بسمه‌تعالی

دانشکده مهندسی کامپیوتر

جزوه درس طراحی سیستم‌های پایگاه داده

Database Systems Design

نگارنده..... محسن رضوانی، استادیار دانشکده مهندسی کامپیوتر

شماره نگارش..... ۰/۳

تاریخ نگارش..... ۱۳۹۶/۰۱/۰۵

شاهرود، بلوار دانشگاه، دانشگاه صنعتی شاهرود، دانشکده مهندسی کامپیوتر، کد پستی: ۳۶۱۹۹۹۵۱۶۱

(023) 32392204-9 (023) 32392204 www.shahroodut.ac.ir

این جزوه جهت استفاده دانشجویان درس طراحی سیستم‌های پایگاه داده تدوین شده است و خلاصه‌ای از برخی از فصول کتاب مرجع زیر است.

Fundamentals of Database Systems, by Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2016.

فهرست مطالب در یک نگاه

فهرست مطالب

۱ پایگاه داده و کاربران

۱-۱ مقدمه

پردازش داده‌ها از دهه ۱۹۵۰ تا کنون فراز و نشیب‌های بسیاری داشته است. در اوایل کاربران مستقیماً با محیط فیزیکی یا همان سخت‌افزار کامپیوتر تماس داشتند و داده‌ها را روی آن ذخیره و بازیابی می‌کردند. با گذشت زمان نرم‌افزارهایی به نام شیوه دستیابی[۱] به وجود آمد که رابط بین کاربر و کامپیوتر بود و کارها را تا حدی آسان می‌نمود. اما باز هم تأمین امنیت داده‌ها و هم چنین اشتراک داده‌ها در سطح قابل قبول با مشکلاتی همراه بود. این گونه مشکلات باعث ایجاد انقلابی در بانک اطلاعات گردید که منجر به ایجاد «سیستم مدیریت پایگاه داده» (DBMS[۲]) گردید.

لازمه وجود Data Base وجود حداقل یک DBMS در محیط نرم‌افزاری است.

دو روش و خط مشی کلی در طراحی یک سیستم کاربردی وجود دارد.

- روش بانکی (سیستم پردازش فایل[۳]): در این روش برای هر زیرمجموعه عملیاتی یک سیستم خاصی که فقط پاسخگوی همان زیرمحیط است طراحی می‌شود به گونه‌ای که هر قسمت، سیستم کاربردی خاص و جداگانهی خود را دارد. یعنی محیط ذخیره‌سازی نامجتمع است. سیستم‌های متداول پردازش فایل توسط سیستم عامل‌های رایج پشتیبانی می‌گردند. در این روش رکوردها درون فایل‌ها ذخیره شده و برنامه‌های کاربردی مختلفی نوشته می‌شوند که یک رکورد را از فایل استخراج و یا رکوردهایی را به یک فایل مشخص اضافه نمایند. پیش از به وجود آمدن DBMS، سازمان‌ها برای نگهداری اطلاعات خود از چنین سیستم‌هایی استفاده می‌کردند. در این سیستم ممکن است عدم سازگاری در داده‌ها و فایل‌ها دیده شود. اطلاعات تکراری و افزونگی (اطلاعات یکسانی در مکان‌های مختلف به صورت تکراری نگهداری شوند) در داده‌ها وجود دارد که سبب افزایش حافظه مورد نیاز برای ذخیره‌سازی و هزینه دسترسی به داده‌ها می‌گردد. امکان ایجاد استانداردهای واحد و یا اعمال یک سری عملیات منسجم به دلیل تعدد سیستم‌ها وجود ندارد. نمی‌توان داده‌ها را به اشتراک گذاشت و باعث مصرف غیر بهینه امکانات نرم‌افزاری و سخت‌افزاری می‌شود.
- روش بانکی[۴]: در این روش یک تیم واحد طراحی و پیاده‌سازی، به سرپرستی متخصصی به نام DBA[۵] مجموعه نیازهای کل محیط عملیاتی را بررسی می‌کنند و توسط نرم‌افزاری محیط واحد و مجتمع ذخیره‌سازی اطلاعات ایجاد می‌گردد

---

۱ Method Access

۲ Data Base Management System

۳ File Processing

۴ Data Base Approach

۵ Data Base Administrator

یعنی کل داده‌ها یک بانک مجتمع هستند و از طریق DBMS با آن‌ها ارتباط برقرار می‌شود. می‌توان در این سیستم داده‌ها را به اشتراک گذاشت. کاربران در یک محیط انتزاعی هستند و DBMS خود به مسائل پردازش فایل می‌پردازد. ناسازگاری در داده‌ها وجود ندارد و میزان افزونگی در داده‌ها نسبت به روش قبل بسیار کاهش یافته است.

۱-۲ داده و اطلاعات

هر حقیقتی که در سیستمِ مورد بررسی ارزشمند باشد را **داده** می‌نامیم. در این تعریف به‌طور خاص بر روی سیستم مورد بررسی تأکید شده است زیرا ممکن است این حقیقت در سیستم دیگر ارزشمند نباشد. همچنین داده‌های پردازش شده را **اطلاعات** می‌نامیم.

۱-۳ تعریف پایگاه داده

پایگاه داده مجموعه‌ای است از داده‌های ذخیره شده، ماندگار[۶] (مانا)، مجتمع[۷]، به هم مرتبط، حتی‌الامکان فاقد افزونگی[۸]، که دارای یک معماری خاص مبتنی بر مدل داده‌ای مشخص، تحت مدیریت یک سیستم کنترل متمرکز و مورد استفاده یک یا چندین کاربر از یک محیط (سازمان) به طور اشتراکی و هم‌زمان می‌باشد.

به بیان خلاصه‌تر یک پایگاه داده (DB) مجموعه‌ای از داده‌های مرتبط به هم می‌باشد که شامل اطلاعاتی در مورد یک سازمان یا ارگان خاص است و هدف از طراحی آن مدیریت مقادیر عظیمی داده است. پایگاه داده آموزش دانشگاه صنعتی شاهرود نمونه‌ای از یک DB می‌باشد.

مدیریت داده شامل دو بخش کلی زیر است:

- تعریف ساختارهایی برای ذخیره اطاعات.
- تدارک مکانیزم‌هایی برای دستکاری[۹]، دسترسی و تغییر اطلاعات.

منظور از ماندگاری داده در تعریف DB این است که تا زمانی که کاربر مجاز، درخواست تغییر یا حذف داده را نداده است، داده محفوظ بماند که مسئولیت این کار با DBMS می‌باشد.

---

[۶] Persistency

[۷] Integrated

[۸] Redundancy

[۹] Manipulation

۱-۴ سیستم مدیریت پایگاه داده

مجموعه‌ای از داده‌های مرتبط با یکدیگر به همراه مجموعه‌ای از برنامه‌ها که از مجموعه برنامه‌های مزکور برای دسترسی به مجموعه داده‌های مرتبط استفاده می‌شود. هدف اصلی DBMS این است که محیطی آسان و قابل فهم از دیدگاه کاربری و در عین حال کارا و قدرتمند از دیدگاه برنامه‌نویسی فراهم آورد که به منظور ذخیره و بازیابی اطلاعات پایگاه داده مورد استفاده قرار می‌گیرد.

به بیان خلاصه‌تر یک DBMS مجموعه‌ای از برنامه‌های نرم‌افزاری است که این امکان را به کاربران می‌دهد که بتوانند پایگاه داده‌ای را ایجاد و نگهداری کنند.

تعداد DBMSها در دنیا کم است. عموم DBMSهای زیر می‌توانند چندین DB را Support کنند.

- Oracle
- SQL Server
- My SQL
- Microfoft Access
- Fox pro
- DB2
- Informix

۱-۵ داده‌های جانبی

۱-۵-۱ فرهنگ داده

شبیه لغت‌نامه‌های معمولی است و تمامی اسامی استفاده شده در سیستم و معنای آن‌ها را دربرمی‌گیرد و مرجعی برای ایجاد یکنواختی و هماهنگی در نام داده‌ها و معنای آن‌هاست. در بانک‌های اطلاعاتی مهم، نرم‌افزارهای ویژه‌ای برای کار با لغت‌نامه‌ها وجود دارد.

۱-۵-۲ کاتالوگ سیستم

علاوه بر اسامی داده‌ها و مشخصات آن‌ها اطلاعات دیگری نیز در مورد بانک‌های اطلاعاتی باید نگهداری شود، مانند اطلاعات مربوط به حق دستیابی افراد به داده‌های مختلف، اندازه هر جدول یا شیء، تاریخ ایجاد و به روز درآوردن داده‌ها و غیره. که این اطلاعات در کاتالوگ سیستم نگهداری می‌شوند.

محیط بانک‌های اطلاعاتی همانند هر محیط دیگر ذخیره و بازیابی از عناصر زیر تشکیل می‌شود:

- سخت‌افزار
  - ذخیره‌سازی داده‌ها
  - سخت‌افزار پردازشی
  - سخت‌افزار ارتباطی

- نرم‌افزار
  - نرم‌افزار کاربردی
  - نرم‌افزار سیستمی
- داده
- کاربر
  - کاربر با نقش مدیریتی (DBA)
  - کاربر با نقش استفاده کننده

۲ معماری پایگاه داده

۲-۱ داده انتزاعی

برای آن‌که سیستمی قابل استفاده باشد لازم است داده را به صورت کارا بازیابی نماید. این الزام منجر به طراحی ساختارهای داده‌ای پیچیده برای نمایش داده در DB می‌گردد. از آن‌جا که بسیاری از کاربران DB لزوماً آشنا با کامپیوتر و دیدگاه مهندسین کامپیوتر نیستند، بنابراین سازندگان DB این پیچیدگی را در قالب سطوح مختلف انتزاع از کاربر خود مخفی می‌نمایند. این موضوع سبب تسهیل ارتباط و تعامل کاربر با سیستم پایگاه داده می‌گردد. که این سطوح عبارتند از:

- سطح فیزیکی که پایین‌ترین سطح انتزاع داده است و مشخص می‌نماید که داده واقعی چگونه ذخیره می‌شود. در این سطح از انتزاع ساختارهای داده‌ای سطح زیرین به طور دقیق و با تمام جزئیات توصیف می‌شوند.
- سطح منطقی که سطح بعدی است و بیانگر آن است که چه داده‌هایی در پایگاه داده ذخیره شده‌اند و چه نوع ارتباطی میان این داده‌ها وجود دارد. سطح منطقی توسط مدیر DB مورد استفاده قرار می‌گیرد.
- سطح دیدگاه که بالاترین سطح انتزاع است و تنها بخشی از کل پایگاه داده را توصیف می‌کند. این سطح برای ساده‌سازی تعامل میان کاربران و DB تعریف می‌شود. در این سطح، سیستم چندین دیدگاه از یک DB واحد به وجود می‌آورد که کاربران به تناسب نیازهای خود با یک یا چند تا از این دیدگاه‌ها در ارتباط و تعامل هستند.

۲-۲ معماری سه لایه و استقلال داده‌ای

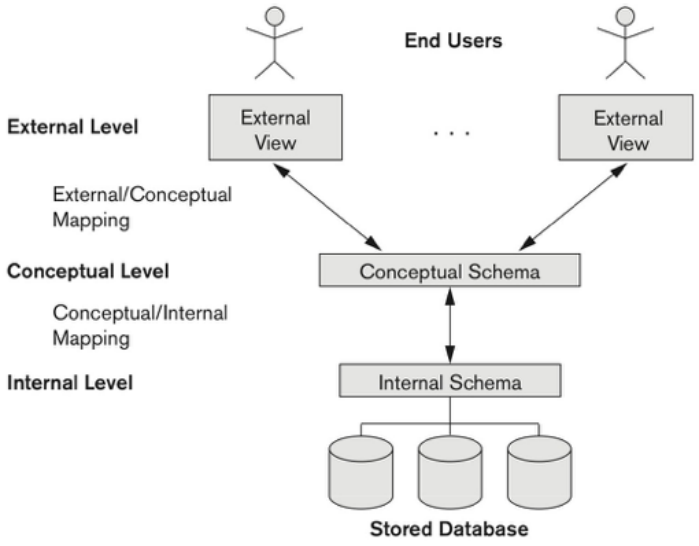
۲-۲-۱ معماری سه لایه

در سال ۱۹۷۵، ANSI یک معماری سه لایه تحت عنوان معماری استاندارد اعلام کرد که البته جای بانک اطلاعاتی به عنوان مدل انتخابی در آن خالی بود. با افزودن این لایه به آن مدل استاندارد می‌توانیم یک مدل معماری کامل‌تر چهار لایه به دست آوریم. این لایه‌ها به‌صورت زیر است.

۱) سطح خارجی: دیدهای کاربران مختلف (views) که بالاترین و خارجی‌ترین لایه است که کاربران تنها حق دستیابی به این لایه را دارند. به هر کاربر تنها اجازه دسترسی به بخشی از بانک اطلاعاتی که به او مربوط می‌شود، داده خواهد شد. مثلاً کارمندان بخش ارزی بانک اجازه دسترسی به اطلاعات مشتریان در بخش وام و حساب افراد را ندارند و لزومی به این دسترسی هم نمی‌باشد. در واقع به هر کس تا آن‌جا اطلاعات داده می‌شود که نیاز دارد.

۲) سطح مفهومی: در این سطح ساختار کامل پایگاه داده شرح داده می‌شود ولی جزئیات مربوط به سطح فیزیکی و نحوه ذخیره‌سازی داده‌ها بررسی نمی‌شود. مدل‌های داده‌ای سطح بالا و نمایشی در این سطح قرار می‌گیرند.

۳) سطح داخلی: در این سطح جزئیات ذخیره‌سازی داده‌ها در پایگاه داده بررسی می‌شود. مدل داده‌ای فیزیکی در این سطح از پایگاه داده قرار می‌گیرد.



هدف اصلی معماری سه لایه‌ای برای سیستم‌ای پایگاه داده فراهم آوردن استقلال داده‌هاست.

۲-۲-۲ استقلال داده‌ای

هدف از استقلال داده‌ای این است که امکان ایجاد تغییر در تعریف شمای پایگاه داده در یک سطح به نحوی فراهم شود که تعریف شمای سطح دیگر تحت تأثیر قرار نگیرد.

دو سطح از استقلال برای داده‌ها تعریف می‌شود:

۱) استقلال فیزیکی داده که توانایی تغییر شمای فیزیکی است بدون آن‌که نیازی به ایجاد تغییر در شمای تعریف شده در سطح منطقی باشد.
۲) استقلال منطقی داده که امکان تغییر در شمای منطقی است بدون آن‌که نیازی به دوباره نویسی برنامه‌های کاربردی باشد.

حفظ استقلال منطقی داده‌ها به مراتب دشوارتر از حفظ استقلال فیزیکی داده‌هاست.

مفهوم استقلال داده‌ای در DB از بسیاری جهات قابل مقایسه با مفهوم نوع داده مجرد (ADT[۱۰]) است که در زبان‌های برنامه‌نویسی مطرح شده است.

هر دو مفهوم فوق جزئیات پیاده‌سازی را از کاربران مخفی نگه می‌دارند و به این ترتیب به کاربر اجازه می‌دهند که به جای پرداختن به جزئیات پیاده‌سازی بر روی ساختار کلی تمرکز نماید.

---

[۱۰] Abstract Data Type

۲-۳ شما، نمونه و حالت پایگاه داده

مجموعه داده‌های ذخیره شـده در DB را در یک زمان خاص یک نمونه[۱۱] از DB می‌نامند. سـاختار کلی پایگاه داده را هم شـمای آن DB می‌نامند. شـمای یک DB معمولاً ثابت و بدون تغییر است و یا در فواصل زمانی بسیار طولانی و به ندرت دچار تغییراتی می‌گردد. شمای یک DB مشابه تعریف نوع در زبان‌های برنامه‌نویسی است.

