردید اولین محصول IBM برای مدت ۲ سال در بازار دچار رکود باشد. این محصول بر روی مینی کامپیوترهای Digital VAX اجرا می‌شد که خیلی از کامپیوترهای بزرگ IBM ارزان تر بودند. امروزه این شرکت با نام Oracle Corporation اولین فروشنده سیستم‌های مدیریت بانک اطلاعاتی رابطه ای است. استادان آزمایشگاه های کامپیوتر در دانشگاه برکلی کالیفرنیا نیز در دهه ۱۹۷۰ مشغول تحقیق در زمینه بانک های اطلاعاتی رابطه ای بودند (مانند تیم تحقیق IBM). گروه فوق نیز یک نمونه از DBMS رابطه ای ایجاد نمودند و سیستم خود را Ingres نام نهادند.

پروژه Ingres شامل یک زبان پرسش[1] بود به نام QUEL، اگر چه از SQL خیلی ساخت یافته تر بود، اما شباهت آن به زبان انگلیسی کمتر بود.

در حالیکه Oracle و Ingres برای ارائه محصولات تجاری در رقابت بودند، پروژه System/R شرکت IBM در تلاش بوده است که یک محصول تجاری با نام SQL/DS[2] عرضه نماید. IBM موجودیت SQL/DS را در ۱۹۸۱ اعلام، و در ۱۹۸۲ شروع به عرضه محصول خود نمود. در سال ۱۹۸۳، IBM یک نسخه از SQL/DS را برای VM/CMS (سیستم عاملی که در کامپیوترهای بزرگ IBM غالباً استفاده شده بود)، اعلام نمود.

همچنین در سال ۱۹۸۳ شرکت IBM، محصول Database2 یا DB2 را معرفی نمود که یک DBMS رابطه ای برای سیستم‌های بزرگ آن شرکت بود. DB2 تحت سیستم عامل IBM's VMS (سیستم عامل مراکز کامپیوتری بزرگ) اجرا می‌شود. اولین نسخه DB2 در ۱۹۸۵ عرضه گردید و مسئولین IBM اعلام نمودند که این محصول یک برنامه استراتژیک برای تکنولوژی نرم‌افزاری IBM می‌باشد. DB2 از آن تاریخ تاکنون DBMS رابطه ای شاخص بوده و IBM از آن حمایت نموده و زبان DB2's SQL استاندارد عملی زبان بانک اطلاعاتی بوده است.

دستورات SQL همانند هر زبان فرعی داده ای دیگر به سه گروه تقسیم می‌شود:

۱- دستورات تعریف داده یا DDL
۲- دستورات دستکاری داده یا DML
۳- دستورات کنترل داده ها یا DCL

تذکر: کلیه دستورات SQL یک پرس وجو محسوب می‌شوند ولی برخی کارشناسان واژه پرس وجو را تنها در مورد دستور SELECT به کار می برند.

[1] Query Language
[2] SQL / Data System

تذکر: برای برخی از پرس و جو های این فصل چند روش ارائه شده است ولی این به این معنای ارائه کلیه روشهای ممکن نیست.

تذکر: در همه مثال های این فصل، از جداول S, P, J و SPJ استفاده شده است که در جدول S، اطلاعات تولیدکنندگان، در جدول P، اطلاعات محصولات و در جدول J، اطلاعات پروژه ها ذخیره شده است و ضمناً در جدول SPJ مشخص می‌شود هر تولید کننده چند کیلوگرم از هر محصول را برای هر پروژه تولید کرده است.

S (s#, sname, city)
P (p#, pname, color)
J (j#, jname, city)
SPJ (s#, p#, j#, qty)

S

s#	sname	city
S1	تهران مصالح	تهران
S2	یزد مصالح	یزد
S3	البرز	اصفهان
S4	ایران مصالح	تهران

P

p#	pname	color
P1	تیرآهن	مشکی
P2	آرماتور	مشکی
P3	سیمان	طوسی
P4	آلومینیم	سفید

J

j#	jname	city
J1	مرمت آثار باستانی	شیراز
J2	مرمت آثار باستانی	اصفهان
J3	فرودگاه	تهران

SPJ

s#	p#	j#	qty
S1	P1	J1	12000
S1	P2	J1	20000
S1	P3	J1	3000
S1	P4	J1	8000
S1	P1	J2	10000
S1	P1	J3	9000
S2	P1	J1	2000
S2	P1	J3	5000
S2	P3	J1	8000
S3	P1	J1	9000
S3	P2	J3	9000
S3	P1	J3	4000

۳-۲ دستورات تعریف داده‌ها

از دستورات تعریف داده ها برای تعیین و یا تغییر ساختار داده ها استفاده می‌شود. این دستورات عبارتند از:

۳-۲-۱ دستور ایجاد پایگاه داده‌ها

قبل از ایجاد جداول لازم است یک پایگاه داده ایجاد شود و سپس جداول مورد نظر در داخل آن ساخته شوند.

قالب کلی:

create database نام پایگاه داده

مثال:

create databse Sale

۳-۲-۲ دستور ایجاد جداول

قالب کلی:

```
create table نام جدول (
        نام ویژگی ۱   نوع ویژگی ۱    [not null] [unique],
        نام ویژگی ۲   نوع ویژگی ۲    [not null] [unique],
        نام ویژگی ۳   نوع ویژگی ۳    [not null] [unique],
        …
        primary key (نام ویژگی های تشکیل دهنده کلید اصلی),
        foreign key (نام ویژگی های کلید خارجی ۱) references نام جدول مورد نظر,
        foreign key (نام ویژگی های کلید خارجی ۲) references نام جدول مورد نظر,
        …
        [check (شرط مورد نظر])
```

تذکر: در کلیه دستورات برای مشخص کردن قسمت های اختیاری از کروشه استفاده شده است.

معروف‌ترین انواع داده در SQL عبارتند از:

نوع داده ای	بازه
integer	اعداد صحیح
smallint	اعداد صحیح
decimal (p,q)	اعدادی با p رقم و q رقم اعشاری در سمت راست
float	اعدادی اعشاری با ممیز شناور
char (n)	رشته های کارامتری با طول n
varchar (n)	رشته های کارامتری با طول متغیر کوچکتر یا مساوی n
date	تاریخ با فرمت yyyymmdd
time	زمان با فرمت hhmmss

تذکر: برای عبارات فارسی از انواع next, nchar و ... استفاده می‌شود.

تذکر: در صورت استفاده از عبارات not null برای یک ویژگی، DBMS از درج تاپل هایی که در آنها برای ویژگی مورد نظر مقداری وارد نشده باشد جلوگیری می کند. در صورت استفاده از عبارت unique برای یک ویژگی، DBMS از درج تاپل هایی که در آنها برای ویژگی مورد نظر مقدار تکراری وارد شده باشد جلوگیری می کند.

مثال: جدول S را به گونه ای ایجاد کنید که کاربر مجاز نباشد هیچ تاپلی با sname خالی یا تکراری در این جدول درج کند:

```
create table S (s# char (2),
                sname char (30) not null unique,
                city nchar (20),
                primary key (s#))
```

مثال: جدول SPJ را به گونه ای ایجاد کنید که بازه مجاز برای qty اعداد بین ۵۰۰ تا ۲۰۰۰۰ باشد.

برای تعیین قوانین جامعیت داده ای از قسمت check استفاده می‌شود:

```
create table SPJ (  s# char (2),
                    p# char (2),
                    j# char (2),
                    qty integer,
                    primary key (s#, p#, j#),
                    foreign key (s#) references S,
                    foreign key (p#) references P,
                    foreign key (j#) references J
                    check (qty>500 and qty<20000))
```

۳-۲-۳ ایجاد ایندکس

ایندکس ها در واقع همان کلید های ثانویه هستند. از ایندکس ها برای تسریع جستجوی یک موجودیت خاص در صورت عدم دسترسی به مقدار کلید اصلی استفاده می‌شود.

قالب کلی:

create [unique] index نام ایندکس on نام جدول (نام ویژگی ها)

مثال: روی جدول S یک ایندکس بر اساس نام تولید کننده و با نام names ایجاد کنید.

```
create index names on S (sname)
```

تذکر: در صورت استفاده از عبارت unique در این دستور، DBMS از ورود اسم های تکراری در جدول S جلوگیری خواهد کرد.

۴-۲-۳ اضافه کردن یک ستون جدید به یک جدول

قالب کلی:

alter table نام جدول add نام ستون جدید مشخصات ستون جدید

مثال: در جدول S ستون جدیدی با نام tel برای درج شماره تلفن تولیدکنندگان اضافه کنید.

```
alter table S add tel char(10)
```

۳-۲-۵ تغییر مشخصات یک ستون از یک جدول

قالب کلی:

alter table نام جدول alter column نام ستون مشخصات جدید

مثال: در جدول S، طول ستون sname را از ۳۰ کاراکتر به ۲۰ کاراکتر تغییر دهید.

```
alter table S alter column sname nchar(20) not null unique
```

۳-۲-۶ حذف یک ستون از جدول

قالب کلی:

alter table نام جدول drop column نام ستون

مثال: ستون sname را از جدول S حذف کنید.

```
alter table S drop column sname
```

۳-۲-۷ حذف یک جدول

قالب کلی:

drop table نام جدول

مثال: جدول S را حذف کنید.

```
drop table S
```

۳-۲-۸ حذف یک ایندکس

قالب کلی:

drop index نام جدول.نام ایندکس

مثال: ایندکس names روی جدول S را حذف کنید.

```
drop index S.sname
```

۳-۳ دستورات دستکاری داده

از این دستورات برای تهیه گزارشات (خواندن تاپل ها)، درج تاپل ها، حذف تاپل ها و تغییر تاپل ها استفاده می‌شود. این دستورات عبارتند از:

۳-۳-۱ دستور انتخاب

این دستور یکی از پرکاربردترین دستورات SQL است. از این دستور برای انتخاب تاپل ها و ستون های مورد نظر از یک یا چند جدول استفاده می‌شود.

قالب کلی:

```
select [distinct] نام ستون ها و یا عبارات محاسباتی مورد نظر
from        نام جدول
[where      [شرط روی تاپل ها
[group by   [نام ستون هایی که قرار است لیست بر اساس آنها گروه بندی شود
[having     [شرط روی گروه ها
[order by   [نام ستون هایی که قرار است لیست بر اساس آنها مرتب شود
```

مثال: نام کلیه شهرهایی را بیابید که تولید کننده ای در آن ها قرار دارد.

برای این کار کافی است ستون city از جدول S را انتخاب کنیم.

```
select city
from s
```

ویا می توان نام جدول را نیز ذکر کرد:

```
select S.city
from S
```

خروجی این پرس‌وجو به شکل زیر خواهد بود:

City
تهران
یزد
اصفهان
تهران

مشاهده می کنید که در این جدول تاپل های تکراری وجود دارند. برای حذف تاپلهای تکراری کافی است از عبارت distinct استفاده کنیم:

```
select distinct city
from S
```

خروجی:

city
تهران
یزد
اصفهان

مثال: مشخصات کلیه تولید کنندگان را بیابید.