مجموعه داده‌هایی که در حالت خاص در DB قرار دارند حالت پایگاه داده نامیده می‌شـوند. بنابراین هرگونه تغییر در داده‌های پایگاه داده منجر به تغییر حالت پایگاه داده می‌شود.

۲-۴ مدل داده‌ای

زیربنای سـاختار DB بر اسـاس مفهوم مدل داده‌ای بنا شـده اسـت. به مجموعه‌ای از ابزارهای مفهومی که به منظور توصیف سـاختار داده، ارتباطات داده‌ای، سـمانتیک داده‌ای و قیود یکپارچگی به کار گرفته می‌شـوند مدل داده‌ای می‌گویند. مدل داده‌ای که تاکنون ارائه شده‌اند در قالب ۳ گروه کلی طبقه‌بندی می‌شوند:

۱) مدل منطقی مبتنی بر اشیاء (Object-Based Logical Models)

که به منظور توصـیف داده‌ها در سـطوح منطقی و دیدگاه به کار گرفته می‌شـوند. این مدل‌ها اجازه می‌دهند که محدودیت‌های داده‌ای دقیقاً مشخص شوند. مدل داده‌ای معروف از این گروه عبارتند از:

- مدل نهاد- رابطه
- مدل شیءگرا
- مدل داده تابعی
- مدل داده سمانتیکی

۲) مدل‌های منطقی مبتنی بر رکوردها (Record-Based Logical Models)

این مدل‌ها هم به منظور توصـیف داده‌ها در دو سطح منطقی و دیدگاه مورد استفاده قرار می‌گیرند. بر خلاف مدل‌های داده‌ای مبتنی بر اشـیاء، مدل مبتنی بر رکورد علاوه بر مشـخص نمودن سـاختار منطقی کلی یک DB توضـیحات سـطح بالاتری نیز برای پیاده‌سازی ارائه می‌کنند.

سه مدل داده‌ای مبتنی بر رکورد که بیش از سایر مدل‌ها مورد توجه و پذیرش قرار گرفته‌اند عبارتند از:

- مدل‌های رابطه‌ای (Relational)

---

[۱۱] Instance

- مدل سلسله مراتبی (Hierarchical)
- مدل شبکه‌ای (Network)

۳) مدل داده فیزیکی (Physical Data Model)

به منظور توصیف داده در پایین‌ترین سطح به کار گرفته می‌شود. از شناخته شده‌ترین مدل فیزیکی:

- مدل یکسان‌سازی (Unifying Model)
- مدل قاب- حافظه (Frame-memory Model)

۲-۵ زبان‌های پایگاه داده

هر سیستم پایگاه داده‌ای باید بتواند دو نوع زبان را فراهم نماید. یک نوع زبان برای مشخص نمودن شمای DB و دیگری زبانی که قادر به توصیف درخواست‌های مطرح‌شده از DB و یا ایجاد تغییر در DB باشد. به‌طور کلی زبان‌های مورد نیاز در یک DBMS شامل موارد زیر است.

- DDL[۱۲]
- SDL[۱۳]
- VDL[۱۴]
- DML[۱۵]

۲-۵-۱ زبان تعریف داده

شمای یک DB به وسیله مجموعه‌ای از دستورات تعریف می‌شود که توسط زبان مخصوصی به نام زبان تعریف داده (DDL) انجام می‌شود. نتیجه کامپایل دستورات DDL به وجود آمدن مجموعه‌ای از جدول است که اطلاعات آن‌ها درون فایل ویژه‌ای به نام فرهنگ داده ذخیره می‌شوند.

۲-۵-۲ زبان دستکاری داده

وقتی سخن از دستکاری داده به میان می‌آید هر یک از عملیات زیر می‌تواند مورد نظر باشد:

- بازیابی داده‌های ذخیره شده در DB

---

[۱۲] Data Definition Language
[۱۳] Storage Definition Language
[۱۴] View Definition Language
[۱۵] Data Manipulation Language

- اضافه کردن داده جدید به DB
- حذف داده از DB
- تغییر داده‌های ذخیره شده در DB

یک زبان دستکاری داده (DML) زبانی است که کاربران را قادر می‌سازد که داده‌ای را که بر اساس مدل داده معینی مورد دسترسی و دستکاری قرار دهند. این زبان‌ها دارای دو نوع اصلی‌اند:

- DMLهای روالی: که نیاز به کاربری دارند که برای آن‌ها مشخص نماید که کدام داده مورد نیاز است و چگونه باید به این داده‌ها دسترسی یافت.
- DMLهای غیر روالی: که در آن‌ها کاربر تنها مشخص می‌کند که به کدام داده نیاز است بدون آن که درباره چگونگی دستیابی به داده توضیحی بدهد. معمولاً برای یادگیری هستند و به کارگیری آن‌ها آسان‌تر از DMLهای روالی است.

زبان SDL در تعیین لایه داخلی استفاده می‌شود و معمولاً طراحان با این زبان کار نمی‌کنند.

برای داشتن یک معماری سه لایه صحیح ما به زبان سومی احتیاج داریم که VDL می‌باشد که برای تعیین و تعریف view کاربر در سیستم و نگاشت آن به لایه مفهومی استفاده می‌شود.

۲-۶ ذخیره و مدیریت داده

زمانی که صحبت از پایگاه داده به میان می‌آید معمولاً با حجم عظیم داده سر و کار داریم. داده‌هایی که حجم آن‌ها به گیگابایت و حتی ترابایت هم می‌رسد. از آن‌جا که اندازه حافظه اصلی جوابگوی نگهداری این میزان داده نیست لازم است از حافظه جانبی برای ذخیره‌سازی استفاده کنیم. از آن‌جا که سرعت انتقال داده از حافظه جانبی به اصلی و برعکس نسبت به سرعت پردازشگر بسیار کم‌تر است، لازم است که سیستم پایگاه داده، داده‌ها را طوری سازمان دهد که تعداد انتقالات داده میان حافظه اصلی و جانبی به حداقل برسد.

مدیر ذخیره‌سازی یک بخش نرم‌افزاری در سیستم پایگاه داده است که واسطی میان داده‌های ذخیره شده در سطوح پایین DB و برنامه‌های کاربردی و پرس‌وجوهای درخواست شده از سیستم را فراهم می‌نماید. مدیر ذخیره‌سازی وظیفه ارتباط با مدیریت فایل‌ها را بر عهده دارد. مدیر ذخیره‌سازی دستورات مختلف DML را به صورت دستورات سطح پایین برای کار با فایل‌ها ترجمه می‌کند. پس مسئولیت ذخیره، بازیابی و تغییر داده‌های موجود در DB بر عهده مدیر ذخیره‌سازی است.

۲-۷ مدیر و کاربران پایگاه داده

۲-۷-۱ مدیر پایگاه داده

یکی از مهم‌ترین دلایل به کارگیری DBMS، ایجاد کنترل مرکزی بر روی داده و برنامه‌هایی است که به داده دسترسی می‌یابند. فردی که چنین کنترلی بر روی سیستم دارد مدیر پایگاه داده (DBA) نامیده می‌شود. وظایف DBA عبارتند از:

- تعریف شما (Schema)

- تعریف ساختار و متد دسترسی به حافظه
- ایجاد تغییر در شما و ساماندهی فیزیکی
- اعطای اعتبار برای دسترسی به داده‌ها
- مشخص کردن قیود یکپارچگی

۲-۷-۲ کاربران پایگاه داده

هدف اصلی از به وجود آمدن سیستم‌های DB ایجاد محیطی برای بازیابی اطلاعات از DB و ذخیره اطلاعات جدید در آن است. چهار دسته مختلف از کابران DB وجود دارند که تفاوتشان در طریقه تعامل آن‌ها با DB است.

- برنامه‌نویسانی که با کامپیوتر آشنا هستند و ارتباط آن‌ها با سیستم از طریق اجرای دستورات DML است.
- کاربران معمولی که درخواست‌های خود را در قالب یک زبان پرس‌وجوی DB تنظیم می‌نمایند و به سیستم می‌فرستند.
- کاربران ویژه‌ای که کاربران معمولی به حساب نمی‌آیند و برنامه‌های خاص پایگاه داده‌ای می‌نویسند که در چهارچوب پردازش داده متداول نمی‌گنجد.
- کاربران متخصصی که کاربران عادی نیستند و با سیستم از طریق فراخوانی یک برنامه کاربردی از پیش نوشته شده در تعامل‌اند.

۲-۸ تراکنش و جامعیت

۲-۸-۱ تراکنش

هرگونه برنامه‌ای که توسط کاربر در محیط بانک اطلاعات اجرا می‌شود تراکنش نام دارد و تحت نظارت DBMS انجام می‌شود. یعنی DBMS می‌تواند تراکنش‌ها را حذف، کنترل و یا به تعویق بیاندازد که تمامی این موارد در جهت حفظ صحت و جامعیت DB صورت می‌گیرد.

۲-۸-۲ جامعیت

به مفهوم صحت داده‌ها و پردازش‌ها و پیروی از مقررات سیستم است. به عنوان مثال موجودی واقعی حساب‌های بانکی نباید منفی باشد و یا شخصی نتواند بیش از موجودی خود از حسابش برداشت کند. نوعی از جامعیت به سازگاری[۱۶] موسوم است که بدین معنا است که اقلام داده در کل سیستم با هم در تضاد نباشند. مثلاً در نسخه‌ای از پرونده بانک، موجودی شخص با نسخه دیگر پرونده که معادل آن است متفاوت نباشد.

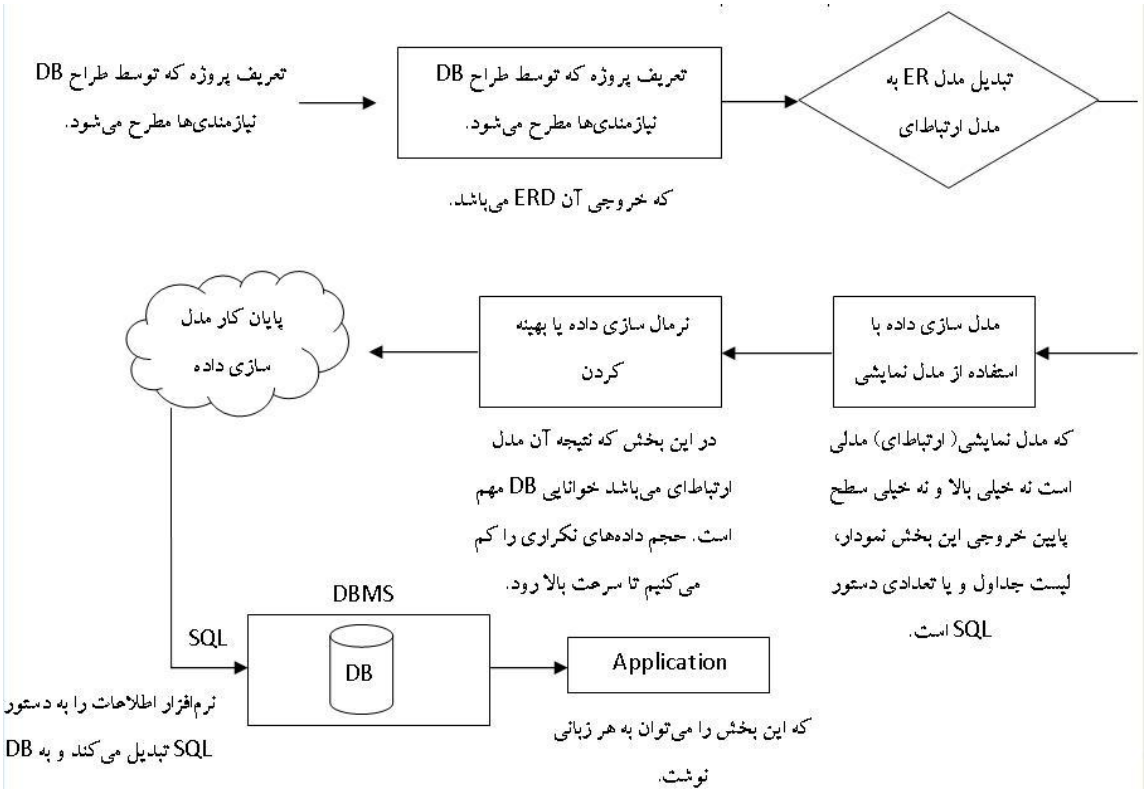
---

[۱۶] Consistency

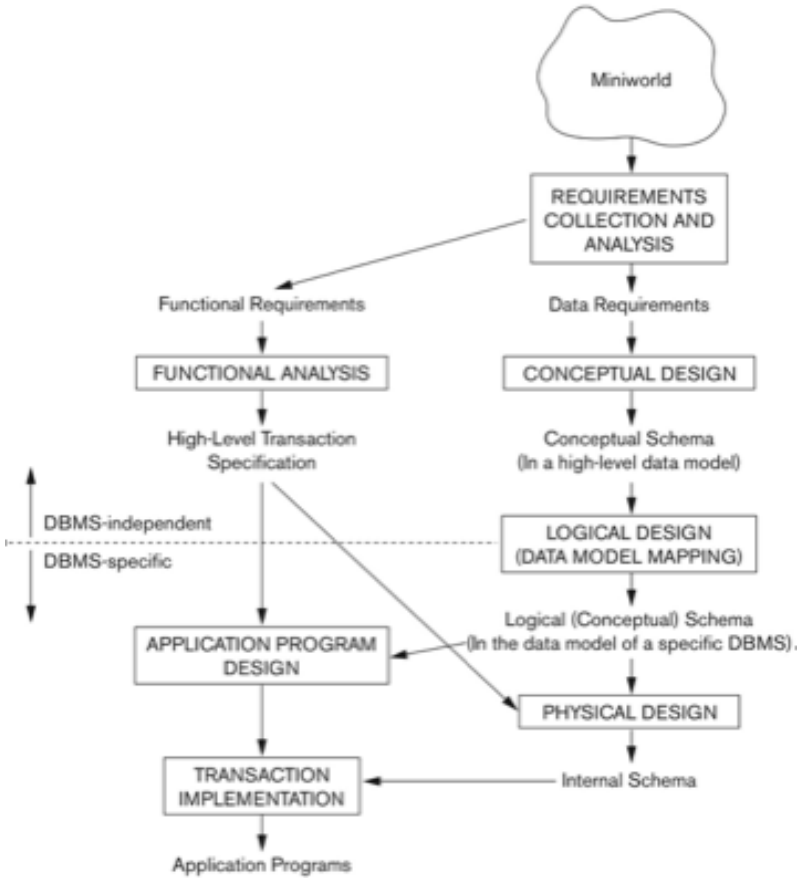
۳ مدل داده‌ای نهاد- ارتباط (ER)

۳-۱ مدل‌سازی داده به کمک مدل‌های سطح بالا

در نظر بگیرید که با یک پروژه پایگاه داده مواجه شده‌اید. به‌طور معمول کارفرما به عنوان ورودی اولیه، خلاصه‌ای از توصیف سیستم ارائه می‌کند. این توصیف که شامل کلیاتی از سیستم است، Mini world نامیده می‌شود. رویه ارائه شده در شکل زیر برای مدل‌سازی داده‌های سیستم باید دنبال شود.



البته در فرایند تولید نرم‌افزار باید فعالیت‌های دیگری نیز اجرا شوند که مجال ارائه آن‌ها در این مستند نمی‌باشد. آن‌چه که در شکل نشان داده شده است، تنها بخش طراحی پایگاه داده سیستم می‌باشد. در شکل زیر خلاصه‌ای از کل روند تولید نرم‌افزار شامل طراحی تراکنش‌ها ارائه شده است.



ارائه یک مدل از پایگاهی که می‌خواهیم بسازیم در بالاترین سطح انتزاع مدل‌سازی معنایی داده نام دارد که روش معمول و رایج روش مدل ER[۱۷] می‌باشد. روش‌هایی مانند UML و OMT نیز برای مدل سازی به کار می‌روند که البته خاص DB نیستند.

مدل داده‌ای نهاد- ارتباط (ER) بر مبنای ادراکی از دنیای حقیقی استوار است که مشتمل بر مجموعه‌ای از اشیاء با نام نهاد و ارتباطات موجود میان این نهادها می‌باشد. در واقع هر شیء قابل تمایز از اشیاء دیگر که مستقل بوده و می‌تواند واقعی یا مجازی باشد یک نهاد است. به عنوان مثال در سیستم بانکی به هر فرد می‌توان به عنوان یک Entity نگاه کرد و یا حساب بانکی را هم می‌توان یک Entity دیگر در نظر گرفت.

این مدل داده‌ای یعنی مدل ER به منظور فراهم نمودن امکان طراحی DB به وسیله مشخصات شمای یک سازمان جامع که نشان دهنده ساختار منطقی کلی یک DB باشد، ابداع و ارائه گردیده است.

مدل ER مدل فوق‌العاده مفید و کارایی در نگاشت معانی ارتباط گسترده دنیای واقعی به شمای مفهومی است.

---

[۱۷] Entity Relationship

۳-۲ مثالی از پایگاه داده

به عنوان یک مثال سیستم یک شرکت را در سراسر این جزوه در نظر می‌گیریم. در نظیر بگیرید که می‌خواهیم سیستم پایگاه داده را برای یک شرکت تولید کنیم. توصیف کلان این شرکت به‌صورت زیر است.

- شرکت از تعدادی دپارتمان تشکیل شده است. هر دپارتمان یک نام واحد، یک شماره واحد و یک کارمند خاصی دارد که آن دپارتمان را مدیریت می‌کند (هر دپارتمان یک مدیر دارد.) برای ما مهم است که تاریخ شروع مدیریت مدیر هر دپارتمان را نگهداری کنیم. یک دپارتمان ممکن است چندین آدرس داشته باشد.
- هر دپارتمان تعدادی پروژه را رهبری می‌کند که هر یک از آن پروژه‌ها یک نام واحد، یک شماره واحد و یک آدرس دارند.
- شرکت تعدادی کارمند دارد. هر کارمند نام، شماره شناسنامه، آدرس، حقوق، جنسیت و تاریخ تولد دارد. هر کارمند وابسته به یک دپارتمان می‌باشد اما ممکن است هم زمان روی چندین پروژه کار کند. آن‌چه که برای ما مهم است تعداد ساعات کاری وی روی هر پروژه می‌باشد. هر کارمند استخدام یک دپارتمان می‌شود و یک مدیر مستقیم دارد.
- مشخصات افراد خانواده هر کارمند (وابسته‌های هر کارمند) برای لیست بیمه مهم می‌باشد. برای هر وابسته هم نام، جنسیت، تاریخ تولد و ارتباط (نسبت) وی با کارمند مهم است.

۳-۳ نوع موجودیت، صفات و کلیدها

۳-۳-۱ موجودیت‌ها و صفات

موجودیت (Entity) مفهوم کلی است در مورد هر آن‌چه که می‌خواهیم در مورد آن در DB اطلاع داشته باشیم. هر شیء یا چیزی که قابل تمایز و تشخیص از سایر اشیاء باشد، موجودیت است. موجودیت باید وجود مستقل در سیستم داشته باشد و چندین نمونه از آن در سیستم وجود داشته باشد. هر موجودیتی یک سری صفات (attribute) دارد که به وسیله آن‌ها شناخته می‌شود.

هر فرد در یک جامعه یک موجودیت است. هر موجودیت دارای مجموعه‌ای از ویژگی‌ها و مقادیری برای آن‌هاست که برخی از این مقادیر ممکن است مشخصه منحصر به فرد آن موجودیت باشند و آن را به صورت یکتا مشخص و متمایز نماید. مانند وام اعطا شده توسط بانک که یک موجودیت است و شماره وام موجود در یک شعبه به صورت منحصر به فرد مشخص کننده آن موجودیت است. بنابراین یک موجودیت می‌تواند حقیقی مانند انسان، کتاب و ... و یا مانند وام یک مفهوم انتزاعی باشد.

باید توجه داشت که اولین مرحله برای مدل‌سازی یک سیستم با استفاده از مدل ER، یافتن موجودیت‌های آن سیستم است. هر موجودیت شامل چندین صفت است که آن موجودیت را توصیف می‌کنند.

صفت (ویژگی) یک خاصیت توصیفی است که همه اعضای مجموعه موجودیت دارای آن می‌باشند. ساختار ذخیره اطلاعات موجودیت‌ها یکسان است ولی هر ویژگی (Attribute) ممکن است مقادیر مخصوص به خود را داشته باشد. مثلاً همه مشتریان دارای صفات نام و کد ملی هستند اما صفت نام برای یک مشتری دارای مقدار خاص خواهد بود حال آن‌که برای مشتری دیگر ممکن است مقدار دیگری داشته باشد و همین طور کد ملی.

در سطح انتزاعی هر موجودیت شامل نام، معنا و دامنه (Domain) می‌باشد. در سطح منطقی و فیزیکی هر صفت شامل کد نمایش، نوع، مقدار صفت طول و محدودیت‌هایی می‌باشد.

انواع صفات به‌صورت زیر دسته‌بندی می‌شوند.

- صفات ساده (simple) و مرکب (composite)

یک صفت ساده قابل تقسیم شدن به بخش‌های جزئی‌تر نیست. مانند شهر مشتری که تنها حاوی نامی است. به‌عبارت دیگر این صفت تجزیه‌پذیر نیست و اتمیک می‌باشد.

صفت مرکب قابل شکسته شدن به چند بخش جزئی دیگر (که هر یک می‌توانند صفت باشند) است. مانند صفت نام مشتری که می‌تواند حاوی اسم و فامیل مشتری باشد.

ساده یا مرکب بودن یک صفت نسبی است یعنی ممکن است یک صفت مرکب در محیط دیگری صفت ساده باشد.

- صفات تک مقداری (single- valued) و چند مقداری (multi- valued)

صفاتی را که به ازای هر موجودیت دارای یک مقدار واحد باشند یا به ازای یک نمونه موجودیت حداکثر یک مقدار می‌گیرند و می‌توانند Null هم باشند، تک مقداری می‌گویند. مانند شماره دانشجویی، کد ملی.

صفاتی را که ممکن است به ازای حداقل یک موجودیت بیش از یک مقدار به آن‌ها تخصیص یابد، چند مقداری می‌نامند. مانند شماره تلفن مشتری که بسته به این‌که مشتری دارای چند خط تلفن باشد ممکن است این صفت برای برخی مشتریان تک مقداری، دو یا بیش‌تر و برای برخی از آن‌ها تهی باشد.

- صفات تهی (Null)

یک مقدار Null زمانی مورد استفاده قرار می‌گیرد که یک موجودیت فاقد یک مقدار برای یک صفت خاص باشد. به عنوان مثال اگر یک مشتری بانک تلفن نداشته باشد به صفت شماره تلفن به ازای آن مشتری مقدار Null تخصیص می‌یابد. در مواردی هم که مقادیر یک صفت ناشناخته باشد می‌توانیم از Null استفاده کنیم. خواه این صفت دارای مقدار باشد و ما مقدار آن را ندانیم و یا این‌که ندانیم که صفت مقداری دارد یا فاقد مقدار است.

- صفات ذخیره شده (Stored) و مشتق شده (Derived)

صفتی که مقدارش ذخیره نشده اما قابل محاسبه از طریق دیگر صفات می‌باشد را مشتق شده می‌گویند. یعنی مقدار آن را می‌توان از مقادیر سایر صفات و یا موجودیت‌های مرتبط به دست آورد. مانند معدل دانشجو.

۲-۳-۳ کلید، دامنه و نوع موجودیت

مجموعه موجودیت به مجموعه‌ای گفته می‌شود که دارای موجودیت‌هایی از یک نوع و با ویژگی‌های یکسان باشند مانند مشتریان یک بانک. مجموعه موجودیت‌های مختلف جدا از هم نیستند. مثلاً در مثال بانک، ما یک مجموعه موجودیت داریم که شامل کارکنان بانک است و مجموعه موجودیت دیگری داریم که مشتریان بانک را تشکیل می‌دهد. یک فرد ممکن

است به هیچ یک از دو مجموعه تعلق نداشته باشد و یا فردی باشد که به هر دو مجموعه تعلق داشته باشد. برای نمونه مشتری که از کارکنان بانک باشد.

موجودیت‌ها بر دو نوع هستند:

- موجودیت قوی (مستقل): وابستگی وجودی به انواع دیگر ندارد و خود مستقلاً مطرح است مثلاً در محیط دانشگاه دانشجو، استاد، درس به عنوان موجودیت قوی می‌باشند.
- موجودیت ضعیف (وابسته): وجودش وابسته به نوعی دیگر است به گونه‌ای که اگر آن نوع دیگر نباشد موجودیت ضعیف هم دیگر معنایی ندارد. مانند کارمندی که در اداره کار می‌کند و مدیریت می‌خواهد اطلاعاتی از خانواده کارمند داشته باشد. خانواده کارمند موجودیت ضعیف است زیرا اگر کارمند نباشد موجودیت اعضای خانواده‌اش نیز دیگر مطرح نمی‌شود.

هر موجودیت با مجموعه‌ای از صفات نشان داده می‌شود.

نحوه تشخیص و تمایز یک موجودیت موجود در یک مجموعه موجودیت و یک ارتباط از یک مجموعه ارتباط مساله مهم و قابل توجهی است. اگرچه به صورت مفهومی موجودیت‌ها و ارتباطات جداگانه قابل تمایز هستند اما از دیدگاه پایگاه داده، تفاوت میان آن‌ها باید از طریق ویژگی‌هایشان قابل استنتاج باشد.

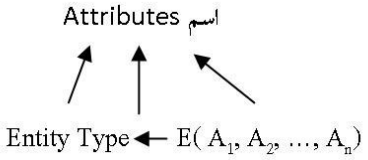
یک کلید (Key) مجموعه‌ای از یک یا چند صفت است که با هم یک موجودیت از مجموعه موجودیت را به صورت منحصر به فرد مشخص می‌کنند. به عنوان مثال شماره ملی افراد کلیدی برای مجموعه موجودیت اشخاص است. در حالی که نام به تنهایی نمی‌تواند به عنوان کلید در نظر گرفته شود زیرا افراد زیادی هستند که دارای نام مشابه می‌باشند.

کلید جزئی (Partial Key) صفتی است که یکتایی مقدار دارد اما تنها در مجموعه نمونه‌های نوع ضعیف وابسته به یک نمونه قوی و نه در مجموعه تمام نمونه‌های ضعیف این خاصیت برقرار است.

صفت چندمقداری را به ویژه اگر مرکب باشد، می‌توان با مفهوم نوع موجودیت ضعیف مدل‌سازی کرد، اما عکس این مطلب چون باعث کاهش انعطاف‌پذیری مدل‌سازی می‌شود توصیه نمی‌شود.

به ازای هر صفت، مجموعه‌ای از مقادیر مجاز و ممکن وجود دارند که تخصیص آن‌ها به آن صفت امکان‌پذیر است. این مجموعه را دامنه (Domain) و یا مجموعه مقادیر می‌نامند . مثلاً دامنه صفت نام مشتری مجموعه‌ای از رشته‌های تشکیل شده از حروف الفباست که دارای حداکثر طول معینی هستند. دامنه مفهومی مهمی در مدل رابطه‌ای است اما در مدل ER در مورد آن بحث نمی‌شود.

یک صفت از مجموعه موجودیت، تابعی است که از مجموعه موجودیت به یک دامنه نگاشت می‌شود. از آن‌جا که هر مجموعه موجودیت می‌تواند دارای چندین صفت باشد، می‌توان هر موجودیت را با یک مجموعه از زوج‌های (مقدار، صفت) توصیف کرد.

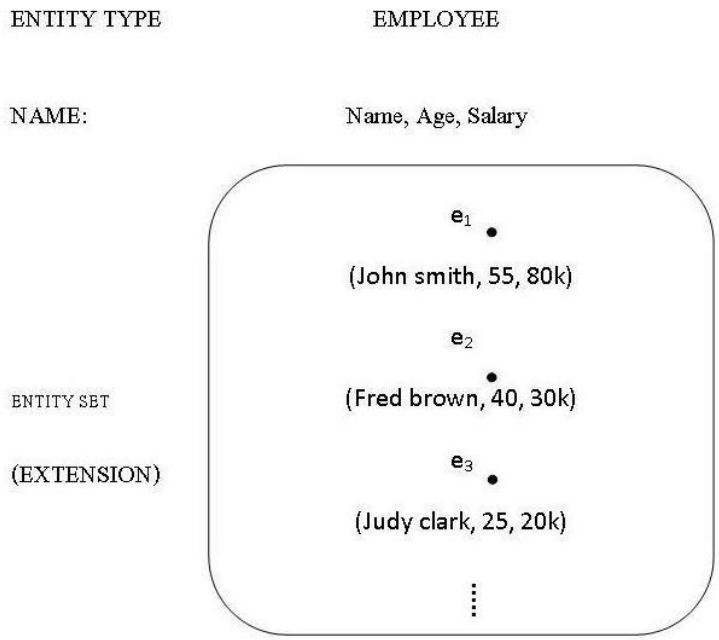


با فرض این‌که نوع موجودیت E به‌صورت E (A_1, A_2, …, A_n) تعریف شده و برای هر صفت A_i دامنه به‌صورت Domain (A_i) نمایش داده می‌شود. آن‌گاه تعداد کل نمونه موجودیت‌ها برای این نوع موجودیت E از رابطه زیر به دست می‌آید:

[تعداد عناصر Domain A_n] *…* [تعداد عناصر Domain A_2]*[تعداد عناصر Domain A_i]

نوع موجودیت (Entity Type) به‌صورت مجموعه‌ای از موجودیت‌هایی که صفات مشابه دارند تعریف می‌شود.

در شکل زیر نوع موجودیتی به نام EMPLOYEE و لیستی از صفات آن نشان داده شده است. مجموعه تمام موجودیت‌های یک نوع موجودیت خاص یک DB در هر لحظه از زمان یک مجموعه موجودیت نامیده می‌شود که معمولاً به عنوان همان نوع موجودیت تعبیر می‌شود.