برای این کار کافی است تمام ستون های جدول S را انتخاب کنیم:

```
select s#,sname,city
from S
```

به جای ذکر نام تمام ستون های یک جدول می توان از * استفاده کرد (* به معنی تمام ستون های جدول است):

```
select *
from S
```

خروجی:

s#	sname	city
S1	تهران مصالح	تهران
S2	یزد مصالح	یزد
S3	البرز	اصفهان
S4	ایران مصالح	تهران

قسمت where : از قسمت where برای انتخاب تاپل هایی از جدول که شرط به خصوصی دارند استفاده می‌شود.

مثال: نام تولید کنندگان تهرانی را بیابید:

```
select sname
from S
where city='تهران'
```

خروجی:

Sname
تهران مصالح
ایران مصالح

عملگرهای in و not in: برای تست وجود یا عدم وجود یک مقدار داخل یک مجموعه استفاده می‌شود.

مثال: مشخصات فروشهای مربوط به محصولات 'p2' یا 'p3' یا 'p4' را بیابید.

روش اول:

```
select *
from SPJ
where p#='p2' or p#='p3' or p#='p4'
```

روش دوم: استفاده از عملگر in

```
select *
from SPJ
where p# in ('p2','p3','p4')
```

خروجی:

s#	p#	j#	qty
S1	P2	J1	20000
S1	P3	J1	3000
S1	P4	J1	8000
S2	P3	J1	8000
S3	P2	J3	9000

مثال: مشخصات فروش هایی را بیابید که مربوط به محصولات 'P2' و 'P3' و 'P4' نیستند.

روش اول:

```
select *
from SPJ
where p#<>'P2' and p#<>'P3' and p#<>'P4'
```

روش دوم: استفاده از عملگر not in

```
select *
from SPJ
where p# not in ('P2','P3','P4)
```

عملگر between: توسط این عملگر می توان بازه یک جستجو را مشخص کرد. در این حالت، تمام مقادیری که بین value1 و value2 قرار می گیرند، انتخاب می‌شوند.

مثال: مشخصات تولیدکنندگانی را بیابید که #s آنها بین 'S1' و 'S4' واقع شده اند.

```
select *
from S
where s# between 'S1' and 'S3'
```

خروجی:

s#	sname	city
S1	تهران مصالح	تهران
S2	یزد مصالح	یزد
S3	البرز	اصفهان

عملگر like: از عملگر like برای جستجوی یک عبارت داخل مقادیر یک ستون رشته ای استفاده می‌شود.

مثال: مشخصات تولید کنندگانی را بیابید که در نام آنها عبارت «مصالح» به کار رفته باشد.

برای این پرس وجو لازم است از عملگر like استفاده کنیم.

```
select *
from S
where sname like '% مصالح%'
```

خروجی:

s#	sname	city
S1	تهران مصالح	تهران
S2	یزد مصالح	یزد
S4	ایران مصالح	تهران

تذکر: در این پرس و جو چون محل قرار گرفتن عبارت «مصالح» مهم نبود، در دو طرف عبارت از ٪ استفاده کردیم.

چنانچه نام تولید کنندگانی که نام آنها با عبارت «مصالح» شروع می‌شود مورد نظر باشد، از دستور زیر استفاده می کنیم:

```
select *
from S
where sname like 'مصالح%'
```

و چنانچه نام تولید کنندگانی که نام آنها به عبارت «مصالح» ختم می‌شود مورد نظر باشد از دستور زیر استفاده می‌شود:

```
select *
from S
where snam like '%مصالح'
```

تابع sum: از تابع sum برای محاسبه مجموع مقادیر یک ستون استفاده می‌شود.

مثال: میزان کل فروش 'P1' را بیابید.

برای این کار کافی است مجموع مقادیر ستون qty را در فروش های مربوط به 'P1' محاسبه کنیم:

```
select sum(qty)
from SPJ
where p#= 'P1'
```

خروجی این پرس و جو به شکل زیر خواهد بود:

51000

تابع avg: از تابع agv برای محاسبه میانگین مقادیر یک ستون استفاده می‌شود.

مثال: میانگین فروش 'P1' را بیابید.

برای این کار کافی است میانگین مقادیر ستون qty را در فروش های مربوط به 'P1' محاسبه کنیم:

```
select avg(qty)
from SPJ
where P#='P1'
```

خروجی این پرس و جو به شکل زیر خواهد بود:

7285.71

تابع max: از تابع max برای به دست آوردن بزرگترین مقدار یک ستون استفاده می‌شود.

مثال: حداکثر فروش 'P1' تا این لحظه را به دست آورید.

برای این کار کافی است ماکزیمم مقادیر ستون qty را در فروش های مربوط به 'P1' محاسبه کنیم:

```
select max(qty)
from SPJ
where p#='P1'
```

خروجی این پرس و جو به شکل زیر خواهد بود:

1200

تابع min: از تابع min برای به دست آوردن کوچکترین مقدار یک ستون استفاده می‌شود.

مثال: کمترین میزان فروش 'P1' توسط 'S1' را به دست آورید.

برای این کار کافی است مینیمم مقادیر ستون qty را در فروش های مربوط به 'S1' و 'P1' محاسبه کنیم:

```
select min(qty)
from SPJ
where p#= 'P1' and s#='S1'
```

خروجی این پرس و جو به شکل زیر می‌باشد:

9000

تابع count: از تابع count برای محاسبه تعداد تاپلهای مورد نظر از یک جدول استفاده می‌شود.

مثال ۱: محصول 'P1' تا کنون چند بار فروخته شده است؟

برای این کار کافی است تعداد تاپلهایی از جدول SPJ که مربوط به 'P1' هستند را به دست آوریم:

```
select count(*)
from SPJ
where p#='P1'
```

خروجی این پرس و جو به شکل زیر خواهد بود:

7

مثال ۲: محصول 'P1' تا کنون توسط چند تولید کننده فروخته شده است؟

برای این کار کافی است تعداد تولید کنندگان غیر تکراری از جدول SPJ که 'P1' را فروخته اند به دست آوریم:

```
select count(distinct s#)
srome SPJ
where p#='P1'
```

خروجی این پرس و جو به شکل زیر خواهد بود:

3

قسمت group by: از قسمت group by برای گروه بندی لیست بر اساس مقادیر یک یا چند ستون استفاده می‌شود.

مثال ۱: لیستی از کد محصولات و میزان کل فروش هر یک از آنها تهیه کنید.

در اینجا sum(qty) برای هر یک از محصولات P1 و P2 و P3 و ... باید به طور جداگانه محاسبه شود. پس لازم است لیست را براساس p# گروه بندی کنیم:

```
select p#, sum(qty)
from SPJ
group by p#
```

خروجی این پرس و جو به شکل زیر خواهد بود:

P#	
P1	51000
P2	29000
P3	11000
P4	8000

تذکر: در صورت استفاده از group by، قسمت select باید شامل نام ویژگی هایی که لیست بر اساس آنها گروه بندی شده است و در صورت نیاز یکی از توابع count, min, max, avg, sum باشد. مثلاً در پرس و جوی بالا چون در قسمت group by از #p استفاده شده است، قسمت select حتماً باید شامل #p و در صورت لزوم یکی از توابع count, min, max, avg, sum باشد و نمی توان از هیچ ویژگی دیگری در این قسمت استفاده کرد.

مثال ۲: لیستی از کد محصولات، کد پروژه ها و میزان کل فروش محصول مورد نظر برای پروژه مورد نظر تهیه کنید.

در اینجا میزان کل فروش برای هر ترکیب شماره محصول / شماره پروژه بایستی به طور جداگانه محاسبه شود. پس لازم است لیست را بر اساس #p و #j گروه بندی کنیم:

```
select p#, j#, sum(qty)
from SPJ
group by p#,j#
```

خروجی این پرس و جو به شکل زیر خواهد بود:

P#	j#	
P1	J1	23000
P2	J1	20000
P3	J1	11000
P4	J1	8000
P1	J2	10000
P1	J3	18000
P2	J3	9000

قسمت having: از قسمت having برای انتخاب گروه هایی که شرط به خصوصی دارند استفاده می‌شود.

مثال: کد محصولاتی را بیابید که میزان کل فروش آنها بیش از ۲۰۰۰۰ کیلوگرم است.

در اینجا میزان کل فروش برای هر محصول بایستی به طورجداگانه محاسبه شده، سپس کد محصولاتی که میزان کل فروش آنها بیش از ۲۰۰۰۰ است استخراج شود:

```
select p#
from SPJ
group by p#
having sum(qty)>20000
```

خروجی این پرس و جو به شکل زیر خواهد بود:

P#
P1
P2

قسمت order by: از این قسمت برای مرتب کردن لیست بر اساس مقادیر یک یا چند ستون عددی، رشته ای و یا تاریخ از یک جدول به ترتیب صعودی (asc[1]) و یا نزولی (desc[2]) استفاده می‌شود.

مثال ۱: لیستی از مشخصات تولید کنندگان تهیه کنید به طوری که براساس ترتیب صعودی نام آنها مرتب باشد.

```
select *
from S
order by sname asc
```

تذکر: چون ترتیب مرتب سازی به طور پیش فرض صعودی است می توان عبارت asc را حذف کرد:

```
select *
from S
order by sname
```

خروجی این پرس و جو به شکل زیر خواهد بود:

s#	sname	city
S3	البرز	اصفهان
S4	ایران مصالح	تهران
S1	تهران مصالح	تهران
S2	یزد مصالح	یزد

مثال ۲: لیستی از مشخصات تولید کنندگان تهیه کنید به طوری که بر اساس ترتیب صعودی شهر و ترتیب نزولی نام مرتب باشد.

```
select *
from S
order by city asc, sname desc
```

خروجی این پرس وجو به شکل زیر خواهد بود:

S#	sname	city
S3	البرز	اصفهان
S1	تهران مصالح	تهران
S4	ایران مصالح	تهران
S2	یزد مصالح	یزد

تذکر: در اینجا لیست ابتدا بر اساس ویژگی اول یعنی city مرتب شده است. سپس در مورد تاپلهایی که ویژگی city در آنها مساوی است، ترتیب نزولی بر اساس ویژگی دوم یعنی Sname اعمال شده است.

عملگر as: از این عملگر برای تغییر نام یک ستون در لیست استفاده می‌شود.

مثال : لیستی از نام تولید کنندگان و شهر سکونت آنها تهیه کنید.

فرض کنید بخواهیم نام ستون sname را به supplierName تغییر دهیم:

[1] Ascending
[2] Descending

```
select sname as supplierName,city
from S
```

خروجی این پرس و جو به شکل زیر خواهد بود:

supplierName	City
تهران مصالح	تهران
یزد مصالح	یزد
البرز	اصفهان
ایران مصالح	تهران

پرس و جو روی چند جدول

در کلیه مثال های قبل اطلاعات مورد نیاز در پرس و جو تنها از یک جدول استخراج شدند ولی در بسیاری از موارد نتیجه پرس و جو را باید از چند جدول استخراج کرد.

مثال: لیستی از نام تولید کنندگان و نام محصولات تهیه کنید به طوری که تولید کننده مورد نظر محصول مورد نظر را فروخته باشد.

نام تولید کنندگان در جدول S و نام محصولات در جدول P قراردارد. همچنین در جدول SPJ مشخص می‌شود که چه تولید کننده ای چه محصولی را فروخته است. بنابراین کافی است جداول S و P و SPJ را با هم پیوند دهیم.

```
select distinct sname , pname
from S,SPJ,P
where SPJ.s#=S.s# and SPJ.p#=P.p#
```

Sname	Pname
تهران مصالح	تیر آهن
تهران مصالح	آرماتور
تهران مصالح	سیمان
تهران مصالح	آلومینم
یزد مصالح	تیر آهن
یزد مصالح	سیمان
البرز	تیر آهن
البرز	آرماتور

عملگرهای exists و not exists: از این عملگرها برای تست وجود یا عدم وجود تاپلهایی خاص در یک جدول استفاده می‌شود.

مثال: نام تولید کنندگانی را بیابید که فروشی داشته اند.

روش اول: این پرس و جو را می توان به این صورت تعبیر کرد: از جدول S نام تولید کنندگانی را پیدا کنید که در جدول فروش (SPJ) فروشی (تاپلی) برای آنها وجود دارد:

```
select sname
from S
```

```
where exists (select *
              from SPJ
              where SPJ.s#=S.s#)
```

روش دوم: با استفاده از پیوند دو جدول نیز می توان نتیجه این پرس و جو را به دست آورد:

```
select sname
from S,SPJ
where S.s#=SPJ.s#
```

خروجی این پرس وجو به شکل زیر خواهد بود:

Sname
تهران مصالح
یزد مصالح
البرز

عملگر union: از این عملگر برای به دست آوردن اجتماع دو جدول سازگار استفاده می‌شود.

مثال: نام شهرهایی را بیابید که تولید کننده ای در آنها قرار دارد **یا** پروژه ای در آنها در حال اجرا است.

نام شهرهایی که تولید کننده ای در آنها قرار دارد

U

نام شهرهایی که پروژه ای در آنها در حال اجرا است

```
(select city from S)
union
(select city from J)
```

عملگر except: از این عملگر برای تفریق دو جدول سازگار استفاده می‌شود.

مثال: نام شهرهایی را بیابید که تولید کننده ای در آنها قرار دارد **ولی** پروژه ای در آنها در حال اجرا نیست.

نام شهرهایی که تولید کننده ای در آنها قرار دارد

−

نام شهرهایی که پروژه ای در آنها در حال اجرا است

```
(select city from S)
except
(select city from J)
```

عملگر intersect: از این عملگر برای به دست آوردن اشتراک دو جدول سازگار استفاده می‌شود.

مثال: نام شهرهایی را بیابید که تولید کننده ای در آنها قرار دارد **و** پروژه ای در آنها در حال اجرا است.

نام شهرهایی که تولید کننده ای در آنها قرار دارد

∩

نام شهرهایی که پروژه ای در آنها در حال اجرا است

```
(select city from S)
intersect
```

(select city from J)

نکته: در هر سه دستور فوق، مقادیر تکراری حذف می‌شوند. برای نشان دادن مقادیر تکراری کافی است کلمه کلیدی all به هر یک اضافه شود یعنی union all یا except all یا intersect all.

۳-۳-۲ دستور ایجاد دید خارجی یا دیدگاه[1]

هر کاربر از دید خود به اطلاعات موجود در پایگاه داده نگاه می کند. مثلاً در یک دانشگاه، اطلاعات زیادی در مورد یک دانشجو وجود دارد. مسائل مالی یا وضعیت تأمین اجتماعی دانشجو به امور مالی مربوط می‌شود و نه به آموزش، پس هیچ لزومی ندارد که این داده ها را در دسترس کاربران قسمت آموزش قرار دهیم. بنابراین بهتر است برای کاربران قسمت آموزش، یک یا چند دیدگاه ایجاد کنیم تا بدون درگیر شدن با داده‌هایی که مربوط به حوزه عملیاتی آنها نیست، عملیات مورد نظر خود را انجام دهند.

دیدگاه در واقع یک جدول است که توسط طراح پایگاه داده ها طراحی می‌شود. DBMS وظیفه دارد پس از اعمال هر تغییر در داده های جداول، محتویات دیدگاه هایی را که روی جداول مورد نظر ساخته شده اند، اصلاح کند. بنابراین داده های موجود در دیدگاه همیشه به روز می‌باشند.

استفاده از دیدگاه ها باعث سهولت انجام بسیاری از پرس و جوها و در نتیجه تسریع تهیه بسیاری از گزارشات می‌شود.

قالب کلی دستور ایجاد دیدگاه:

create view نام دیدگاه (نام ستون های دیدگاه)
as
یک دستور انتخاب

مثال: دیدگاهی به نام PartSale ایجاد کنید که شامل کد محصولات و میزان کل فروش آنها باشد.

```
create view PartSale (p#, sum1)
as
select p#, sum(qty)
from SPJ
group by p#
```

خروجی این پرس و جو به شکل زیر خواهد بود:

PartSale

p#	Sum1
P1	51000
P2	29000
P3	11000
P4	8000

تذکر: نام ستون های دیدگاه به ترتیب با نام ستون های انتخاب شده در دستور select نظیر می‌شوند.

مثال: نام محصولاتی را بیابید که میزان کل فروش آنها بیشتر از ۲۰۰۰۰ کیلوگرم است.

با فرض وجود دیدگاه PartSale، تهیه این گزارش بسیار ساده خواهد بود:

```
select pname
```

[1] View

```
from PartSale, P
where sum1>20000 and ParstSale.p#=P.p#
```

خروجی این پرس و جو به شکل زیر خواهد بود:

pname
تیرآهن
آرماتور

تذکر: می توان دیدگاهی را روی یک دیدگاه دیگر ساخت.

مثال: دیدگاهی به نام PartSale2 ایجاد کنید که شامل نام محصولاتی باشد که میزان کل فروش آنها بیشتر از ۲۰۰۰۰ کیلوگرم است.

```
create view PartSale2 (pname)
as
select pname
from PartSale, P
where sum1>20000 and PartSale.p#=P.p#
```

خروجی این پرس و جو به شکل زیر خواهد بود:

PartSale2

Pname
تیرآهن
آرماتور

۳-۳-۳ حذف یک دیدگاه

قالب کلی:

```
drop view نام دیدگاه
```

مثال: دیدگاه PartSale را حذف کنید.

```
drop view PartSale
```

تذکر: با حذف دیدگاه PartSale، دیدگاه PartSale2 نیز که روی آن ساخته شده است به طور خودکار نابود می‌شود.

۳-۳-۴ درج یک تاپل

قالب کلی:

```
insert into نام جدول (نام ویژگی ها) values (مقادیر ویژگی ها)
```

مثال: در جدول مربوط به تولیدکنندگان، تولید کننده ای با کد 'S5' و نام 'بهین مصالح' اضافه کنید:

```
insert into S (s#, sname) values ('S5','بهین مصالح')
```

۳-۳-۵ اصلاح تاپل‌ها

قالب کلی:

```
update نام جدول set ۱ نام ویژگی = مقدار۱ ,۲ نام ویژگی = مقدار ۲ ,... [where شرط]
```

مثال: شهر کلیه تولیدکنندگان را به شیراز تغییر دهید.

update S set city = 'شیراز'

مثال: نام تولید کننده 'S1' را به 'بهین مصالح' و شهر وی را به 'شیراز' تغییر دهید.

update S set sname = 'بهین مصالح', city = 'شیراز' where s#='S1'

۳-۳-۶ حذف تاپل ها

قالب کلی:

delete from نام جدول [where شرط]

مثال: کلیه تولید کنندگان را حذف کنید.

delete from S

مثال: کلیه تولید کنندگان تهرانی را حذف کنید.

delete from S where city = 'تهران'

۳-۴ دستورات کنترل داده‌ها

از دستورات DCL برای کنترل دسترسی کاربران مختلف به داده های پایگاه داده استفاده می‌شود. DBA وظیفه دارد برای هر کاربر یا گروه کاربران، نام کاربری و کلمه عبوری تعیین کرده، مجوزهای آنها را برای DBMS تعریف کند. DBMS هر کاربر را با توجه به نام کاربری وی شناسایی کرده و عملیات وی را کنترل می کند. چنانچه عملی با مجوزهای کاربر همخوانی نداشته باشد، DBMS از انجام آن سرباز می زند. دستورات DCL عبارتند از:

۳-۴-۱ دستور واگذاری مجوز

DBA وظیفه دارد پس از ایجاد پایگاه داده با توجه به حوزه اختیارات هر کاربر، مجوزهای دسترسی وی را برای DBMS تعریف کند. برخی از کاربران این امکان را دارند که کلیه و یا قسمتی از مجوزهای خود را با سایر کاربران سهیم شوند. قالب کلی دستور واگذاری مجوز به شکل زیر است:

grant	select	خواندن همه ستون ها	[with grant option] نام کاربران to نام جدول on
	select (نام ستون ها)	خواندن ستون های مشخص شده	
	update	اصلاح همه ستون ها	
	update (نام ستون ها)	اصلاح ستون های مشخص شده	
	insert	درج تاپل	
	delete	حذف تاپل	
	alter	تغییر ساختار جداول	
	index	ایجاد ایندکس	
	all	کلیه گزینه های بالا	

چنانچه عبارت with grant option ذکر شود، کاربر یا کاربرانی که مجوزهایی را دریافت می‌کنند به نوبه خود می‌توانند این مجوزها را با دیگران سهیم شوند.