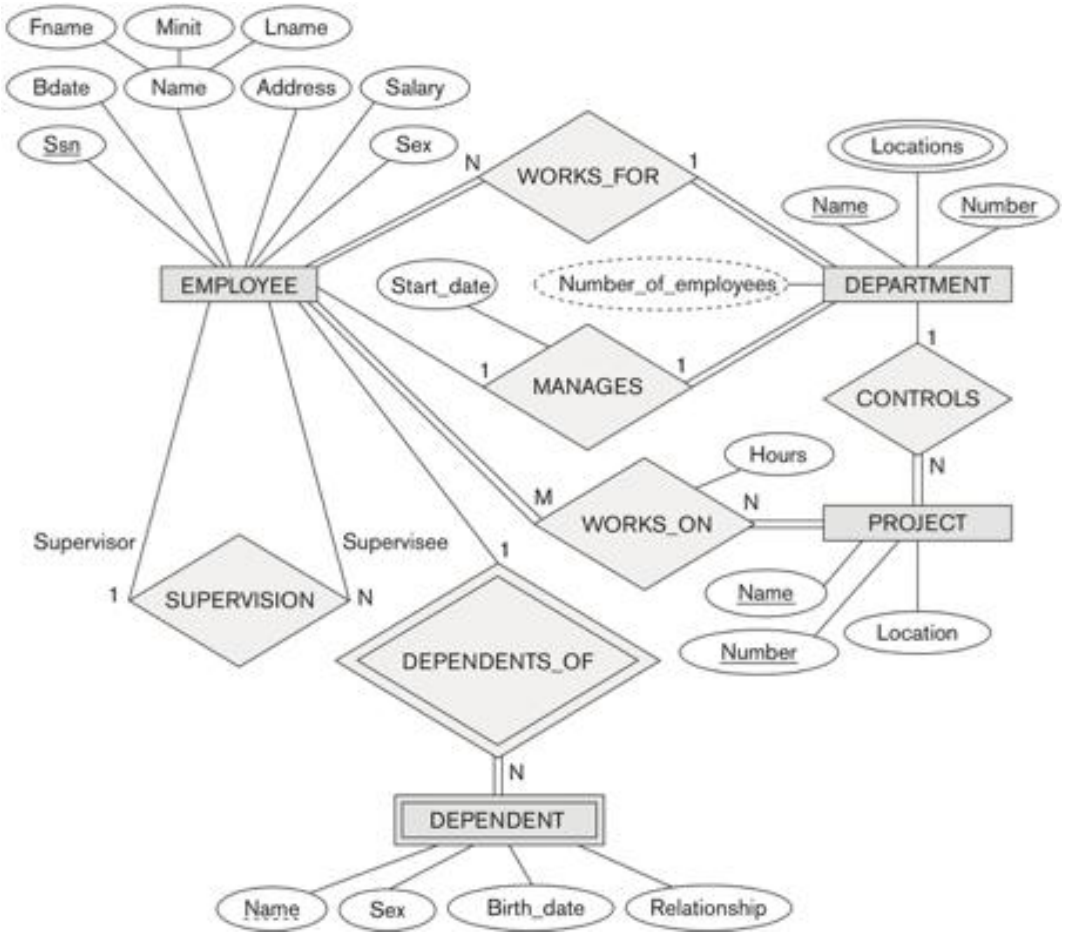


تا کنون مباحث مربوط به موجودیت‌ها در مدل ER ارائه شد. بنابراین برای هر سیستم می‌توانید مجموعه موجودیت‌ها را استخراج نمایید. خروجی مدل ER که شامل موجودیت‌ها، صفات و ارتباطات است، به‌صورت یک نمودار نمایش داده می‌شود که به نمودار ER (ERD[۱۸]) معروف است. به‌طور کلی موجودیت‌ها به‌صورت مستطیل در ERD نشان داده می‌شوند.

به‌عنوان مثال خروجی مدل ER (نمودار ERD) برای Company در شکل زیر ترسیم شده است. البته این نمودار شامل ارتباطات نیز می‌باشد که در بخش بعدی مورد بررسی قرار خواهد گرفت. در این نمودار موجودیت‌های EMPLOYEE، DEPARTMENT، PROJECT و DEPENDENT تشخیص داده شده‌اند. موجودیت DEPENDENT یک موجودیت ضعیف است که توسط موجودیت EMPLOYEE قوی می‌شود.

همان‌طور که گفته شد، صفات را می‌توان از دیدگاه‌های مختلف دسته‌بندی نمود. هر کدام از این دسته‌ها باید در نمودار ERD نشان داده شوند. برای نمونه در نمودار ERD مربوط به Company، برخی از صفات از جمله صفت SSN در EMPLOYEE کلید هستند. این صفات به‌صورت زیر خط دار نشان داده شده‌اند. همچنین صفات چند مقداری به‌صورت بیضی دو خط نمایش داده می‌شوند. صفت Name در موجودیت EMPLOYEE یک صفت مرکب است که اجزای آن نشان داده شده است. همچنین صفت NumberOfEmployee در موجودیت DEPARTMENT مشتق شده است و به‌صورت نقطه‌چین نشان داده شده است.

[۱۸] Entity Relationship Diagram



۳-۴ ارتباط و نوع ارتباط

۳-۴-۱ نوع و مجموعه نمونه ارتباط

ارتباط (Relationship) بین چندین موجودیت تعریف می‌شود. مثلاً می‌توان ارتباطی تعریف کرد که یک مشتری خاص را با یک وام خاص مرتبط نماید. این ارتباط بیانگر آن است که مشتری خاصی آن وام را دریافت کرده است.

هر نوع ارتباط یک معنای مشخص دارد و با یک نام بیان می‌شود و در واقعاً ارتباطی است که بین موجودیت‌ها وجود دارد. بین دو موجودیت می‌تواند بیش از یک ارتباط وجود داشته باشد. ارتباط را می‌توان فعل یا عمل بین m تا موجودیت که $m \geq 1$ معرفی کرد.

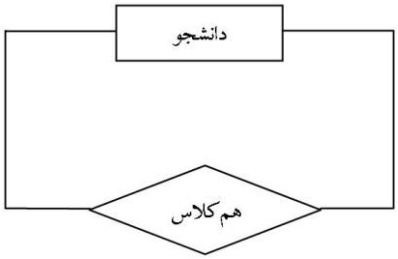
ارتباط می‌تواند بین m موجودیت که $m \geq 1$ است، وجود داشته باشد. اگر $m=1$ باشد، این نوع ارتباط بازگشتی نامیده می‌شود. ارتباطات در واقع بین نوع موجودیت‌ها برقرار نمی‌شوند بلکه بین نمونه‌های موجود ارتباط برقرار می‌شود.

۳-۴-۲ درجه ارتباط و نقش موجودیت

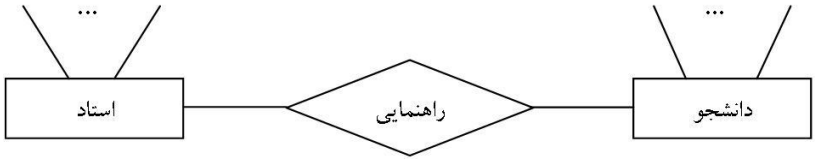
در صورتی که ارتباطی بین چند موجودیت تعریف شود، موجودیت‌های مربوطه در آن ارتباط مشارکت می‌کنند. تعداد مجموعه موجودیت‌هایی را که بر روی یک مجموعه ارتباط مشارکت دارند، درجه (Degree) مجموعه ارتباط مزکور می‌نامند.

درجه ارتباط برابر با تعداد موجودیت‌هایی است که در آن ارتباط شرکت دارند. در مدل ER درجه ارتباط عددی صحیح و کوچک‌تر از ۵ است. ارتباط درجه ۴ کمیاب و غیرمعمول است و ارتباط بالاتر از درجه ۴ غیر قابل رسم است. اما ارتباط‌های ۱، ۲ و ۳ معمول می‌باشند. در ادامه برای هر کدام از این نوع ارتباطات نمونه‌ای ارائه می‌شود.

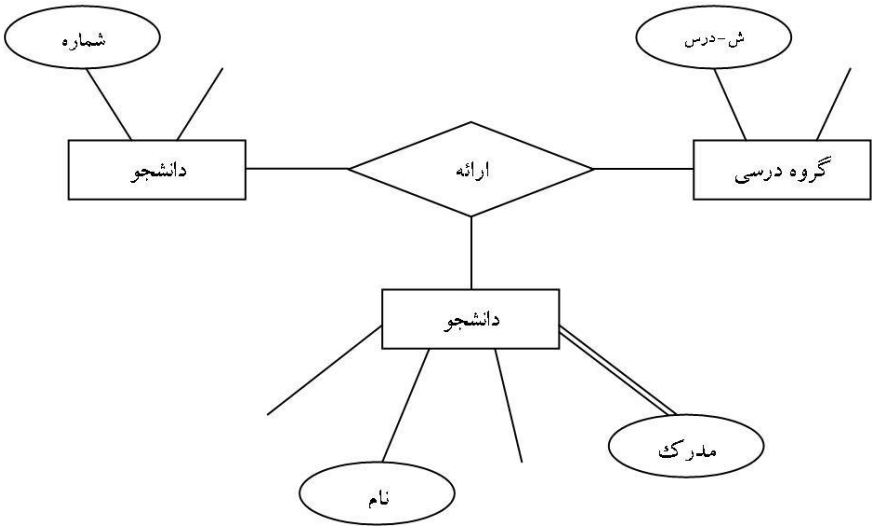
- ارتباط درجه یک یا بازگشتی: ارتباط داشتن یک موجودیت با خودش تشکیل این ارتباط را می‌دهد. برای نمونه ارتباط دانشجو با همکلاسی خود که یک دانشجو است، یک ارتباط بازگشتی است و به‌صورت زیر نشان داده می‌شود.



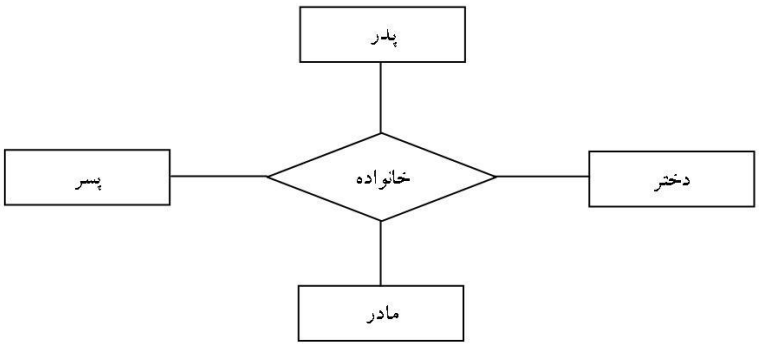
- ارتباط درجه ۲: در این نوع ارتباط دو نوع موجودیت متفاوت درگیر می‌شوند. ارتباط بین دانشجو و استاد راهنما نمونه‌ای از ارتباط دوتایی است که به‌صورت زیر نشان داده شده است.



- ارتباط درجه ۳: در این نوع ارتباط س موجودیت متفاوت شرکت می‌کنند. این ارتباط را ۳تایی نیز می‌نامند و به‌عنوان نمونه در زیر یک ارتباط ۳تایی نشان داده شده است.



- ارتباط درجه ۴: در این نوع ارتباط چهار نوع موجودیت شرکت دارند. مثالی از این نوع ارتباط به‌صورت زیر است.



همواره می‌توان به جای یک ارتباط چند تایی از تعدادی دوتایی جداگانه استفاده کرد. یک ارتباط nتایی مجموعه موجودیت‌هایی را که بر روی یک ارتباط مشارکت دارند واضح‌تر نشان می‌دهد. در طراحی متناظری که با ارتباط‌های دوتایی انجام می‌شود تشخیص این مشارکت دشوار است.

نقش (Role) به کارکرد هر موجودیت در یک ارتباط گفته می‌شود که در نمودار ERD، نقش را روی خطوط اتصالی ارتباط و موجودیت‌ها می‌نویسند. یک نوع موجودیت می‌تواند در ارتباطات مختلف نقش‌های متفاوت داشته باشد. مفهوم نقش می‌تواند در تکمیل مدل ER کمک نماید.

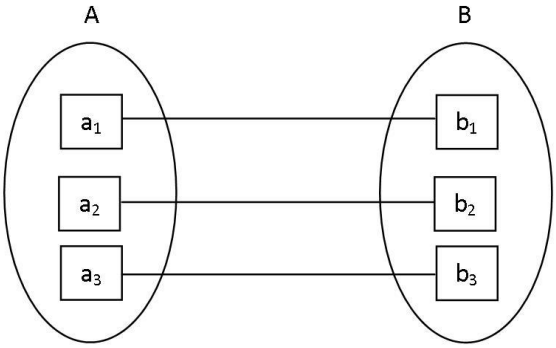
۳-۵ محدودیت‌های ارتباط

هر ارتباط در مدل ER شامل محدودیت کاردینالیتی (Cardinality) و نوع مشارکت می‌باشد. این محدودیت‌ها جزء اصلی یک ارتباط می‌باشند که باید در مدل ER برای هر ارتباط مشخص شده باشند.

۳-۵-۱ کاردینالیتی

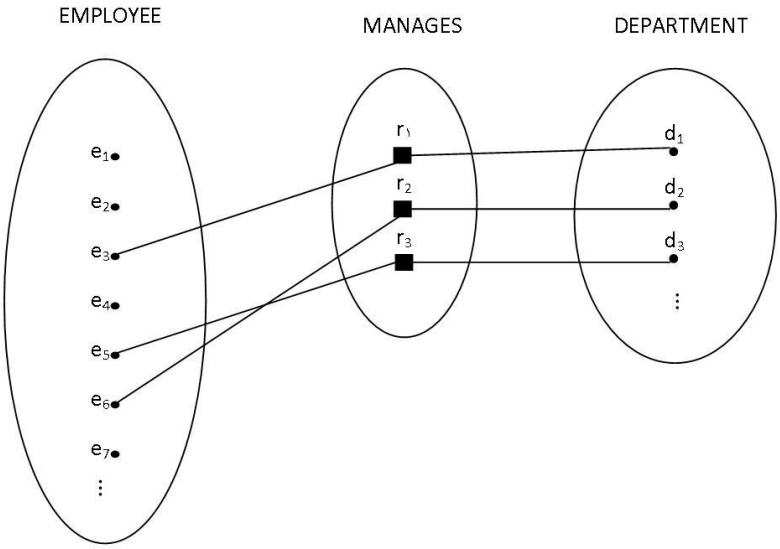
هر نوع ارتباط از نظر نوع اتصال یا تعداد مشارکت نمونه موجودیت‌های آن ارتباط بر سه نوع 1-1، 1-N و N-M تقسیم می‌شود. این خاصیت ارتباط را کاردینالیتی آن ارتباط می‌گویند. در ادامه برای هر نوع ارتباط از این نوع نمونه‌ای ارائه می‌شود. در نمونه‌های زیر فرض کنید که ارتباط بین دو موجودیت A و B برقرار است.

در ارتباط یک به یک (one-to-one) هر نمونه موجودیت از نوع موجودیت A حداکثر با یک نمونه موجودیت از نوع موجودیت B ارتباط دارد و هر نمونه موجودیت از نوع موجودیت B حداکثر با یک نمونه موجودیت از نوع موجودیت A ارتباط دارد.

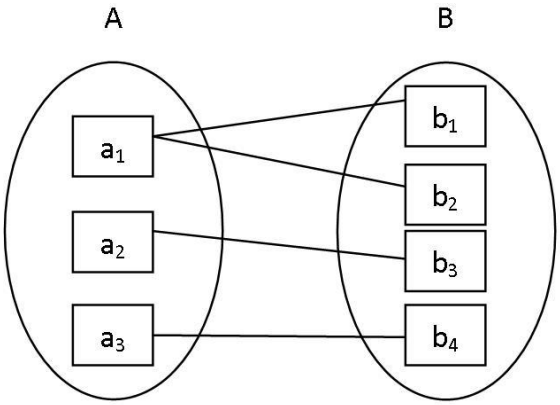


در یک دانشگاه هر استاد یک کامپیوتر اختصاصی دارد. بنابراین این ارتباط یک به یک است.

در مثال Company ارتباط بین موجودیت‌های EMPLOYEE و DEPARTMENT نمونه‌ای از ارتباط یک به یک می‌باشد. در این ارتباط هر دپارتمان یک مدیر دارد و هر کارمند می‌تواند حداکثر مدیر یک دپارتمان باشد.