مثال: فرض کنید کاربری با نام کاربری ali بخواهد مجوز خواندن کلیه ستون ها و تغییر ستون sname و درج تاپل روی جدول S را با کاربری با نام کاربری arash سهیم شود و به وی اجازه دهد این مجوزها را در اختیار کاربران دیگر نیز قرار دهد. در این صورت ali با نام کاربری و کلمه عبور خود وارد سیستم شده، دستور زیر را صادر می کند:

grant select, update (sname), insert on S to arash with grant option

۳-۴-۲ دستور بازپس گیری مجوز

کاربرانی که مجوزهایی را در اختیار کاربران دیگر قرار داده اند، می‌توانند همه یا تعدادی از مجوزها را از آنها باز پس گیرند. فرض کنید کاربر A مجوزهایی را در اختیار کاربرB و کاربر B نیز این مجوزها را در اختیار کاربران C و D قرار داده باشد. در این صورت چنانچه کاربر A مجوزها را از کاربر B پس بگیرد، کاربران C و D نیز این مجوزها را از دست خواهند داد. قالب کلی دستور بازپس گیری مجوز به شکل زیر است:

revoke	select	خواندن همه ستون ها	نام کاربران from نام جدول on
	select (نام ستون ها)	خواندن ستون های مشخص شده	
	update	اصلاح همه ستون ها	
	update (نام ستون ها)	اصلاح ستون های مشخص شده	
	insert	درج تاپل	
	delete	حذف تاپل	
	alter	تغییر ساختار جداول	
	index	ایجاد ایندکس	
	all	کلیه گزینه های بالا	

مثال: فرض کنید کاربری با نام کاربری ali بخواهد مجوز تغییر ستون sname از جدول S را از کاربری به نام کاربری arash پس بگیرد. در این صورت ali با نام کاربری و کلمه عبور خود وارد سسیستم شده، دستور زیر را صادر می کند:

Revoke update(sname) on S from arash

۳-۵ تمرین‌های حل شده

Q1: کد تولید کنندگانی را بیابید که آرماتور فروخته اند.

```
1)  select s#
    from SPJ
    where pname='آرماتور' and P.p#=SPJ.p#
2)  select s#
    from S
    where exists (select *
                  from P
                  where pname='آرماتور' and exists (select *
                                                    from SPJ
                                                    where SPJ.s#=S.s#
                                                    and SPJ.p#=P.p#)
```

```
3)      select s#
        from SPJ
        where p# in (select p#
                     from P
                     where pname='آرماتور')
```

Q2: نام تولیدکنندگانی را بیابید که آرماتور فروخته اند.

```
select sname
from S,SPJ,P
where pname='آرماتور' and P.p#=SPJ.p# and SPJ.s#=S.s#
```

Q3: نام تولیدکنندگانی را بیابید که آرماتور نفروخته اند.

```
select sname
from S
where s# not in (select s#
                 from SPJ, P
                 where pname='آرماتور' and SPJ.p#=P.p#)
```

Q4: لیستی از کد تولیدکنندگان و تعداد محصولات فروخته شده توسط هر یک از آنها تهیه کنید.

```
select s#, count(distinct p#)
from SPJ
group by s#
```

Q5: کد تولیدکنندگانی را بیابید که سه محصول متفاوت را فروخته اند.

```
select s#
from SPJ
group by s#
having count(distinct p#)=3
```

Q6: کد تولیدکنندگانی را بیابید که کلیه محصولات را فروخته اند.

```
select s#
from SPJ
group by s#
having count(distinct p#)=(select count(*) from P)
```

Q7: نام پروژه هایی را بیابید که برای آنها بیش از ۲۰۰۰۰ کیلوگرم سیمان خریداری شده است.

```
select jname
from J
where j# in (select j#
             from SPJ, P
             where pname='سیمان' and P.p#=SPJ.p#
             group by j#
             having sum(qty)>20000)
```

Q8: نام پروژه هایی را بیابید که **تنها** تولیدکنندگان تهرانی در آنها همکاری داشته اند.

نام پروژه هایی که تولیدکنندگان تهرانی در آنها همکاری داشته اند

-

نام پروژه هایی که تولیدکنندگان غیرتهرانی در آنها همکاری داشته اند

```
(select jname
 from SPJ, J, S
 where S.city='تهران' and S.s#=SPJ.s# and SPJ.j#=J.j#)
except
(select jname
 from SPJ, J, S
 where S.city<>'تهران' and S.s#=SPJ.s# and SPJ.j#=J.j#)
```

Q9: نام پروژه هایی را بیابید که هیچ تولیدکننده غیر تهرانی در آنها همکاری نداشته است.

نام همه پروژه ها

-

نام پروژه هایی که تولیدکنندگان غیر تهرانی در آنها همکاری داشته اند

```
(select jname
 from J)
except
(select jname
 from SPJ, J, S
 where S.city<>'تهران' and S.s#=SPJ.s# and SPJ.j#=J.j#)
```

۳-۶ تمرین‌های فصل

۱) پایگاه داده سیستم کتابخانه یک دانشگاه از جداول زیر تشکیل شده است:

book (book#, bname, author, publisher, year, edit#, translator)

book#: شماره کتاب (به هر کتاب یک شماره منحصر به فرد داده شده است. مثلاً اگر در کتابخانه سه کپی از یک کتاب وجود داشته باشد، هر یک از آنها یک شماره مجزا خواهند داشت)

bname: نام کتاب

author: نام نویسنده

publisher: نام ناشر

year: سال انتشار کتاب

edit#: ویرایش

translator: نام مترجم

student (st#, sname, field)

st#: شماره دانشجو

sname: نام دانشجو

field: رشته تحصیلی دانشجو

loan (loan#, book#, st#, loanDate, returnDate)

loan#: شماره امانت (به ازای امانت هر کتاب، یک شماره امانت منحصر به فرد اختصاص داده می‌شود)

#book: شماره کتاب

#st: شماره دانشجو

loanDate: تاریخ امانت گرفتن کتاب

returnDate: تاریخ برگشت کتاب به کتابخانه

book

book#	bname	author	publisher	year	edit#	translator
1400	شبکه های کامپیوتری و اینترنت	Douglas E.Comer	دانشگاه علم و صنعت ایران	1380	1	دکتر احمد اکبری – دکتر ناصر مزینی
1403	An introduction to database systems	C.J.Date	Addison Wesley	2000	7	
1404	Database systems, design, implementation, and management	Rob, Core nel	Boyed and fraser	1995	2	

student

st#	sname	field
8001	آرش راد	کامپیوتر
8002	عسل شاملو	ریاضی
8003	سیاوش آزاد	کامپیوتر
8101	ساغر راد	کامپیوتر

loan

loan#	book#	st#	loanDate	returnDate
1	1400	8001	83/06/06	83/06/10
2	1403	8001	83/06/06	83/06/10
3	1404	8003	83/06/10	83/06/16
4	1400	8003	83/06/11	83/06/20

Q1: نام کلیه کتاب هایی را بیابید که توسط «علی راد» امانت گرفته شده اند.

Q2: کتاب شماره 1403 تاکنون چند بار از کتابخانه امانت گرفته شده است؟

Q3: نام کتاب هایی را بیابید که تاکنون بیش از ۲۰ بار امانت گرفته شده اند.

Q4: نام کتاب هایی را بیابید که توسط کلیه دانشجویان رشته کامپیوتر امانت گرفته شده اند.

Q5: نام دانشجویانی را بیابید که تاکنون هیچ کتابی را به امانت نگرفته اند.

Q6: دیدگاهی ایجاد کنید که شامل نام دانشجویان و تعداد کل کتاب های به امانت گرفته شده توسط هر یک از آنها باشد.

نکته: توجه داشته باشید که گروه بندی بر اساس نام دانشجو صحیح نمی‌باشد زیرا ممکن است در یک دانشگاه دو دانشجو با نام های یکسان وجود داشته باشند که یکی ۱۰ کتاب به امانت گرفته و دیگری ۱۵ کتاب. بهتر است ابتدا یک دید بر اساس گروه بندی طبق شماره دانشجویی ایجاد شود و سپس بر اساس آن، دید دیگری شامل نام دانشجویان ایجاد گردد.

Q7: نام کتاب های 'Database' را بیابید که نویسنده آنها، Coronel است.

۲) جداول بخش ۲-۳ (تمرین حل شده فصل قبل) را در نظر بگیرید.

پرس و جوهای زیر را به زبان SQL بنویسید.

Q8: تعداد اساتیدی را بیابید که دارای مدرک دکترا هستند.

Q9: نام دانشجویانی را بیابید که هم درس پایگاه داده و هم درس مهندسی اینترنت را گذرانده اند.

Q10: نام دانشجویانی را بیابید که درس پایگاه داده را گذرانده اند ولی درس مهندسی اینترنت را نگذرانده‌اند.

Q11: تعداد دروس تخصصی را که دانشجوی شماره ۸۰۰۱ گذرانده است، به دست آورید.

Q12: لیستی از شماره دانشجویان و تعداد کل واحدهای تخصصی گذرانده شده توسط هر یک از آنها تهیه کنید.

Q13: میانگین نمرات درس پایگاه داده در نیم سال اول ۸۰ را به دست آورید.

Q14: لیستی از نیم سال های تحصیلی و میانگین نمرات درس پایگاه داده در هر یک از آنها تهیه کنید.

Q15: میانگین کل نمرات دانشجویان رشته مهندسی کامپیوتر در نیم سال دوم ۸۳ را به دست آورید.

Q16: لیستی از نام رشته های تحصیلی و میانگین کل نمرات هر یک از این رشته ها در نیم سال دوم ۸۳ تهیه کنید.

Q17: لیستی از نام رشته های تحصیلی که میانگین کل نمرات دانشجویان آنها در نیم سال اول ۸۳ بیشتر از ۱۴ بوده است، تهیه کنید.

Q18: لیستی از شماره دانشجویانی که هم در نیم سال اول ۸۳ و هم در نیم سال دوم ۸۳ مشروط شده اند تهیه کنید.

Q19: کل هزینه ثبت نام دانشجوی شماره ۸۰۰۱ شامل شهریه ثابت و هزینه مربوط به واحدها در نیم سال اول سال ۸۲ را محاسبه کنید.

Q20: لیستی از نام دانشجویانی که بیش از نصف دروس رشته مهندسی کامپیوتر را گذرانده اند تهیه کنید.

Q21: لیستی از نام اساتید و تعداد کل واحدهای تدریس شده توسط هر یک از آنها در نیم سال اول ۸۳ تهیه کنید.

Q22: لیستی از نام اساتیدی که تاکنون بیش از چهار درس مختلف تدریس کرده اند، تهیه کنید.

Q23: معدل کل دانشجوی شماره ۸۰۰۲ را محاسبه کنید.

Q24: لیستی از نام دروس تخصصی رشته کامپیوتر که قیمت هر واحد آنها بیشتر از ۳۰۰۰۰ تومان است تهیه کنید.

Q25: نام اساتیدی را بیابید که کلیه دروس رشته کامپیوتر را تدریس کرده اند.

Q26: لیستی از نام دروس عملی تهیه کنید که دانشجوی ۸۰۰۱ آنها را نگذرانده است (دروسی که این دانشجو نگذرانده و یا در آنها نمره قبولی نگرفته است).

Q27: لیستی از نام دروسی که مربوط به هر دو رشته مهندسی کامپیوتر و ریاضی محض هستند را بیابید.

فصل چهارم

«نرمال‌سازی»

برای کاهش افزونگی و ناهنجاری‌های حذف، درج و اصلاح، لازم است جداول پایگاه داده، نرمال باشند. از نظر درجه نرمال بودن، جداول را می‌توان به ترتیب به ۶ گروه تقسیم کرد:

۱- جداول آنرمال (کمترین درجه نرمالی)

۲- جداول نرمال ۱ یا 1NF یا FNF[1]

۳- جداول نرمال ۲ یا 2NF یا SNF[2]

۴- جداول نرمال ۳ یا 3NF و جداول نرمال BCNF[3]

۵- جداول نرمال ۴

۶- جداول نرمال ۵ یا PJNF[4] (بالاترین درجه نرمالی)

۴-۱ جداول آنرمال

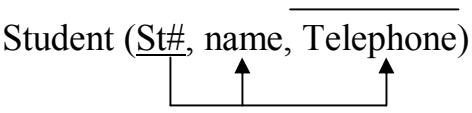
جدول Student را که شامل شماره دانشجویی، نام و شماره تلفن‌های تماس دانشجو است، در نظر بگیرید:

Student

St#	Name	Telephone
7801	آرش	0311-6262778 0913-311-5234
7802	علی	021-2956677 0912-314-4532

همانطور که مشاهده می‌کنید، در برخورد هر سطر جدول با ستون Telephone، این احتمال وجود دارد که به جای یک شماره تلفن، مجموعه‌ای از شماره تلفن‌ها یا به عبارت بهتر، یک گروه اطلاع تکرار شونده[5] وجود داشته باشد. بنابراین، جدول Student، آنرمال است.

جدول آنرمال، به جدولی اطلاق می‌شود که گروه اطلاع تکرارشونده دارد یا به عبارت دیگر، در برخورد هر سطر با هر ستون آن، به جای یک مقدار اتمی و تجزیه‌ناپذیر، مجموعه‌ای از مقادیر وجود دارد. در نمودار وابستگی، برای نمایش گروه اطلاع تکرارشونده از یک خط در بالای ویژگی استفاده می‌شود.



[1] First Normal Form
[2] Second Normal Form
[3] Boyce-Codd Normal Form
[4] Projection-Join Normal Form
[5] Repeating Group

۲-۴ جداول نرمال ۱

مهم‌ترین عیب یک جدول آنرمال این است که برای هر یک از عملیات درج، حذف و اضافه، به دو دسته عملگر احتیاج داریم: یکی در سطح تاپل و دیگری در سطح مجموعه. مثلاً در جدول Student اگر بخواهیم اطلاعات دانشجوی جدیدی را درج کنیم، به عملگر درج در سطح تاپل و اگر بخواهیم برای یک دانشجو، شماره تلفن جدیدی درج کنیم به عملگر درج در سطح مجموعه، نیاز خواهیم داشت. برای رفع این معایب، جداول آنرمال را به نرمال ۱ تبدیل می‌کنیم.

تذکر: اصولاً در مدل رابطه‌ای، یک جدول حداقل باید نرمال ۱ باشد.

یک جدول، در حالت نرمال ۱ است اگر هیچ گروه اطلاع تکرارشونده در آن وجود نداشته باشد و یا به عبارتی دیگر، در برخورد هر سطر با هر ستون جدول، به یک مقدار تجزیه‌ناپذیر برسیم.

مثال: برای آن‌که جدول آنرمال Student را به نرمال ۱ تبدیل کنیم، لازم است مقادیر ویژگی‌های St# و Name را به ازاء هر شماره تلفن، تکرار کنیم:

Student

St#	Name	Telephone
7801	آرش	0311-6262778
7801	آرش	0913-311-5234
7802	علی	021-2956677
7802	علی	0912-314-4532

با تبدیل جدول Student به این فرم، در برخورد هر سطر با ستون Telephone، تنها به یک شماره تلفن می‌رسیم ولی در این حالت، St# به تنهایی نمی‌تواند کلید اصلی باشد چون برای آن، مقادیر تکراری وجود دارند. برای رفع این مشکل، کلید اصلی را به St# + Telephone تغییر می‌دهیم.

مثال: جدول St را در نظر بگیرید:

کلید اصلی جدول، St# می‌باشد.

St

St#	Name	Course				
7801	علی	1400	پایگاه داده	3	20	79-2
		1500	ریاضی ۱	3	10	80-1
		1600	تجزیه و تحلیل	3	20	80-1
7902	آرش	1400	پایگاه داده	3	7	80-1
		1700	تربیت بدنی	1	20	80-1

برای تبدیل این جدول به حالت نرمال ۱، باید آن را به فرم زیر تبدیل کنیم:

کلید اصلی جدول، St# + Crs# + Term می‌باشد.

St

St#	Name	Crs#	Course	Unit	Grade	Term
7801	علی	1400	پایگاه داده	3	20	79-2
7801	علی	1500	ریاضی ۱	3	10	80-1
7801	علی	1600	تجزیه و تحلیل	3	20	80-1
7902	آرش	1400	پایگاه داده	3	7	80-1
7902	آرش	1700	تربیت بدنی	1	20	80-1

۳-۴ جداول نرمال ۲

برخی از جداول در حالت نرمال ۱ هستند ولی هنوز ناهنجاری دارند مثلاً با آن‌که جدول St به نرمال ۱ تبدیل شد، هنوز دارای ناهنجاری‌های زیر است:

۱- ناهنجاری در حذف:

فرض کنید علی تنها دانشجویی باشد که درس ریاضی ۱ را گرفته است. اگر لازم باشد اطلاعات علی را حذف کنیم، به طور ناخواسته، اطلاعات ریاضی ۱ را نیز از دست می‌دهیم.

۲- ناهنجاری در درج:

فرض کنید بخواهیم اطلاعات دانشجوی جدیدی را که هنوز هیچ درسی را نگرفته است درج کنیم، از آنجا که #Course و Term، جزئی از کلید اصلی هستند و نمی‌توانند مقدار تهی داشته باشند، انجام این عمل، ممکن نیست.

۳- ناهنجاری در اصلاح:

فرض کنید بخواهیم تعداد واحد درس پایگاه داده را از ۳ به ۴ تغییر دهیم. در این صورت، برای حفظ سازگاری داده‌ها لازم است این تغییر را به دفعات مکرر و در تاپل‌های مختلف اعمال کنیم (اصلاح منتشرشونده).

یک جدول، در حالت نرمال ۲ است اگر:

۱- نرمال ۱ باشد.

۲- در آن هیچ وابستگی جزئی به کلید اصلی وجود نداشته باشد. به عبارت دیگر، هیچ ویژگی جدول، تنها به قسمتی از کلید اصلی، وابستگی نداشته باشد.

مثال: جدول St که در مثال قبل تبدیل به نرمال ۱ شد، نرمال ۲ نیست چرا که کلید اصلی این جدول، St# + Crs# + Term است درحالی‌که، تنها با داشتن #St می‌توان به Name رسید و یا تنها با داشتن #Crs می‌توان به Cname و Unit رسید.

برای تبدیل یک جدول نرمال ۱ به نرمال ۲، مراحل زیر را دنبال کنید:

۱- کلیه ترکیبات ممکن میان اجزا کلید اصلی را به ترتیب در سطرهای مجزا بنویسید (اول ترکیبات یک‌جزئی، سپس ترکیبات دوجزئی، سپس ترکیبات سه‌جزئی و ...):

St#
Crs#

Term
St# + Crs#
St# + Term
Crs# + Term
St# + Crs# + Term

۲- در کنار هر یک از ردیف‌ها، ویژگی‌هایی را که در تشکیل کلید اصلی نقشی ندارند و به ویژگی‌های مورد نظر وابستگی تابعی دارند و در سطرهای بالاتر نیز نوشته نشده‌اند بنویسید. مثلاً با داشتن یک St#، تنها به یک Name می‌رسیم پس Name به St# وابستگی تابعی دارد. Name را در ردیف اول اضافه کرده و در ردیف‌های بعدی، آن را در نظر نمی‌گیریم:

St#, Name
Crs#, Cname, Unit
Term
St# + Crs#
St# + Term
Crs# + Term
St# + Crs# + Term, Grade

۳- کلیه ردیف‌هایی را که هیچ ویژگی غیرکلیدی در آن‌ها وجود ندارد و شامل هیچ اطلاعات مفیدی که در جداول دیگر نیز وجود ندارد نمی‌باشد، حذف کرده و سایر سطرها را به جدول مجزا تبدیل کنید.

Student (St#, Name)
Course (Crs#, Cname, Unit)
SC (St#, Crs#, Term, Grade)

به این ترتیب، جدول St به جداول زیر تبدیل خواهد شد:

Student

St#	Name
7801	علی
7902	آرش

Course

Crs#	Cname	Unit
1400	پایگاه داده	3
1500	ریاضی ۱	3
1600	تجزیه و تحلیل	3
1700	تربیت بدنی	1

SC

St#	Crs#	Term	Grade
7801	1400	79-2	20
7801	1500	80-1	10
7801	1600	80-1	20
7902	1400	80-1	7
7902	1700	80-1	20

۴-۴ جداول نرمال ۳ و BCNF

۴-۴-۱ جداول نرمال ۳

برخی از جداول، در حالت نرمال ۲ هستند ولی هنوز مشکلاتی دارند. به عنوان مثال، جدول Professor را که شامل شماره، نام، کد آخرین مدرک تحصیلی و نام آخرین مدرک تحصیلی استاد است، در نظر بگیرید:

Professor

Prof#	Pname	LastDegree#	LastDegreeName
100	علی راد	2	کارشناسی ارشد
101	آرش رضایی	2	کارشناسی ارشد
102	عسل شاملو	3	دکترا

این جدول، نرمال ۲ است زیرا نرمال ۱ است و از آنجا که کلید اصلی، تنها یک جز دارد، بدون شک، هیچ وابستگی جزیی در آن وجود ندارد. با این وجود، این جدول هنوز دارای ناهنجاری‌های زیر است:

۱- ناهنجاری حذف:

فرض کنید عسل شاملو، تنها کسی باشد که مدرک دکترا دارد. در این صورت، اگر بخواهیم اطلاعات عسل شاملو را حذف کنیم، اطلاعات مربوط به مدرک دکترا (کد مدرک و نام مدرک) را نیز از دست می‌دهیم.

۲- ناهنجاری در اصلاح:

اگر بخواهیم نام مدرک ۲ را از کارشناسی ارشد به کارشناسی تغییر دهیم، در این صورت مجبوریم این تغییر را در تاپل‌های مختلف، تکرار کنیم (اصلاح منتشر شونده).

برای رفع این معایب، جداول نرمال ۲ را به نرمال ۳ تبدیل می‌کنیم. یک جدول در حالت نرمال ۳ است اگر:

- نرمال ۲ باشد.
- هیچ وابستگی متعدی (وابستگی باواسطه[1]) در آن وجود نداشته باشد به عبارت دیگر، در آن، هیچ ویژگی غیرکلیدی به ویژگی غیرکلیدی دیگر، وابستگی تابعی نداشته باشد.