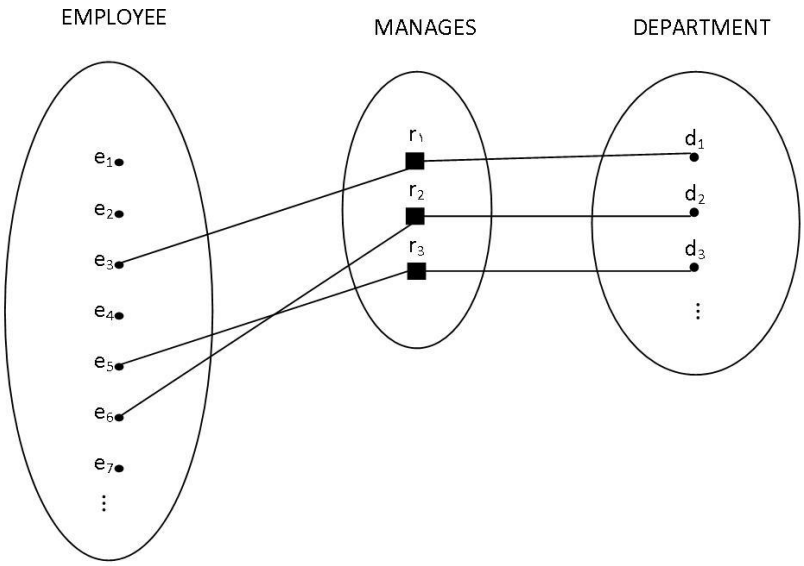


در ارتباط یک به چند (one-to-many)، یک نمونه موجودیت از A می‌تواند با هر تعداد از نمونه موجودیت‌های B ارتباط برقرار کند ولی یک نمونه موجودیت از B حداکثر با یک نمونه موجودیت از A ارتباط دارد.

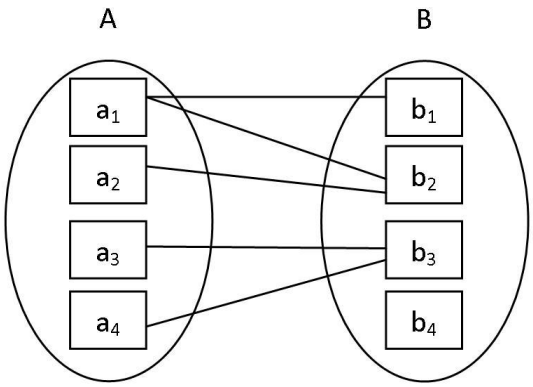


برای نمونه یک استاد می‌تواند چندین دانشجو راهنمایی می‌کند ولی هر دانشجو تنها یک استاد راهنما دارد.

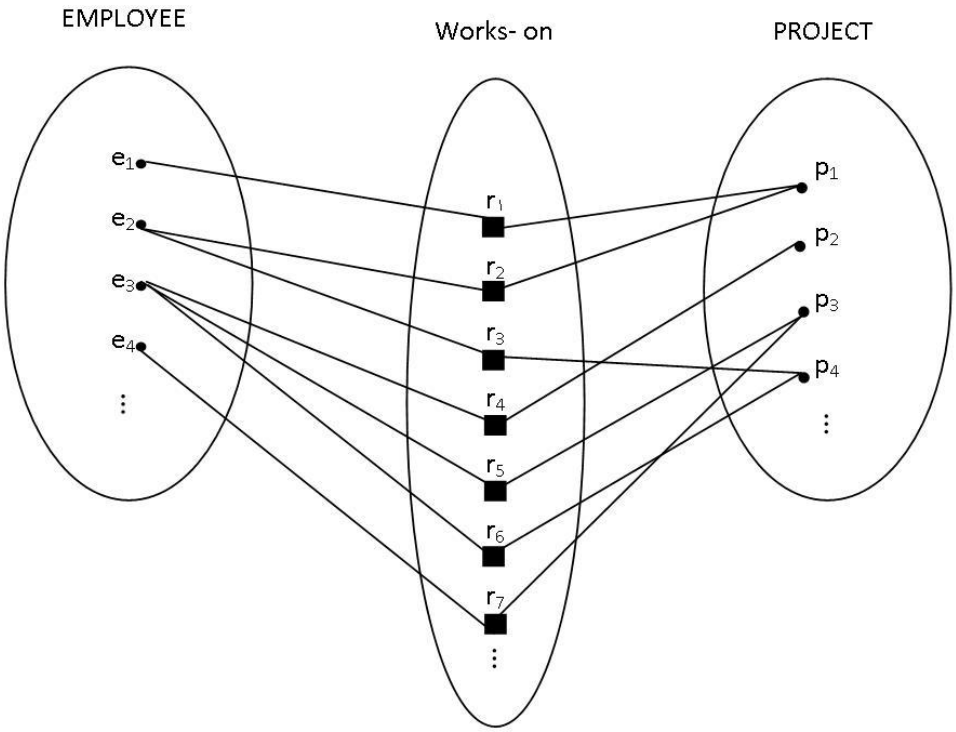
در مثال Company، ارتباط بین کارمندان و دپارتمان برای استخدام شدن یک ارتباط یک به چند است. در این سیستم هر کارمند تنها برای یک دپارتمان استخدام می‌شود ولی هر دپارتمان می‌تواند بیش از یک کارمند را استخدام نماید.



در ارتباط چند به چند (many-to-many)، هر نمونه موجودیت از A می‌تواند با هر تعداد نمونه موجودیت از B مرتبط شود و یک نمونه موجودیت از B هم می‌تواند با هر تعداد دلخواهی از نمونه موجودیت‌های A ارتباط داشته باشد. یک استاد ممکن است بیش از یک درس ارائه دهد و یک درس ممکن است توسط بیش از یک استاد ارائه شود. بنابراین این ارتباط یک ارتباط چند به چند است.



در سیستم Company ارتباط بین کارمند و پروژه از نوع چند به چند است. این ارتباط نشان می‌دهد که هر کارمند می‌تواند در بیش از یک پروژه کار کند و هر پروژه می‌تواند بیش از یک کارمند را داشته باشد.



همان‌طور که شرح داده شد، کاردینالیتی نشان دهنده تعداد نمونه موجودیت‌های دو نوع موجودیت است که می‌توانند از طریق یک مجموعه ارتباط با یکدیگر ارتباط برقرار نمایند. به‌عبارت دیگر این خاصیت مربوط به نمونه موجودیت‌های ارتباط می‌باشد و برای تشخیص آن نباید به نوع موجودیت مراجعه نمود.

۳-۵-۲ مشارکت

موجودیت‌های درگیر در یک ارتباط می‌توانند مشارکت متنوعی در آن ارتباط داشته باشند. برای مشارکت هر موجودیت در یک ارتباط دو روش کلی (Total) و جزئی (Partial) وجود دارد.

- مشـارک کلی (Total participation): مشـارکت یک نوع موجودیت مانند E را در یک ارتباط مانند R مشـارکت کلی می‌گویند اگر هر نمونه موجودیت در E حداقل در یک ارتباط R مشارکت نماید.
- مشـارکت جزئی (Partial participation): در این نوع مشـارکت برخی از نمونه موجودیت‌ها ممکن اسـت در هیچ ارتباطی از مجموعه ارتباطات مشـارکت نداشـته باشند که در این صورت مشارکت نوع موجودیت E در R را مشارکت جزئی می‌نامند.

مشارکت کامل را در نمودار ER با دو خط و مشارکت جزئی را با یک خط نشان می‌دهند.

۳-۶ موجودیت ضعیف

یک موجودیت ممکن است فاقد صفات کافی برای تشکیل یک کلید اصلی باشد. به عبارت دیگر ممکن است هیچ ترکیبی از صفات یک نوع موجودیت قادر نباشند که یک نمونه موجودیت از آن مجموعه را به صورت منحصر به فرد مشخص نمایند. در این شـرایط وجود هر نمونه در یک موجودیت وابسـته به وجود یک نمونه در موجودیت دیگر اسـت. چنین نوع موجودیتی را موجودیت ضعیف می‌نامند. موجودیت قوی، موجودیتی است که دارای کلید اصلی باشد.

مثلاً اگر بازپرداخت را به عنوان یک موجودیت در نظر بگیریم که دارای سه صفت شماره قسط، میزان قسط و تاریخ پرداخت اسـت. از آن‌جایی که بازپرداخت چندین وام دارای شـماره قسـط مشـترک اسـت، فاقد کلید اصـلی اسـت و بنابراین یک نوع موجودیت ضـعیف اسـت. یا در مثال Company چون موجودیت DEPENDENT فاقد کلید اصـلی اسـت، یک موجودیت ضعیف می‌باشد.

اگرچه یک نوع موجودیت ضـعیف فاقد کلید اصـلی اسـت ولی لازم اسـت بتوانیم میان موجودیت‌های مختلف موجود در این موجودیت تمایز قائل شـویم. مشـخصـه یک موجودیت ضـعیف مجموعه‌ای از صـفات اسـت که باعث به وجود آمدن این تمایز می‌گردد.

مشـخصـه موجودیت ضعیف، کلید جزئی (Partial Key) آن موجودیت نامیده می‌شود. معمولاً هر موجودیت ضعیف در مدل ER توسط یک ارتباط به یک موجودیت قوی متصل می‌شود. این ارتباط خاص را Identifying Relationship می‌گویند.

موجودیت‌های ضعیف در یک نمودار ER با مستطیل دوتایی نشان داده می‌شوند و کلید آن‌ها هم با نقطه‌چین نمایانگر است.

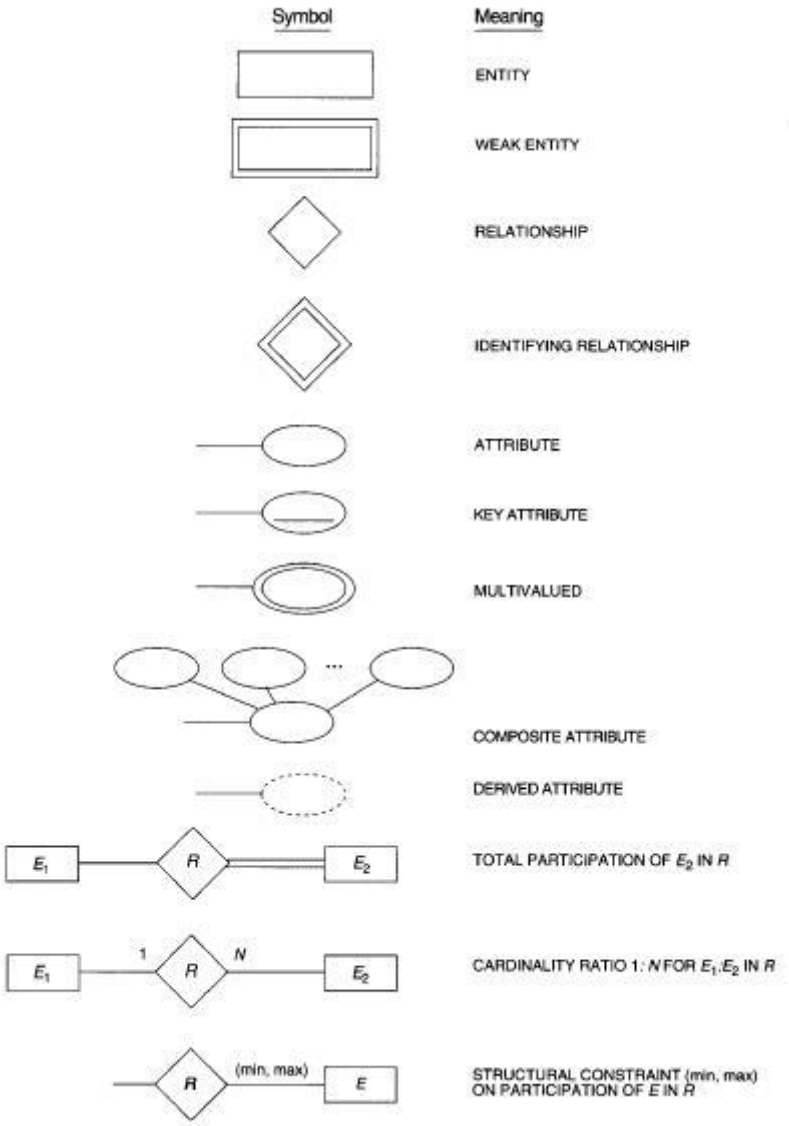
۳-۷ قراردادهای نام‌گذاری و طراحی ER

سـاختـار منطقی کلی یـک DB را می‌توانیم در قـالب نمودار ERD ترسـیم کنیم. نمودار ERD از بخش‌های گرافیکی زیر تشکیل شده است.

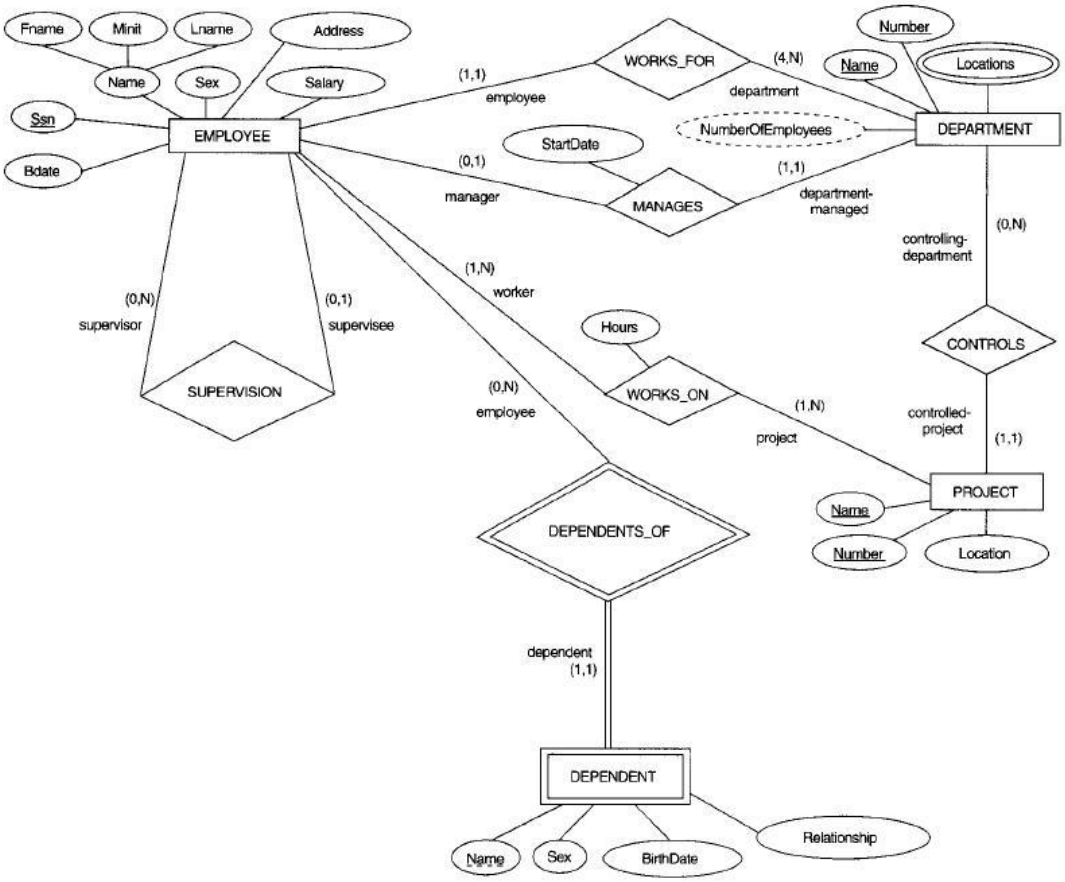
- مستطیل‌ها نشان دهنده موجودیت‌ها در ER هستند.
- بیضی مشخص کننده یک صفت است.
- لوزی‌ها روابط را در نمودار ER نشان می‌دهند.

- خطوط برای اتصـال و انتصـاب صـفات به موجودیت‌ها و ارتباطات به کار می‌روند. علاوه بر آن با اسـتفاده از خطوط، موجودیت‌های شرکت کننده در یک ارتباط به آن وصل می‌شوند.
- بیضی‌های دو خطی نشان دهنده صفات چند مقداری هستند.
- بیضی‌های خط چین شده بیانگر صفات مشتق شده هستند.
- صفاتی از یک موجودیت که کلید اصلی هستند، زیر خطدار می‌باشند.

در ادامه خلاصه‌ای از تمامی علائم موجود در نمودار ERD دیده می‌شود.