جدول Professor، نرمال ۳ نیست چون:

LastDegree# → LastDegreeName

برای تبدیل جداول نرمال ۲ به نرمال ۳، مراحل زیر را دنبال کنید:

۱- ویژگی‌هایی را که در وابستگی متعدی شرکت دارند، در یک جدول مجزا قرار داده، ویژگی‌هایی را که در طرف چپ این وابستگی قرار دارند، به عنوان کلید اصلی معرفی کنید.

Degree (LastDegree#, LastDegreeName)

۲- بقیه ویژگی‌های جدول اولیه را در جدول مجزای دیگری قرار داده (Prof# و Pname)، ویژگی‌های تشکیل‌دهنده کلید اصلی در جدول اول (LastDegree#) را به آن‌ها اضافه کنید.

Prof (Prof#, Pname, LastDegree#)

Degree

LastDegree#	LastDegreeName
2	کارشناسی ارشد
3	دکترا

Prof

Prof#	Pname	LastDegree#
100	علی راد	2
101	آرش رضایی	2
102	عسل شاملو	3

[1] Transitive Dependence

۴-۴-۲ جداول نرمال BCNF

برخی از جداول در حالت نرمال ۳ هستند ولی هنوز ناهنجاری دارند. به عنوان مثال، دانشگاهی با قوانین زیر را در نظر بگیرید:

۱- هر دانشجو می‌تواند در چند رشته تحصیلی، تحصیل کند ولی در هر یک از رشته‌های تحصیلی، تنها یک استاد راهنما دارد.

۲- هر استاد، تنها می‌تواند در یک رشته تحصیلی تدریس کند ولی در هر رشته تحصیلی، چندین استاد وجود دارند

جدول Project را که شامل شماره دانشجویی، رشته تحصیلی و استاد راهنمای دانشجو در رشته مربوطه است، در نظر بگیرید:

Project

St#	Field	Tutor
7801	مهندسی کامپیوتر	مجید رضایی
7801	ریاضی محض	آرش ریاضی‌دان
7801	هنر	گلناز هنردوست
7902	مهندسی کامپیوتر	مجید ریاضی
8001	مهندسی کامپیوتر	پروین صبا

در جدول Project، دو کلید کاندیدا وجود دارد:

کلید کاندیدای ۱: St# + Field

کلید کاندیدای ۲: St# + Tutor

فرض کنید کلید کاندیدای اول را به عنوان کلید اصلی انتخاب کنیم.

جدول Project، نرمال ۲ است چون در آن هیچ ویژگی غیرکلیدی به قسمتی از کلید اصلی، وابستگی ندارد (تنها ویژگی غیرکلیدی این جدول، Tutor است که نه به St# تنها وابسته است و نه به Field تنها).

این جدول، نرمال ۳ نیز هست چون در آن، هیچ ویژگی غیرکلیدی به ویژگی غیرکلیدی دیگر، وابستگی ندارد (چون این جدول تنها یک ویژگی غیرکلیدی دارد، این مسأله بدیهی است).

با وجود آن‌که این جدول، نرمال ۳ است، ولی هنوز ناهنجاری‌هایی دارد:

۱- ناهنجاری در درج:

به عنوان مثال، نمی‌توان استاد جدیدی را که هنوز هیچ دانشجویی با وی پروژه نگرفته است، در جدول درج کرد چون St# جزئی از کلید اصلی است و نمی‌تواند تهی باشد.

۲- ناهنجاری در حذف:

فرض کنید دانشجوی شماره 8001، تنها دانشجویی باشد که با پروین صبا پروژه گرفته است. در این صورت اگر این دانشجو را حذف کنیم، اطلاعات مربوط به این استاد نیز از بین می‌رود.

برای رفع این معایب، دو شخص به نام‌های Boyce و Codd فرم بهینه‌تری از نرمال ۳ را پیشنهاد کردند که به افتخار این دو شخص با نام BCNF معروف شد.

یک جدول، نرمال BCNF است اگر و تنها اگر کلیه تعیین‌کننده[1]های آن، کلید کاندیدا باشند.

تعریف تعیین کننده: اگر در یک جدول، ویژگی B به ویژگی A، وابستگی تابعی داشته باشد (A → B)، آنگاه A یک تعیین کننده خواهد بود.

جدول Project، نرمال BCNF نیست چون:

Tutor → Field

پس Tutor، یک تعیین کننده است درحالی‌که کلید کاندیدا نیست.

روند تبدیل جداول نرمال ۲ به نرمال BCNF، کاملاً مشابه روند تبدیل جداول نرمال ۲ به نرمال ۳ است.

برای تبدیل جدول Project به جدول نرمال BCNF:

۱- Tutor و Field را در یک جدول مجزا قرار داده، Tutor را به عنوان کلید اصلی این جدول معرفی می‌کنیم.

TC (Tutor, Field)

۲- بقیه ویژگی‌ها (St#) را در جدول دیگری قرار داده، کلید اصلی جدول اول یعنی Tutor را به آن‌ها اضافه می‌کنیم:

ST (St#, Tutor)

TC

Tutor	Field
مجید رضایی	مهندسی کامپیوتر
آرش ریاضی‌دان	ریاضی محض
گناز هنردوست	هنر
مجید رضایی	مهندسی کامپیوتر
پروین صبا	مهندسی کامپیوتر

ST

St#	Tutor
7801	مجید رضایی
7801	آرش ریاضی‌دان
7801	گلناز هنردوست
7902	آرش رضایی
8001	پروین صبا

تذکر: حالت‌های نرمال ۳ و نرمال BCNF، معمولاً با هم معادلند. تنها در صورتی که جدول، بیش از یک کلید کاندیدا داشته باشد و میان کلیدهای کاندیدا، ویژگی مشترک وجود داشته باشد، این دو حالت با هم معادل نخواهند بود. مثلاً در جدول Project، دو کلید کاندیدا وجود دارد و St# میان دو کلید کاندیدا مشترک است. به همین دلیل، حالت نرمال ۳ و BCNF این جدول، معادل نیستند.

۴-۵ جداول نرمال ۴

برخی از جداول، در حالت نرمال BCNF هستند ولی هنوز ناهنجاری‌هایی دارند. جدول employee را در نظر بگیرید. این جدول شامل شماره کارمندی و مهارت‌ها و زبان‌هایی است که هر کارمند بر آن‌ها مسلط است:

[1] Determinate

employee

Emp##	Skills	Languages
100	برنامه‌نویسی جاوا تجزیه و تحلیل شی‌گرا	انگلیسی
101	برنامه‌نویسی دلفی تجزیه و تحلیل شی‌گرا طراحی وب‌سایت	انگلیسی آلمانی

در این جدول، ستون‌های Skills و Languages دارای گروه‌های اطلاع تکرار شونده هستند. نمودار وابستگی این جدول، به شکل زیر است:

employee (Emp#, $\overline{\text{Skills}}$, $\overline{\text{Languages}}$)

فرض کنید این جدول را به صورت زیر، تبدیل به نرمال ۱ کنیم:

employee2

Emp##	Skills	Languages
100	برنامه‌نویسی جاوا	انگلیسی
100	تجزیه و تحلیل شی‌گرا	انگلیسی
101	برنامه‌نویسی دلفی	انگلیسی
101	تجزیه و تحلیل شی‌گرا	انگلیسی
101	طراحی وب‌سایت	انگلیسی
101	برنامه‌نویسی دلفی	آلمانی
101	تجزیه و تحلیل شی‌گرا	آلمانی
101	طراحی وب‌سایت	آلمانی

نمودار وابستگی جدول employee2، به شکل زیر خواهد بود:

employee2 (Emp#, Skills, Languages)

جدول employee2، نرمال BCNF است (چون یک جدول تمام کلید است و این مسأله، بدیهی است) ولی در درج، حذف و اصلاح، ناهنجاری دارد. مثلاً اگر کارمند ۱۰۱، مهارت «برنامه‌نویسی جاوا» را نیز کسب کند، به جای اضافه کردن یک تاپل، باید دو تاپل جدید زیر به جدول اضافه شود:

101	برنامه‌نویسی جاوا	انگلیسی
101	برنامه‌نویسی جاوا	آلمانی

و یا اگر بخواهیم زبان آلمانی را از اطلاعات کارمند ۱۰۱ حذف کنیم، به جای حذف یک تاپل، باید سه تاپل زیر را حذف کنیم:

101	برنامه‌نویسی دلفی	آلمانی
101	تجزیه و تحلیل شی‌گرا	آلمانی
101	طراحی وب‌سایت	آلمانی

برای رفع این معایب، جداول را به نرمال ۴ تبدیل می‌کنیم.

یک جدول، نرمال ۴ است اگر:

۱- نرمال BCNF باشد.

۲- هیچ وابستگی چندمقداری[1] در آن، وجود نداشته باشد.

تعریف وابستگی چندمقداری: اگر در جدول T (A, B, C)، به ازاء هر مقدار برای ویژگی A، مجموعه‌ای از مقادیر برای ویژگی C وجود داشته باشد و این مجموعه، مستقل از مقادیر ویژگی B باشد، ویژگی C به ویژگی A، وابستگی چندمقداری دارد (A →→ C).

مثال: جدول employee2 (Emp#, Skills, Languages) را در نظر بگیرید:

(Emp#, Skills) = (101, برنامه‌نویسی دلفی) ⇨ Languages = {آلمانی، انگلیسی}

(Emp#, Skills) = (101, تجزیه و تحلیل شی‌گرا) ⇨ Languages = {آلمانی، انگلیسی}

(Emp#, Skills) = (101, طراحی وب‌سایت) ⇨ Languages = {آلمانی، انگلیسی}

مشاهده می‌کنید که تنها #Emp، مجموعه Languages را تعیین می‌کند و تغییر Skills هیچ نقشی در تغییر مجموعه Languages ندارد. دلیل این امر نیز آن است که مهارت‌های یک شخص و زبان‌هایی که شخص بر آن‌ها تسلط دارد، هیچ ارتباطی با یکدیگر ندارند. پس Languages به #Emp وابستگی چندمقداری دارد (Emp# →→ Languages).

از طرف دیگر:

(Emp#, Languages) = (101, انگلیسی) ⇨ Skills = {برنامه‌نویسی دلفی, تجزیه و تحلیل شی‌گرا, طراحی وب‌سایت}

(Emp#, Languages) = (101, آلمانی) ⇨ Skills = {برنامه‌نویسی دلفی, تجزیه و تحلیل شی‌گرا, طراحی وب‌سایت}

در واقع، تنها #Emp مجموعه Skills را تعیین می‌کند و Languages هیچ نقشی در تغییر مجموعه Skills ندارد پس Skills نیز به #Emp وابستگی چندمقداری دارد (Emp# →→ Skills).

برای جلوگیری از ایجاد وابستگی چندمقداری، در همان مرحله اول و هنگام تبدیل جدول آنرمال به نرمال ۱، به ازای هر گروه اطلاع تکرارشونده، باید جدول مجزایی در نظر بگیریم:

emp_languages

Emp#	Languages
100	انگلیسی
101	انگلیسی
101	آلمانی

emp_skills

Emp#	Skills
100	برنامه‌نویسی جاوا
100	تجزیه و تحلیل شی‌گرا
101	برنامه‌نویسی دلفی
101	تجزیه و تحلیل شی‌گرا
101	طراحی وب‌سایت

[1] MultiValued Dependency

emp_languages (Emp#, Languages) emp_skills (Emp#, Skills)

هر دو جدول emp_languages و emp_skills، نرمال ۴ هستند.

مثال: جدول معروف SPJ را در نظر بگیرید:

SPJ

s#	p#	j#	qty
S1	P1	J1	12000
S1	P2	J1	20000
S1	P3	J1	3000
S1	P4	J1	8000
S1	P1	J2	10000
S1	P1	J3	9000
S2	P1	J1	2000
S2	P1	J3	5000
S2	P3	J1	8000
S3	P1	J1	9000
S3	P2	J3	9000
S3	P1	J3	4000

هر تولیدکننده ممکن است برای چندین پروژه، محصولاتی تولید کند. بنابراین، به ازای هر #s، مجموعه‌ای از #jها و مجموعه‌ای از #pها وجود دارند اما:

۱) ویژگی #j به #s، وابستگی چندمقداری ندارد چون:

(s#, p#) = (S2, P1) ⇨ j# = {J1, J3}

(s#, p#) = (S2, P3) ⇨ j# = {J1}

پس تنها #s، مجموعه #j را تعیین نمی‌کند بلکه با تغییر مقدار #p نیز احتمال تغییر مجموعه مقادیر #j وجود دارد.

۲) ویژگی #p به #s وابستگی چندمقداری ندارد چون:

(s#, j#) = (S1, J1) ⇨ p# = {P1, P2, P3, P4}

(s#, j#) = (S1, J2) ⇨ p# = {P1}

پس تنها #s، مجموعه #p را تعیین نمی‌کند بلکه با تغییر مقدار #j نیز احتمال تغییر مجموعه مقادیر #p وجود دارد.

می‌توان نتیجه گرفت در جدول SPJ، وابستگی چندمقداری وجود ندارد و این جدول، نرمال ۴ است.

۴-۶ جداول نرمال ۵

یک جدول در حالت نرمال ۵ است اگر:

۱- نرمال ۴ باشد.

۲- نتوان آن را به جداول کوچک‌تر تجزیه کرد به طوری‌که حداقل یکی از جداول، شامل هیچ یک از کلیدهای کاندیدای جدول اولیه نباشد.

مثال: طبق قوانین یک دانشگاه:

- هر دانشجو می‌تواند هر درس را چند بار ولی تنها یک بار با هر استاد بگیرد.
- هر استاد می‌تواند درس‌های مختلفی را تدریس کند و هر درس توسط اساتید مختلف تدریس می‌شود.

جدول enroll را در نظر بگیرید:

enroll

St#	Crs#	Prof#
S1	C1	P1
S1	C2	P2
S2	C2	P1

تنها کلید کاندیدای این جدول، St# + Crs# + Prof# و نمودار وابستگی این جدول، به شکل زیر است:

enroll (St#, Crs#, Prof#)

این جدول، نرمال BCNF است و چون وابستگی چندمقداری ندارد، نرمال ۴ نیز هست. برای تشخیص نرمال ۵ بودن این جدول، باید بررسی کنیم آیا این جدول قابل تجزیه به چند جدول کوچک‌تر که پیوند آن‌ها، جدول اولیه را نتیجه دهد هست یا خیر؟

این جدول را به دو جدول enroll1 و enroll2 می‌شکنیم:

enroll1

St#	Crs#
S1	C1
S1	C2
S2	C2

enroll2

St#	Prof#
S1	P1
S1	P2
S2	P1

نتیجه پیوند دو جدول، به صورت زیر خواهد بود:

enroll1 join enroll2

St#	Crs#	Prof#
S1	C1	P1
S1	C1	P2
S1	C2	P1
S1	C2	P2
S2	C2	P1

همانطور که مشاهده می‌کنید، پیوند دو جدول، دو تاپل اضافی تولید کرد که در جدول enroll اولیه وجود نداشت. پس این تجزیه، بی‌فایده است.

به همین ترتیب، اگر این جدول را به دو جدول enroll3 و enroll4 بشکنیم، داریم:

enroll3

St#	Crs#
S1	C1
S1	C2
S2	C2

enroll4

Crs#	Prof#
C1	P1
C2	P1
C2	P2

نتیجه پیوند دو جدول، به صورت زیر خواهد بود:

enroll3 join enroll4

St#	Crs#	Prof#
S1	C1	P1
S1	C2	P1
S1	C2	P2
S2	C2	P1
S2	C2	P2

همانطور که مشاهده می‌کنید، پیوند دو جدول، دو تاپل اضافی تولید کرد که در جدول enroll اولیه وجود نداشت. پس این تجزیه نیز بی‌فایده است.

حال روی جدول enroll، سه پرتو[1] (از این عملگر که در جبر رابطه‌ای تعریف شده و به صورت Π نشان داده می‌شود، برای گزینش عمودی در یک جدول استفاده می‌شود یعنی با استفاده از این عملگر می‌توان ستون‌های مورد نظر از یک جدول را انتخاب کرد) می‌گیریم:

SC

St#	Crs#
S1	C1
S1	C2
S2	C2

CP

Crs#	Prof#
C1	P1
C2	P2
C2	P1

SP

St#	Prof#
S1	P1
S1	P2
S2	P1

ویژگی مشترک بین دو جدول SC و CP، #Crs است. نتیجه پیوند دو جدول، جدول زیر خواهد بود:

SC join CP

St#	Crs#	Prof#
S1	C1	P1
S1	C2	P2
S1	C2	P1
S2	C2	P2
S2	C2	P1

ویژگی مشترک بین دو جدول (SC join CP) و SP، #St# + #Prof است. نتیجه پیوند دو جدول، جدول زیر خواهد بود:

(SC join CP) join SP

St#	Crs#	Prof#
S1	C1	P1
S1	C2	P2
S1	C2	P1
S2	C2	P1

همانگونه که مشاهده می‌کنید، نتیجه (SC join CP) join SP، با جدول enroll برابر نیست و شامل تاپل اضافی (S1, C2, P1) است که در جدول enroll وجود ندارد. این مشکل از آنجا ناشی می‌شود که:

۱- دانشجوی S1، درس C2 را گرفته است

۲- دانشجوی S1 با استاد P1 درس گرفته است

۳- درس C2 جز درس‌هایی است که استاد P1 تدریس می‌کند

[1] Projection

چنین نتیجه شده است که دانشجوی S1، درس C2 را با استاد P1 گرفته است در حالی که واقعیت ندارد. جدول enroll، سه ستون دارد پس نمی‌توان آن را به صورت ستونی به بیش از ۳ جدول تجزیه کرد و چون پیوند هیچ یک از تجزیه‌های دوتایی و سه‌تایی، ما را به جدول اصلی نرساندند، می‌توان نتیجه گرفت جدول enroll قابل تجزیه به جداول کوچک‌تر که پیوند آن‌ها جدول اولیه را نتیجه دهد، نیست پس جدول enroll، نرمال ۵ است.

مثال: فرض کنید قوانین زیر در یک دانشگاه حاکم باشند:

- هر استاد می‌تواند درس‌های مختلفی را تدریس کند و هر درس توسط اساتید مختلف تدریس می‌شود
- هر استاد برای هر درس، می‌تواند چند کتاب معرفی کند.
- اگر استادی یک درس را تدریس کند، علاوه بر کتاب‌هایی که خود معرفی می‌کند، مجبور است کلیه کتاب‌هایی که سایر مدرسین آن درس، برای دانشجویان معرفی کرده‌اند را نیز معرفی کند!!!

جدول PCB را در نظر بگیرید:

PCB

Prof#	Crs#	Book#
P1	C1	B1
P1	C2	B2
P3	C7	B8

تنها کلید کاندیدای این جدول، Prof# + Crs# + Book# است و نمودار وابستگی این جدول، به شکل زیر است:

PCB (<u>Prof#, Crs#, Book#</u>)

این جدول، نرمال ۴ است ولی یک ناهنجاری بسیار عجیب دارد. مثلاً اگر بخواهیم تاپل (P2, C2, B5) را در جدول درج کنیم، برای رعایت محدودیت اعمال شده از طرف دانشگاه، باید تاپل (P2, C2, B2) و تاپل (P1, C2, B5) را نیز درج کنیم!!!

حال باید تشخیص دهیم که با توجه به محدودیت اعمال شده از طرف دانشگاه، این جدول قابل تجزیه به جداول کوچک‌تر است یا خیر؟ تجزیه به دو جدول نتیجه‌ای ندارد پس تجزیه سه‌تایی را مورد بررسی قرار می‌دهیم:

PC

Prof#	Crs#
P1	C1
P1	C2
P3	C7

CB

Crs#	Book#
C1	B1
C2	B2
C7	B8

PB

Prof#	Book#
P1	B1
P1	B2
P3	B8

پیوند سه جدول فوق، ما را به جدول اولیه می‌رساند. به عبارت دیگر:

PC join CB join PB = PCB

از طرف دیگر، هیچ‌یک از جداول PC یا CB یا PB شامل کلید کاندیدای جدول اصلی نیستند پس جدول PCB، نرمال ۵ نیست و باید به سه جدول PC و CB و PB که هر سه، نرمال ۵ هستند، شکسته شود.

مثال: جدول Course را در نظر بگیرید:

Course

Crs#	Cname	Unit
1100	ادبیات	2
1105	تربیت بدنی	1
1400	برنامه‌سازی ۱	3
1402	برنامه‌سازی ۲	3

کلیدهای کاندیدای این جدول عبارتند از:

کلید کاندیدای ۱: #Crs

کلید کاندیدای ۲: Cname

اگر #Crs را به عنوان کلید اصلی معرفی کنیم، نمودار وابستگی این جدول، به شکل زیر خواهد بود:

Course (Crs#, Cname, Unit)

این جدول را می‌توان:

۱) به دو جدول {(Crs#, Cname), (Cname, Unit)} تجزیه کرد. در این صورت، نتیجه پیوند دو جدول، با جدول Course اولیه، یکسان خواهد بود ولی انجام این تقسیم کاملاً بی‌مورد است چون در هر دو جدول، حداقل یکی از کلیدهای کاندیدا حضور دارند.

۲) به دو جدول {(Crs#, Cname), (Crs#, Unit)} تجزیه کرد. در این صورت، نتیجه پیوند دو جدول با جدول Course اولیه یکسان خواهد بود ولی انجام این تقسیم کاملاً بی‌مورد است چون در هر دو جدول، حداقل یکی از کلیدهای کاندیدا حضور دارند.