روش دیگر نمایش تعداد مشـارکت یک موجودیت در یک ارتباط به صـورت (min, max) می‌باشـد که مزیت آن این اسـت که کاردینالیتی و نوع مشـارکت را با هم نمایش می‌دهد. اگر min=0 نوع مشـارکت جزئی اسـت و اگر min>0 نوع شـرکت کلی می‌باشـد. همچنین در این نوع نمـایش می‌توان روی حداقل و حداکثر تعداد مشـارکت نیز محدودیت تعریف نمود. مدل ER برای Company با این نمایش به‌صوت زیر می‌باشد.


Fname
Minit
Lname
Address
Name
Sex
Salary
Ssn
Bdate
EMPLOYEE
WORKS_FOR
(1,1)
employee
(4,N)
department
Number
Name
Locations
DEPARTMENT
NumberOfEmployees
StartDate
(0,1)
manager
MANAGES
(1,1)
department-managed
(0,N)
controlling-department
CONTROLS
(1,N)
worker
(0,N)
supervisor
(0,1)
supervisee
SUPERVISION
Hours
WORKS_ON
(0,N)
employee
(1,N)
project
controlled-project
(1,1)
PROJECT
Name
Number
Location
DEPENDENTS_OF
dependent
(1,1)
DEPENDENT
Name
Sex
BirthDate
Relationship

۴ مدل گسترش یافته ER

۴-۱ جنبه‌های توسعه ER

جنبه‌های گسترش یافته ارائه شده برای ER برای مدل کردن اکثر جنبه‌های پایگاه داده کافی هستند، اما جنبه‌هایی از پایگاه داده وجود دارند که برای توصیف و مدل نمودن آن‌ها نیازمند گسترش برخی از جنبه‌های ER هستیم.

در این فصل در ارتباط با جنبه‌های گسترش یافته ER مثل تخصیص یا اختصاصی کردن (Specialization) تعمیم (Generalization)، تجمع (Aggregation) توضیح داده خواهد شد.

در مدل Enhanced ER یک سری مفاهیم شئ‌گرا به مدل ER افزوده شده و مفاهیم موجود در ER نیز در برخی از موارد توسعه داده شده است

۴-۲ تخصیص، تعمیم و ارث‌بری

۴-۲-۱ تخصیص

هر مجموعه موجودیت یا همان نوع موجودیت ممکن است شامل زیرمجموعه‌ای از موجودیت‌ها باشد که به طریق خاصی از سایر موجودیت‌های موجود در این مجموعه متمایز شده باشند. به عنوان مثال ممکن است دسته‌ای از موجودیت‌های موجود در یک نوع موجودیت شامل صفت یا صفاتی باشد که سایر موجودیت‌های موجود در آن مجموعه فاقد آن هستند. در چنین مواردی مدل ER ابزاری را برای نشان دادن این تمایز و خاص نمودن این زیرگروه متمایز تدارک نمی‌بیند.

نوع موجودیت حساب بانکی را در نظر بگیرید. حساب بانکی می‌تواند یک حساب قرض‌الحسنه یا یک حساب جاری باشد. هر یک از این دو نوع حساب دارای صفات مشترک یک حساب بانکی یعنی شماره حساب و میزان موجودی هستند و علاوه بر آن صفات اضافه‌تری دارند که مختص هر یک از این انواع حساب است. مثلاً حساب قرض‌الحسنه پس‌انداز می‌تواند دارای ویژگی کارکرد دوره‌ای حساب و حساب جاری می‌تواند شامل صفت گردش مالی باشد.

فرآیند مشخص نمودن زیرگروه‌های متمایز از یک نوع موجودیت، تخصیص نامیده می‌شود. تخصیص فرآیند تولید subclassها از یک class براساس یک معیار مشخص می‌باشد.

یک نوع موجودیت می‌تواند با استفاده از چندین جنبه مختلف تخصیص گردد. در مثال ارائه شده برای حساب بانکی جنبه قابل تشخیص برای تخصیص نوع حساب بود. جنبه دیگری که می‌توان عمل اختصاصی نمودن را بر اساس آن انجام داد، وضعیت صاحبان است. صاحب یک حساب می‌تواند دارای شخصیت حقیقی باشد (حساب شخصی) و یا شخصیت حقوقی داشته باشد (حساب دولتی یا تجاری). هنگامی که بیش از یک عمل تخصیص بر روی یک نوع موجودیت انجام شود یک موجودیت خاص ممکن است به چندین گروه خاص تعلق داشته باشد. مثلاً یک حساب بانکی می‌تواند شخصی و یا تجاری باشد.

در نمودار ER تخصیص به صورت یک مثلث و با برچسب ISA نشان داده می‌شود که به معنای "is a" است و مشخص می‌کند که مثلاً حساب قرض‌الحسنه یک حساب بانکی و یا حساب طلایی یک حساب جاری است. یک ارتباط ISA گاهی با عنوان ارتباطی زیرکلاس (SubClass) و سوپرکلاس (SuperClass) نیز شناخته می‌شود. مجموعه نهادهای سطح بالاتر و سطح پایین‌تر هر دو به صورت موجودیت‌های معمولی در نمودار ER نمایش داده می‌شوند. (مستطیلی حاوی نام موجودیت)

۴-۲-۲ تعمیم

اصلاح نمودن یک مجموعه موجودیت اولیه یا همان Superclass، با مشتق نمودن موجودیت یا همان اشیای سطح پایین‌تر از آن، نشان دهنده یک فرآیند طراحی از بالا به پایین است که در آن جداسازی زیرگروه‌ها به صورت صریح اعمال می‌شود. طراحی می‌تواند به صورت پایین به بالا نیز انجام شود، یعنی چندین موجودیت بر اساس جنبه‌های مشترک در موجودیت سطح بالاتر به اشتراک برسند. مثلاً در طراحی مجموعه موجودیت حساب جاری با در نظر گرفتن سه ویژگی شماره حساب، میزان موجودی و کارکرد دوره‌ای حساب و سپس با توجه به صفات مشترک این دو مجموعه موجودیت، آن‌ها را به یک حالت عمومی‌تر یعنی مجموعه موجودیت حساب بانکی تعمیم دهیم.

تعمیم یک ارتباط شمول است که بین یک مجموعه موجودیت سطح بالاتر با یک یا چند مجموعه موجودیت سطح پایین‌تر (یعنی همان superclass و subclassها) برقرار است. عمل تعمیم عکس عمل تخصیص است. نمودار ER تمایزی میان این دو عمل قائل نمی‌شود.

۴-۲-۳ ارث‌بری

در اصطلاح گفته می‌شود که زیرکلاس‌ها صفات سوپرکلاس را ارث می‌برند. به این مفهوم که مجموعه موجودیت‌های سطح پایین دارای همه ویژگی‌های مجموعه موجودیت سطح بالاتر از خود می‌باشند، ارث‌بری گفته می‌شود. به عنوان مثال حساب جاری با ارث‌بری از superclass حساب بانکی دو صفت دیگر شماره حساب و میزان موجودی را نیز دارد و این علاوه بر ویژگی گردش دوره‌ای حساب است که خودش داشته است.

هم‌چنین یک زیرکلاس مشارکت در یک مجموعه ارتباط را که superclass در آن مشارکت دارد از superclass مزکور به ارث می‌برد. به‌عبارت دیگر ارث‌بری علاوه‌بر مجموعه صفات شامل مجموعه تمامی ارتباطات است.

۴-۳ محدودیت‌های طراحی

ممکن است برخی از طراحان پایگاه داده به منظور مدل کردن هرچه دقیق‌تر یک مساله، محدودیت‌های خاص را بر روی یک تخصیص خاص قرار دهند. یک نمونه از این محدودیت‌ها برای این است که مشخص نمایند چه موجودیت‌هایی می‌توانند عضوی از یک مجموعه موجودیت سطح پایین‌تر باشند. چنین عضویتی می‌تواند یکی از موارد زیر باشد.

- تعریف شده با شرط (condition- defined): در این موارد عضویت در یک مجموعه موجودیت سطح پایین بر اساس برآورده شدن یا نشدن یک شرط صریح مشخص می‌گردد. به عنوان مثال فرض کنید مجموعه موجودیت سطح بالاتر حساب بانکی دارای صفت نوع حساب باشد. می‌توانیم به صورت قراردادی نوع حساب جاری را با عدد ۱ و حساب‌های

پس‌انداز را با عدد ۰ نمایش دهیم. در این صورت تنها موجودیت‌هایی که مقدار صفت نوع حساب آن‌ها برابر ۱ باشد، می‌توانند عضو کلاس حساب جاری باشند. چون در این حالت همه مجموعه نهادهای سطح پایین‌تر با یک صفت ارزیابی می‌شوند، این نوع از تخصیص را تعریف شده توسط ویژگی می‌نامند.

- تعریف شده توسط کاربر (user- defined): در این حالت کاربر مشخص می‌نماید که یک موجودیت یا همان شئ به کدام مجموعه موجودیت متعلق باشد. به عنوان مثال فرض کنید که کارمندان یک بانک سه ماه پس از استخدام به عضویت یکی از چهار تیم موجود در بانک درآیند. در این صورت باید کاربر پایگاه داده به وسیله عملی که یک موجودیت را به یک مجموعه موجودیت می‌افزاید مشخص کند که کارمند به کدام تیم متعلق است. این نوع محدودیت‌ها مشخص می‌کند که در یک عمل تخصیص خاص، یک نمونه موجودیت به بیش از یک مجموعه موجودیت تعلق دارد یا خیر.

مجموعه موجودیت‌های سطح پایین‌تر یکی از دو حالت زیر را دارند:

- جدا از هم (Disjoint): یعنی هر نمونه موجودیت تنها به یکی از مجموعه موجودیت‌های سطح پایین‌تر تعلق دارد. مثلاً هر حساب بانکی می‌تواند پس‌انداز یا جاری باشد، اما نمی‌تواند به هر دو متعلق باشد.
- هم‌پوشان (Overlapping): یعنی یک نمونه موجودیت می‌تواند به بیش از یک مجموعه موجودیت سطح پایین تعلق داشته باشد. در مورد چهار تیم موجود در یک بانک فرض کنید که مدیران قادر باشند در چند تیم عضو شوند.

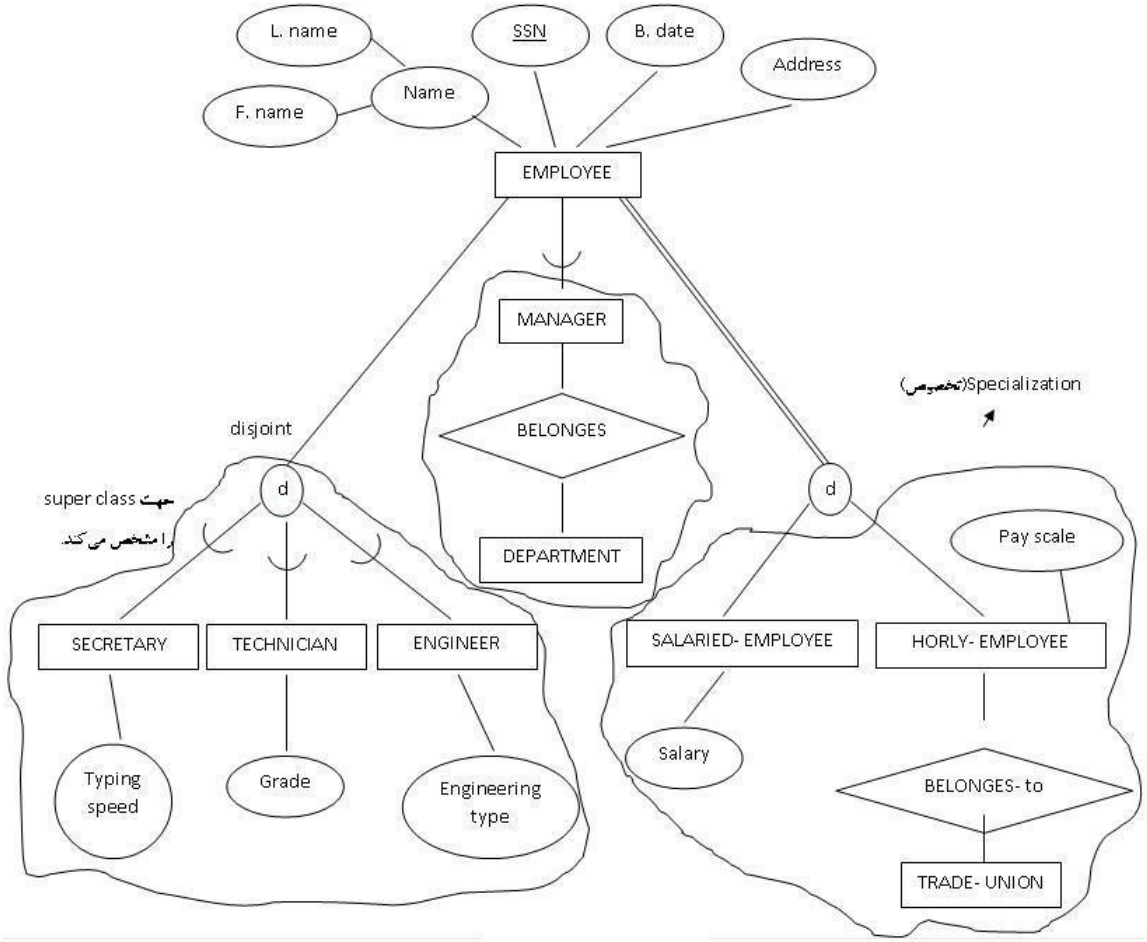
در حالت پیش فرض مجموعه موجودیت‌های سطح پایین هم‌پوشان فرض می‌شوند و خاصیت جدا از هم بودن باید صریحاً در مورد یک تخصیص اعلام گردد.

آخرین محدودیت مربوط به کامل بودن است (completeness constraint) که مشخص می‌کند آیا یک موجودیت از مجموعه موجودیت سطح بالاتر حداقل به یک مجموعه موجودیت سطح پایین‌تر تعلق دارد یا خیر. این محدودیت یکی از دو وضعیت زیر را می‌تواند نشان دهد:

- کامل (Total): که هر موجودیت از مجموعه موجودیت بالاتر حتماً عضو یک مجموعه موجودیت پایین‌تر هم هست.
- جزئی (Partial): که برخی از موجودیت‌های مجموعه موجودیت سطح بالا به هیچ مجموعه موجودیت سطح پایینی تعلق ندارند.

مثلاً تخصیص حساب بانکی یک تخصیص کامل است زیرا هر حساب یا جاری و یا پس‌انداز است. اما در مثال کارمندان و تیم‌های بانکی چون سه ماه پس از استخدام هر کارمند عضو یک تیم می‌شود، بنابراین کارمندانی وجود دارند که هنوز سه ماه از استخدام آن‌ها نگذشته و عضو هیچ تیمی نیستند.

نمونه‌ای از یک مدل EER در شکل زیر نمایش داده شده است. در این نمودار برای موجودیت EMPLOYEE سه تخصیص متفاوت ارائه شده است. حرف d در یک تخصیص نشان‌دهنده جدا از هم بودن آن می‌باشد. همچنین تخصیص‌های کامل به‌صورت دوخطی و تخصیص‌های جزئی به‌صورت تک خطی نشان داده می‌شوند.