پس می‌توان نتیجه گرفت که جدول Course، نرمال ۵ است و لازم نیست به جداول کوچک‌تر تجزیه شود.

تذکر: نرمال ۵، در عمل کاربردی ندارد و اکثر جداول تنها تا مرحله 3NF نرمال می‌شوند چون در حالت نرمال ۳، هیچ مشکلی ندارند.

۴-۷) تمرین‌های حل شده

۴-۷-۱) تمرین ۱

سیستم کتابخانه یک دانشگاه را در نظر بگیرید. در این کتابخانه، به هر کتاب، یک شماره منحصر به فرد داده شده است مثلاً اگر در کتابخانه، سه نسخه از یک کتاب وجود داشته باشد، هر یک از آن‌ها یک شماره مجزا خواهد داشت. در هر امانت، دانشجو می‌تواند حداکثر سه کتاب را به امانت بگیرد (مثلاً در امانت شماره ۱۰۰۱ ممکن است دو کتاب ۱۴۰۳ و ۱۸۰۹ به امانت گرفته شوند) و باید همه کتاب‌هایی را که با هم از کتابخانه به امانت گرفته است، با هم به کتابخانه برگرداند. با توجه به این قوانین، نمودار وابستگی جدول loan را رسم کرده، آن را به جدول نرمال ۳ تبدیل کنید.

loan (loan#, book#, bname, author, st#, sname, field, loanDate, returnDate)

loan#: شماره امانت

book#: شماره کتاب

bname: نام کتاب

author: نام نویسنده

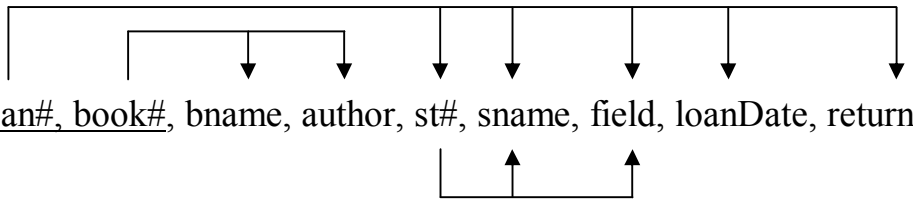
#st: شماره دانشجویی که کتاب را به امانت گرفته است

sname: نام دانشجو

field: رشته تحصیلی دانشجو (هر دانشجو، تنها در یک رشته تحصیل می‌کند)

loanDate: تاریخ امانت گرفتن کتاب

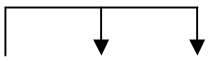
returnDate: تاریخ برگشت کتاب به کتابخانه توسط دانشجو



loan (loan#, book#, bname, author, st#, sname, field, loanDate, returnDate)

تبدیل به نرمال ۲:

این جدول، نرمال ۲ نیست چون bname و author به #book که جزیی از کلید اصلی است، وابستگی تابعی دارند. همچنین، #st و sname و field و loanDate و returnDate به #loan که جزیی از کلید اصلی است وابستگی تابعی دارند. پس این جدول را به جداول نرمال ۲ می‌شکنیم:



L (loan#, st#, sname, field, loanDate, returnDate)
Book (book#, bname, author)
LB (loan#, book#)

نکته: توجه کنید اگر چه هیچ ویژگی غیرکلیدی که به #loan, #book وابستگی داشته باشد و در سطرهای قبل نیامده باشد، وجود ندارد، ولی این ردیف را به جدول LB تبدیل کرده‌ایم. دلیل این امر، آن است که تنها دراین جدول مشخص می‌شود در هر امانت، چه کتاب‌هایی امانت گرفته شده‌اند. پس این جدول، اطلاعات مفیدی در اختیار می‌گذارد که در جداول دیگر موجود نیست.

تبدیل به نرمال ۳:

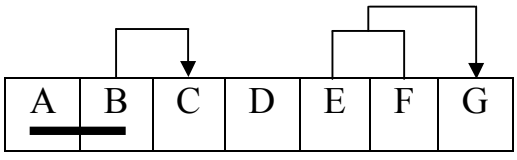
جداول B و LB، نرمال ۳ هستند چون در آن‌ها هیچ ویژگی غیر کلیدی به ویژگی غیر کلیدی دیگر، وابستگی تابعی ندارد ولی جدول L، نرمال ۳ نیست چون در آن، sname و field، به #st که غیر کلیدی است، وابستگی دارد. پس جدول L را به جداول زیر که هر یک، نرمال ۳ هستند، می‌شکنیم:

L1 (st#, sname, field)
L2 (loan#, loanDate, returnDate)

پس جدول loan به چهار جدول LB و Book و L1 و L2 که همگی، نرمال ۳ هستند، شکسته شد.

۴-۷-۲) تمرین ۲

جدول زیر را با ذکر کلیه مراحل، به جداول نرمال ۳ تبدیل کنید.

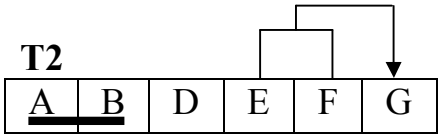


تبدیل به نرمال ۲: از آن‌جا که ویژگی C به ویژگی B که بخشی از کلید اصلی است، وابستگی تابعی دارد، در این جدول، وابستگی جزیی وجود دارد. پس نرمال ۲ نیست. برای تبدیل به نرمال ۲:

T1

B	C

T2



تبدیل به نرمال ۳: جدول T1، نرمال ۳ است ولی در جدول T2، ویژگی G که یک ویژگی غیر کلیدی است، به ویژگی‌های E و F که غیر کلیدی می‌باشند وابستگی تابعی دارد، پس جدول T2 نرمال ۳ نیست. برای تبدیل به نرمال ۳، این جدول را به دو جدول T21 و T22 می‌شکنیم:

T21

E	F	G

T22

A	B	D	E	F

۴-۸) تمرین‌های فصل

۱- جدول factor را که شامل اطلاعات مربوط به فاکتورهای فروش یک فروشگاه است، در نظر بگیرید. در هر فاکتور ممکن است اقلام زیادی وجود داشته باشند.

factor (fact#, fdate, customer#, cname, address, item#, iname, fee, qty)

fact#: شماره فاکتور

fdate: تاریخ صدور فاکتور

customer#: شماره مشتری

cname: نام مشتری

address: آدرس مشتری

item#: کد کالای خریداری شده

iname: نام کالا

fee: هزینه هر واحد

qty: مقدار یا تعداد کالای خریداری شده

نمودار وابستگی این جدول را رسم کرده، آن را به جداول نرمال ۳ تبدیل کنید.

۲- یک وب‌سایت برای برگزاری آزمون‌های مختلف را درنظر بگیرید. هر آزمون، تعدادی سؤال و هر سؤال، تعدادی گزینه دارد که تنها یکی از آن‌ها صحیح است. اعضا سایت، با وارد کردن نام کاربری و کلمه عبور خود، وارد سایت شده، آزمون مورد نظر خود را انتخاب کرده، به سؤالات چندگزینه‌ای آن پاسخ می‌دهند و

امتیاز می‌گیرند. هر کاربر می‌تواند در آزمون‌های مختلف ولی در هر آزمون، تنها یک بار شرکت کند. نحوه ارزیابی آزمون‌های مختلف، متفاوت است مثلاً ممکن است در یک آزمون، امتیاز منفی به ازای هر جواب نادرست، 0.25 و در دیگری 0.5 باشد و یا حداقل امتیاز کل برای قبولی در یک آزمون، ۱۰ و در دیگری، ۸۰ باشد. جدول test را که شامل اطلاعات آزمون‌های مختلف است در نظر گرفته، نمودار وابستگی آن را رسم کرده، به جداول نرمال ۳ تبدیل کنید:

test (userid, password, test#, title, creditPerTrue, negPerFalse, total, neededCredit, tdate, q#, qtext, trueChoice#, choice#, ctext, userChoice#)

userid: نام کاربری که در آزمون شرکت کرده است (هر کاربر، یک نام منحصر به فرد دارد)

password: کلمه عبور کاربر

test#: شماره آزمون (هر آزمون، یک شماره منحصر به فرد دارد)

title: عنوان آزمون

creditPerTrue: امتیاز به ازای هر جواب صحیح

negPerFalse: امتیاز منفی به ازای هر جواب نادرست

total: کل امتیاز آزمون

neededCredit: حداقل امتیاز مورد نیاز برای قبولی در آزمون

tdate: تاریخ شرکت کاربر در آزمون

q#: شماره سؤال (در هر آزمون، هر سؤال یک شماره منحصر به فرد دارد)

qtext: متن سؤال

trueChoice#: شماره گزینه درست (در هر یک از سؤال‌ها، تنها یکی از گزینه‌ها صحیح است)

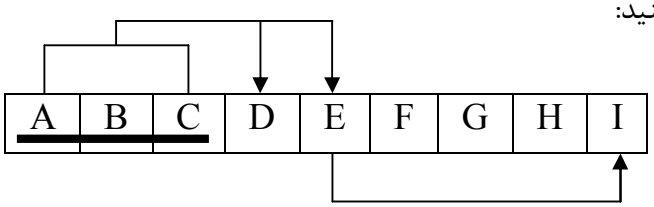
choice#: شماره گزینه (در هر سؤال، هر گزینه یک شماره دارد)

ctext: متن گزینه

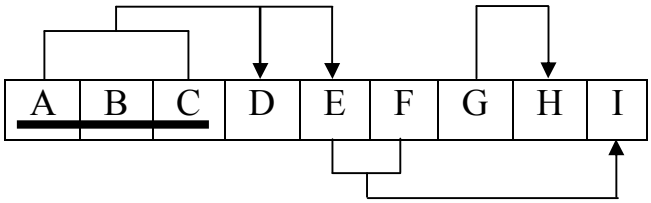
userChoice#: شماره گزینه‌ای که کاربر برای این سؤال انتخاب کرده است.

۳- جداول زیر را به جداول نرمال BCNF تبدیل کنید:

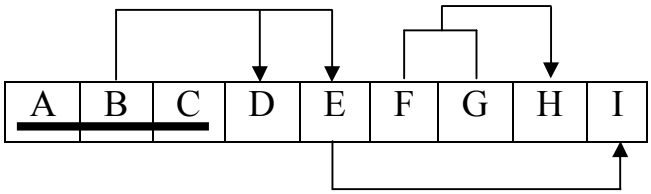
الف)



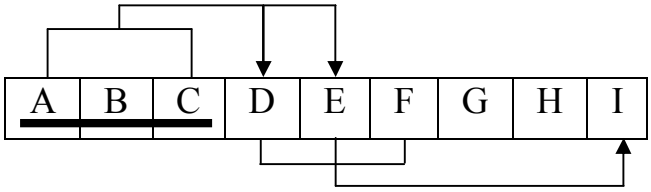
ب) به روش معمولی حل نمی‌شود.



ج)



د) به روش معمولی حل نمی‌شود.



هـ)