L. name
SSN
B. date
Address
Name
F. name
EMPLOYEE
MANAGER
BELONGES
DEPARTMENT
Specialization(تخصیص)
disjoint
d
d
super class جهت
را مشخص می کند.
Pay scale
SECRETARY
TECHNICIAN
ENGINEER
SALARIED- EMPLOYEE
HORLY- EMPLOYEE
Typing speed
Grade
Engineering type
Salary
BELONGES- to
TRADE- UNION

۵ مدل داده‌ای رابطه‌ای

۵-۱ مفاهیم مدل رابطه‌ای

در این فصل مدل داده رابطه‌ای مورد بررسی قرار می‌گیرد. مدل رابطه‌ای در ابتدا توسط Ted Codd از مرکز تحقیقات IBM در سال ۱۹۷۰ در یک مقاله معرفی شد. به خاطر سادگی و مبنای ریاضی‌اش بلافاصله مورد توجه قرار گرفت. دلیل منسوخ نشدن آن هم این است که مبنای ریاضی دارد. این مدل از مفهوم رابطه در نظریه مجموعه‌های ریاضی، که چیزی شبیه به جدولی از مقادیر به نظر می‌رسد، استفاده می‌کند. اصول تئوری مدل رابطه‌ای توسط نظریه مجموعه‌ها و منطق ریاضی بیان می‌شود. امروزه مدل رابطه‌ای به عنوان مدل داده اصلی در کاربردهای پردازش داده‌های تجاری به کار گرفته شده است. این به خاطر سادگی آن می‌باشد که منجر به ساده شدن کار برنامه‌نویس در مقایسه با مدل داده‌های قبلی مثل مدل شبکه‌ای و سلسله مراتبی می‌شود. قبل از ابداع مدل داده‌ای، پایگاه داده بر اساس مدل‌های شبکه‌ای و سلسله مراتبی طراحی می‌شدند. این دو مدل قدیمی نسبت به مدل رابطه‌ای به پیاده‌سازی زیرساخت پایگاه داده نزدیک می‌باشند.

مدل رابطه‌ای پایگاه داده را به صورت کلکسیونی از رابطه‌ها که هر کدام نام واحدی دارند، نمایش می‌دهد. یک رابطه به عنوان جدولی از مقادیر در نظر گرفته می‌شود. سطری از جدول ارتباط بین مجموعه‌ای از مقادیر را نشان می‌دهد. به طور غیر رسمی می‌توان گفت که جدول مجموعه‌ای از نمونه موجودیت‌ها و سطر یک نمونه موجودیت است. در مدل رابطه‌ای هر سطر در جدول یک حقیقت را که عموماً به یک موجودیت یا رابطه در جهان واقعی شبیه است نمایش می‌دهد. تمام مقادیر در یک ستون از یک نوع داده هستند. مفهوم رابطه (Relation) از مفاهیم اساسی در مدل رابطه‌ای می‌باشد. مدل رابطه‌ای فقط از تعدادی رابطه تشکیل شده و مفهوم ارتباط (Relationship) در آن وجود ندارد.

در اصطلاحات مدل رابطه‌ای هر سطر یک چندتایی (Tuple)، عنوان ستون‌ها (header) هر ستون یک صفت و هر جدول یک رابطه نامیده می‌شود. برای هر صفت مجموعه‌ای از مقادیر مجاز وجود دارد. به این معنی که این صفت می‌تواند این مقادیر را بگیرد. این مجموعه مقادیر را دامنه آن صفت می‌نامند. هر صفت باید دامنه داشته باشد.

یک دامنه مجموعه‌ای از مقادیر اتمیک است. منظور از اتمیک، این است که هر مقدار در دامنه غیر قابل تجزیه است. به طور کلی جدولی با n صفت باید زیرمجموعه‌ای از $D_1 \times D_2 \times \ldots \times D_{n-1} \times D_n$ باشد که D_i دامنه صفت iام است.

ریاضی دانان رابطه را زیرمجموعه‌ای از ضرب دکارتی دامنه‌ها می‌دانند. این تعریف متناظر با تعریف جدول است، تنها تفاوت این است که برای صفات نام انتخاب کرده‌ایم. چون جداول اساساً رابطه‌اند، به جای واژه‌های جدول و سطر از واژه‌های رابطه و چندتایی استفاده می‌کنیم. ترتیب ظاهر شدن چندتایی‌ها در رابطه مهم نیست زیرا رابطه مجموعه‌ای از چندتایی‌هاست.

چندین صفت ممکن است دامنه یکسانی داشته باشند. به عنوان مثال فرض کنید رابطه Customer دارای سه صفت customer- name، customer- street و customer- city و رابطه Employee شامل صفت، employee- name است. ممکن است دامنه‌های صفات customer- name و employee- name یکسان بوده و برابر باشد با مجموعه‌ای از تمام اسامی افراد که در سطح فیزیکی برابر با مجموعه‌ای از تمام رشته‌های کاراکتری است. از سوی دیگر ویژگی‌هایی مانند

میزان موجودی و نام شعبه قطعاً دارای دامنه‌های کاملاً مجزایی هستند. مقدار تهی (NULL) عضوی از هر دامنه است که مشخص می‌کند مقدار واقعی معلوم نیست یا وجود ندارد. به عنوان مثال اگر یک Employee تلفن نداشته باشد به جای صفت شماره تلفن در چندتایی متناظر آن Employee مقدار NULL قرار می‌گیرد. در زمان دستیابی یا به هنگام‌سازی پایگاه داده، مقادیر NULL مشکلاتی را ایجاد خواهند کرد و در صورت امکان باید حذف شوند.

۲-۵ مدل داده‌ای و محدودیت‌های مدل رابطه‌ای

۱-۲-۵ شمای رابطه‌ای

وقتی در مورد پایگاه داده صحبت می‌کنیم باید بین شمای پایگاه داده که طراحی منطقی پایگاه داده است و نمونه پایگاه داده که تصویری از داده‌های پایگاه داده در لحظه خاصی از زمان است، تمایز قائل شویم.

مفهوم رابطه متناظر با فرضیه متغیرها در زبان برنامه‌سازی است. مفهوم شمای رابطه متناظر با فرضیه تعریف نوع در زبان برنامه‌سازی است. در زبان برنامه‌سازی برای تعریف نوع نام انتخاب می‌شود به راحتی می‌توان نامی برای شمای رابطه تعیین کرد.

به طور کلی شمای رابطه شامل لیستی از صفات و دامنه‌های متناظر با آن‌هاست. یک شمای رابطه‌ای R که با $R(A_1, A_2, \ldots, An)$ مشخص می‌شود از نام رابطه R و لیستی از صفات $A_1, A_2, \ldots, A_n$ تشکیل شده است. هر صفت مانند A_i نام یک نقش (Role) است که توسط بعضی دامنه‌های D در شمای رابطه‌ای R ایفا می‌شود. شمای رابطه‌ای برای توصیف یک رابطه استفاده می‌شود.

درجه هر رابطه تعداد صفات شمای رابطه‌ای آن‌هاست. یک رابطه r از یک شمای رابطه‌ای مانند $R(A_1,A_2, \ldots, A_n)$ نیز توسط r (R) نشان داده می‌شود که مجموعه‌ای از چندتایی‌هاست یعنی:

$r = \{t_1, t_2, \ldots, t_m\}$

مقدار صفت iام در تاپل t که به صفت A_i مربوط است، به صورت $t[A_i]$ نشان داده می‌شود.

یک رابطه $r(R)$، یک رابطه ریاضی از درجه n در دامنه‌های $dom(A_1), dom(A_2), \cdots, dom(A_n)$ است که زیرمجموعه ضرب کارتزین از دامنه‌هایی است که R را توصیف می‌کنند. بنابراین اگر تعداد عناصر دامنه D را با $|D|$ نشان دهیم و تمام دامنه‌ها را متناهی در نظر بگیریم، تعداد کل تاپل‌های رابطه R عبارتست از:

$$|dom(A_1)| \times |dom(A_2)| \times \cdots \times |dom(A_n)|$$

۲-۲-۵ ویژگی‌های مدل رابطه‌ای

به‌طور کلی ویژگی‌های مدل رابطه‌ای عبارتست از:

- عدم اهمیت ترتیب در چند تایی‌های یک رابطه
- ترتیب صفات در شمای رابطه‌ای اهمیت ندارد (با عوض شدن ترتیب صفات یک رابطه جدید به وجود نمی‌آید.)
- یک شمای پایگاه داده نمی‌تواند دو تا رابطه هم نام داشته باشد.

- هر مقدار یک چندتایی اتمیک است.

علائم نمایشی زیر برای توصیف مدل رابطه‌ای استفاده می‌شود.

$R(A_1, A_2, \cdots, A_n)$

$r(A) = \{t_1, t_2, \cdots, t_m\}$

$t = \langle v_1, v_2, \cdots, v_n \rangle$

$v_i \in dom(A_i)$

$v_i = t[A_i]$

۵-۲-۳ محدودیت دامنه

یکی از محدودیت‌های مهم در مدل رابطه‌ای، محدودیت دامنه است. طبق این محدودیت هر صفت در هر رابطه باید یک دامنه مشخص داشته باشد. این در حالی است که در مدل ER این محدودیت وجود نداشت. بنابراین باید توجه داشت که در تعریف مدل رابطه‌ای باید دامنه هر صفت مشخص شود. برای این منظور هر سیستم پایگاه داده رابطه‌ای تعدادی دامنه پیش‌فرض دارد و همچنین امکان تعریف دامنه‌های جدید را برای طراح پایگاه داده فراهم می‌کند.

۵-۲-۴ محدودیت کلید

برای تمایز چندتایی‌ها در داخل یک رابطه باید راهی داشته باشیم که معمولاً بر اساس صفات آن‌ها بیان می‌شوند. یعنی مقادیر صفت چندتایی باید طوری باشند که بتوانند انحصاراً آن چندتایی را مشخص کنند. برای این منظور مفهوم کلید در مدل رابطه‌ای وجود دارد. در مدل رابطه‌ای چندین نوع کلید وجود دارد.

- سوپرکلید (Super Key): مجموعه‌ای از یک یا چند صفت است که چندتایی‌های یک رابطه را از یکدیگر متمایز می‌کند. به طور مثال: صفت customer_id مربوط به رابطه customer برای تمایز یک چندتایی customer از دیگری کافی است. بنابراین customer_id یک سوپرکلید است. ترکیب customer_name و customer_id هم برای رابطه customer یک سوپرکلید است. صفت customer_name مربوط به رابطه customer سوپرکلید نیست زیرا ممکن است اسامی چند نفر یکسان باشد. هر جدول یا رابطه حتماً یک سوپرکلید دارد و آن هم مجموعه همه صفات آن است.
- کلید حداقل یا کلید کاندید (Candidate key): سوپرکلیدی که هیچ زیرمجموعه محض آن سوپرکلید نباشد، یک کلید کاندید است. ممکن است چندین مجموعه مجزا از صفات به عنوان کلید کاندید عمل کنند. فرض کنید ترکیبی از customer_name و customer_street برای تمایز بین اعضای رابطه customer کافی باشد. در این صورت هم {customer_id} و هم {customer_name,customer_street} کلید کاندید هستند. اگرچه صفات customer_id و customer_name با هم می‌توانند چندتایی‌های Customer را متمایز کنند، اما ترکیب آن‌ها کلید کاندید نیست، زیرا صفت customer_id به تنهایی یک کلید کاندید است. می‌توانیم در یک رابطه بیش از یک

candidate key داشــته باشــیم. کلید کاندید را کلید حداقل نیز می‌گوییم، به این دلیل که هیچ صــفت اضــافی ندارد. به‌عبارت دیگر با حذف هر صفت، دیگر کلید مربوطه سوپرکلید نیست.

- کلید اصـلی (Primary Key): یکی از کلیدهای کاندید توسـط طراح پایگاه داده به‌عنوان کلید اصـلی انتخاب می‌شـود. طراح پایگاه داده باید برای هر رابطه یک کلید اصلی انتخاب نماید.

کلیدها در مدل رابطه‌ای خاصـیتی از کل رابطه می‌باشند و مربوط به چندتایی‌های خاص از یک رابطه نمی‌باشند. بنابراین هیچ دو چندتایی در یک رابطه هم‌زمان نمی‌توانند مقدار یکسانی در صفات کلید داشته باشند.

کلید اصـلی باید طوری انتخاب شـود که مقادیر صـفات آن هرگز تغییر نکنند یا به ندرت تغییر کنند. مثلاً فیلد آدرس افراد نباید به عنوان کلید اصلی در نظر گرفته شود زیرا احتمالاً تغییر خواهد کرد.

محدودیت کلید در مدل رابطه‌ای بیان می‌کند که هر رابطه در یک مدل رابطه‌ای باید کلید اصلی داشته باشد.

۵-۲-۵ محدودیت جامعیتی

محدودیت‌های جامعیتی بخشــی از محدودیت‌های مدل رابطه‌ای می‌باشــند که جهت برقراری ارتباط مناســب بین رابطه‌ها تعریف می‌شوند. این محدودیت‌ها عبارتند از:

- جامعیت موجودیتی (Entity Integrity)
- جامعیت ارجاعی (Referential Integrity)

محدودیت جامعیت موجودیتی بیان می‌دارد که مقدار کلید اصــلی نمی‌تواند Null باشــد. این محدودیت به آن دلیل اســت که مقدار کلید اصلی برای مشخص کردن چندتایی‌های منحصر به فرد در یک رابطه استفاده می‌شود.

محدودیت جامعیت ارجاعی بین دو رابطه تعیین می‌شــود و برای نگهداری ســازگاری بین چندتایی‌ها ملحق شــده از دو رابطه اســتفاده می‌شــود. محدودیت جامعیت ارجاعی بیان می‌دارد که یک چندتایی در یک رابطه که به یک رابطه دیگر ارجاع پیدا می‌کنـد، باید به یک چندتایی موجود در رابطه مورد ارجاع رجوع کند. برای توصــیف این محدودیت به طور دقیق‌تر ابتدا مفهوم کلید خارجی (Foreign key) را بیان می‌کنیم.

شـرایطی که برای کلید خارجی در زیر آمده اسـت یک محدودیت جامعیت ارجاعی را بین دو شمای رابطه‌ای R_1 , R_2 مشخص می‌کند. کلید خارجی لازم نیست هم‌نام با کلید اصلی باشد.

مجموعه‌ای از صفات در شمای رابطه‌ای R_1 کلید خارجی مورد ارجاع به رابطه R_2 است اگر:

شـرط اول: صـفات موجود در کاید خارجی (FK) دامنه مشابهی با صفات کلید کلید اصلی (PK) از شمای رابطه دیگری نظیر R_2 داشته باشد.

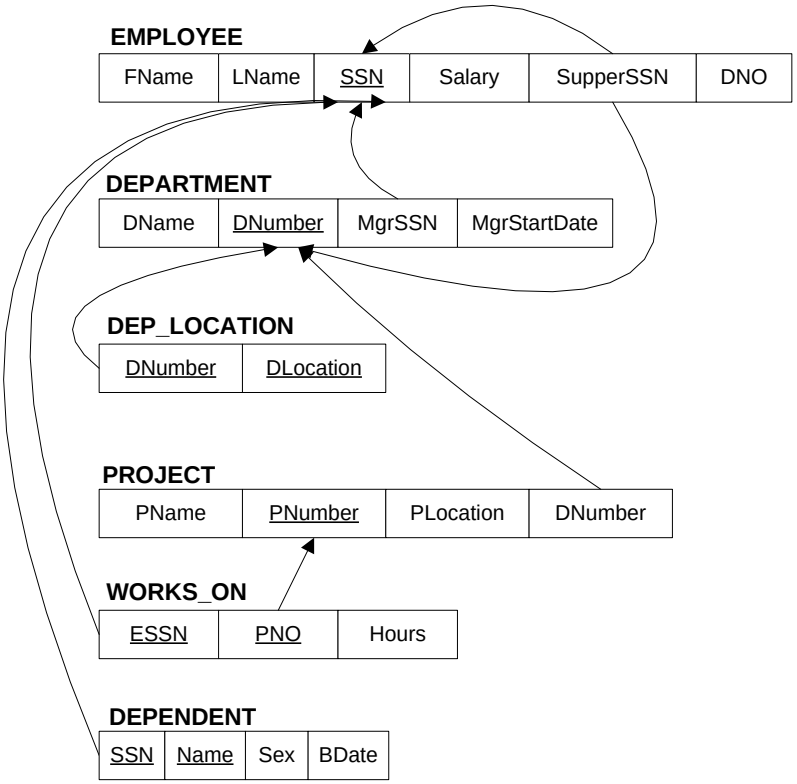
شـرط دوم: مقداری از FK در یک چندتایی t_1 از R_1 یا برابر مقداری از PK برای یک چندتایی t_2 در R_2 باشد و یا برابر NULL باشد.

$T_{1[FK]}$ = either $t_{2[PK]}$ or NULL

ویژگی‌های زیر برای کلید خارجی در مدل رابطه‌ای برقرار است.

- کلید خارجی در یک رابطه می‌تواند به خود رابطه ارجاع دهد.
- یک جدول می‌تواند بیش از یک کلید خارجی داشته باشد و یا اصلاً نداشته باشد.

مدل رابطه‌ای برای مثال Company در شکل زیر آورده شده است. همان‌طور که مشخص است این مدل شامل تعدادی جدول و صفات مربوطه است. فلش‌ها نمایانگر ارجاع یک کلید خارجی به کلید اصلی است. در هر رابطه صفاتی که زیرخط دارند، نمایانگر کلید اصلی آن رابطه می‌باشند. باید توجه داشت که برخی از روابط کلید ترکیبی دارند.



۳-۵ عملیات به روزرسانی

در مدل رابطه‌ای سه عملگر به‌روز رسانی زیر وجود دارد.

- درج کردن (Insert): برای درج کردن یک چندتایی جدید یا چندتایی‌های جدید در یک رابطه مورد استفاده قرار می‌گیرد. این عملگر می‌تواند هر ۴ نوع محدودیت را تحت تأثیر قرار می‌دهد.
- حذف کردن (Delete): برای حذف کردن یک چندتایی یا چندتایی‌های در یک رابطه مورد استفاده قرار می‌گیرد. این عملگر تنها جامعیت ارجاعی را تحت تأثیر قرار می‌دهد، اگر چندتاییی که قرار است حذف شود ارجاعی از کلید خارجی چندتایی‌های دیگر داشته باشد.
- تغییر دادن (Modify): این عملگر برای تغییر یا تعویض مقادیر بعضی از صفات چندتایی‌های موجود استفاده می‌شود. این عملگر می‌تواند هر ۴ نوع محدودیت را تحت تأثیر قرار می‌دهد.

۶ جبر رابطه‌ای

جبر رابطه‌ای قوی‌ترین مبنای تئوریک مدل رابطه‌ای است. جبر رابطه‌ای یک زبان پرس‌وجوی روالی است. این زبان شامل تعدادی عملگر است که این عملگرها یک یا دو رابطه را به عنوان ورودی دریافت می‌کنند و رابطه جدیدی را به عنوان نتیجه عمل بر روی رابطه (های) ورودی باز می‌گردانند.

۶-۱ عملگرهای انتخاب و تصویر

۶-۱-۱ عملگر انتخاب

این عملگر چندتایی‌هایی را که شرط خاصی را برقرار نمایند انتخاب می‌نماید. برای نمایش این عملگر از نماد سیگمای کوچک یونانی (σ) استفاده می‌شود. شرط به عنوان اندیس σ نوشته می‌شود و رابطه ورودی که شرط بر روی آن اعمال می‌شود، در داخل پرانتزی در جلوی σ قرار می‌گیرد. این عملگر فقط زیرمجموعه‌ای از چندتایی‌ها و همه صفات یک جدول را که شرط را برقرار می‌کنند، انتخاب می‌کند. پس در پرانتز نمی‌توان نام چند جدول یا رابطه را با علامت، قرار داد.

خروجی عملگر Select یک رابطه است با تمامی صفات رابطه پایه. (هم درجه با جدول پایه می‌باشد) با توجه به تعریف عملگر انتخاب، این عملگر جابه‌جاپذیر است. بنابراین روابط زیر برقرار است.

$$\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}{}^{(R)}) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition1}>}{}^{(R)}) = \sigma_{<\text{condition1} \wedge \text{condition2}>}{}^{(R)}$$

$$\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\dots(\sigma_{<\text{condn}>}{}^{(R))\dots}) = \sigma_{<\text{cond1}>}\ \text{AND}\ _{<\text{cond2}>}\ \text{AND}\ \dots\ \text{AND}\ _{<\text{condn}>}{}^{(R)}$$

نمونه‌هایی از چند دستور select بر روی جداول مختلف در زیر آمده است.

- مشخصات کارمندانی که در دپارتمان ۴ کار می‌کنند.

$$\sigma_{\text{DNO} = 4}(\text{EMPLOYEE})$$

- مشخصات کارمندانی که حقوق بیش‌تر از ۳۰۰۰۰$ دریافت می‌کنند.

$$\sigma_{\text{salary} > \$30000}(\text{EMPLOYEE})$$

با در نظر گرفتن رابطه Loan و Customer در سیستم بانکی هم می‌توانیم به چندین پرس‌وجو که در زیر آمده با استفاده از عملگر انتخاب پاسخ دهیم.

Costomer

Customer_name	Social_security	Customer_street	Customer_city
Jones	312- 22- 7412	Main	Horisson
Hayes	719- 21- 3224	Main	Horrison

Loan

Branch- name	Loan- number	Amount
DownTown	L- 15	1500
DownTown	L- 27	1500
Perryridge	L- 21	1700
Mianus	L- 11	2000
Redwood	L- 13	2000

- مشخصات مشتریانی که ساکن خیابان اصلی (Main) هستند.

$\sigma_{\text{customer- street = “main”}}$ (Customer)

- مشخصات تمامی وام‌هایی که میزان آن‌ها بیش از ۱۰۰۰ دلار بوده است.

$\sigma_{\text{amount > \$1000}}$ (Loan)

- مشخصات وام‌هایی که در شعبه DownTown پرداخت شده و میزان آن‌ها بیش از $1000 بوده است.

$\sigma_{\text{amount > \$1000 AND branch\_name = “DownTown”}}$ (Loan)

۶-۱-۲ عملگر تصویر

عملگر تصویر (Project) یک عملگر یکتایی است (بر روی یک رابطه عمل می‌کند) که یک رابطه را به عنوان ورودی می‌گیرد و با کنار گذاشتن صفات مشخص از آن رابطه جدیدی را تولید می‌نماید. ممکن است سطرهایی در جدول وجود داشته باشند که تنها در ویژگی‌های حذف شده با هم تفاوت داشته باشند. بنابراین پس از حذف صفات مزکور این سطرها یکسان می‌شوند و با توجه به این که هر رابطه یک مجموعه است، در چنین مواردی سطرهای تکراری حذف می‌شوند. بنابراین در عمل تصویر ابتدا تعدادی از صفات کنار گذاشته می‌شوند و سپس سطرهای تکراری در صورت وجود حذف خواهند شد.

عملگر تصویر با استفاده از حرف یونانی (Π) نشان داده می‌شود. در پانوشت Π نام صفات نوشته خواهد شد که می‌خواهیم در رابطه حاصل وجود داشته باشند. (یعنی تصویرسازی روی آن‌ها صورت می‌گیرد) و پس از آن رابطه ورودی را در یک پرانتز جلوی Π قرار می‌دهیم. مثلاً پرس‌وجو که مشخص کننده شماره و مقدار وام‌های پرداخت شده توسط بانک باشد به صورت زیر است.

$\Pi_{\text{Lean\_number, amount}}$ (LOAN)

- لیستی از نام و نام خانوادگی و حقوق کارمندان Company

$\Pi_{\text{Fname, Lname, salary}}$ (EMPLOYEE)

$\Pi_{<\text{List1}>}(\Pi_{<\text{List2}>}{}^{(R)}) = \Pi_{<\text{List1}>}{}^{(R)}$

۶-۱-۳ دنباله‌ای از عملگرها و عملگر تغییر نام

این‌که نتیجه اعمال یک عملگر رابطه‌ای بر روی یک رابطه، خود یک رابطه اســت واقعیت مهمی اســت که به ما این اجازه را می‌دهد که عملگرهای رابطه‌ای را با یکدیگر ترکیب نماییم.

- نام مشتریانی را بیابید که در شهر Tehran زندگی می‌کنند.

$\Pi_{customer\_name}(\sigma_{customer\_city = \text{“Tehran”}}(customer))$

- نام و نام خانوادگی و حقوق همه کارمندانی که در دپارتمان شماره ۵ کار می‌کنند.

$\Pi_{Fname, Lname, Salary}(\sigma_{DNO = 5}(EMPLOYEE))$

یا

$DEP5\_EMPS \leftarrow \sigma_{DNO = 5}(EMPLOYEE)$

$RESULT \leftarrow \Pi_{Fname, Lname, Salary}(DEP5\_EMPS)$

در پایگاه داده جداول حاصــل از انجام عملیات و اعمال عملگرهای جبری بر روی چند جدول فاقد نام هســتند. با اســتفاده از عملگر تغییر نــام می‌توانیم بــه این جــداول نــام تخصــیص دهیم. برای نمــایش عملگر تغییر نام از حرف کوچک یونانی رو (ρ) اســتفاده می‌کنیم. اگر E یک عبارت جبر رابطه‌ای باشــد ρxE نتیجه عبارت E را (که یک رابطه اســت) با نام x برمی‌گرداند. می‌توانیم عملگر تغییر نــام را بر روی یــک رابطــه مــانند σ نیز اعمــال کنیم چرا کــه می‌توانیم هر رابطــه مانند r را به عنوان یک عبارت جبر رابطه‌ای در نظر بگیریم. اگر حاصــل عبارت جبر رابطه‌ای E یک رابطه با n صــفت باشــد می‌توانیم ρ را به صــورت زیر به کار گیریم:

$\rho_{x(A1, A2\ldots An)}(E)$

نتیجه عملگر ρ بر روی پرس‌وجوی قبل به صورت زیر است:

$TEMP \leftarrow \sigma_{DNO = 5}(EMPLOYEE)$

$R(FIRSTNAME, LASTNAME, SALARY)\ \Pi_{Fname, Lname, Salary} \leftarrow (TEMP)$

نتیجه عملگر ρ بر روی رابطه‌ای مانند R از درجه n به‌صورت زیر است.

$\rho_{s(B1, B2, \ldots, Bn)}(R)$ or $\rho_{s}(R)$ or $\rho_{(B1, B2, \ldots, Bn)}(R)$

۶-۲ عملگرهای نظریه مجموعه‌ها

۶-۲-۱ اجتماع، اشتراک و تفاضل

پرس‌وجوی زیر را در نظر بگیرید:

نام مشتریانی که یا در بانک حساب داشته باشند و یا وام دریافت کرده باشند و یا هر دو.

با توجه به این که لزومی ندارد یک مشــتری بانک حتماً در آن بانک حســاب داشــته و یا وامی دریافت کرده باشــد، لذا رابطه Customer حاوی اطلاعات کافی در این زمینه نیست. برای آن‌که نام مشتریانی را بیابیم که از بانک وام گرفته‌اند، به سادگی

از (BORROWER) $\Pi_{customer\_name}$ استفاده می‌کنیم و $\Pi_{customer\_name}$(DEPOSITOR) نیز مشخص کننده نام مشتریان صاحب حساب در بانک است.

اجتماع این دو مجموعه پاسخ پرس‌وجوی ما خواهد بود.

$\Pi_{customer\_name}$ (BORROWER) U $\Pi_{customer\_name}$ (DEPOSITOR)

در حالت کلی باید توجه کنیم که اجتماع باید بر روی مجموعه‌هایی صورت گیرد که سازگار (Compatible) باشند. به عنوان مثال نمی‌توانستیم میان دو رابطه BORROWER و DEPOSITOR اجتماع بگیریم زیرا تعداد صفات آن‌ها یکسان نیست.

بین دو مجموعه نام مشتری و میزان موجودی حساب هم نمی‌توانستیم اجتماع بگیریم، زیرا این مجموعه‌ها از دو جنس مختلف هستند (دامنه‌های جدا از هم دارند.) به طور کلی برای آن‌که عمل rUs مجاز باشد لازم است دو شرط زیر برقرار باشد:

۱) رابطه r و s هم‌درجه باشند. (تعداد صفات آن‌ها برابر باشد.)
۲) به ازای هر i باید دامنه ویژگی iام و ویژگی sام یکسان باشد.

عملگر تفاضل مجموعه‌ای (Set Difference) یا (MINUS) را با - نشان می‌دهیم. این عملگر به ما اجازه می‌دهد که اعضایی از یک رابطه را بیابیم که در رابطه دیگر وجود نداشته باشد. عبارت r-s رابطه‌ای است که شامل همه چندتایی‌های موجود در r است که در s موجود نباشد.

به‌عنوان نمونه نام مشتریانی از بانک که در بانک حساب دارند اما وامی دریافت نکرده‌اند، را درخواست کرده‌ایم. پرس‌وجوی آن به‌صورت زیر است.

$\Pi_{customer\_name}$ (DEPOSITOR) – $\Pi_{customer\_name}$ (BORROWER)

در این‌جا هم r و s باید سازگار باشند یعنی دو شرط گفته شده در مورد اجتماع باید در مورد تفاضل هم برقرار باشد.

به‌عنوان نمونه دیگر نام مشتریانی را مشخص کنید که هم در بانک دارای حساب هستند و هم از بانک وام دریافت کرده‌اند. برای پاسخ به این پرس‌وجو باید از عملگر اشتراک استفاده کنیم.

$\Pi_{customer\_name}$ (BORROWER) – $\Pi_{customer\_name}$ (DEPOSITER)

یک عبارت جبر رابطه‌ای شامل اشتراک را می‌توان با استفاده از دو عملگر تفاضل مجموعه‌ای هم نوشت در حالت کلی داریم.

$$r \cap s = r - (r - s)$$

در مورد این عملگر تنها می‌توان گفت نوشتن $r \cap s$ آسان‌تر از نوشت $r - (r - s)$ است. همچنین روابط زیر برقرار است.

$$(R \cup S) \cup T = R \cup (S \cup T)$$
$$(R \cap S) \cap T = R \cap (S \cap T)$$
$$R - S \neq S - R$$

۶-۲-۲ ضرب دکارتی

عملگر ضـرب دکارتی که با علامت (×) نشـان داده می‌شـود، به ما اجازه می‌دهد که اطلاعات دو رابطه را با هم ترکیب نماییم. این حاصـل ضـرب ($r_1 \times r_2$) شـامل همه ستون‌های دو رابطه r_1 و r_2 است، سطرهای آن حاصل ترکیب همه سطرهای r_1 و r_2 با یکدیگر اسـت. از آن‌جا که ممکن است دو رابطه r_1 و r_2 دارای صفات هم‌نامی باشند، برای آن‌که میان ستون‌های هم‌نام در رابطه حاصل تمایز قائل شویم با استفاده از نام رابطه و یک نقطه مشخص می‌کنیم که هر صفت از کدام رابطه آمده است.

به عنوان مثال اگر رابطه r_1 دارای صفات ($a_1, a_2, \ldots, a_n, c_1, \ldots, c_k$) و رابطه r_2 دارای صفات ($b_1, b_2, \ldots, b_m, c_1, \ldots,$) باشد رابطه $r_1 \times r_2$ دارای صفات زیر خواهد بود.

$$(a_1, a_2, \ldots, a_n, r_1.c_1, r_1.c_2 \ldots r_1.c_k, b_1, b_2, \ldots b_m, r_2.c_1, r_2.c_2, \ldots, r_2 c_k)$$

- لیستی از نام و نام خانوادگی کارمندان و وابسته‌های آن‌ها ارائه کنید.

$$FAMALE\_EMPS \leftarrow \sigma_{SEX= 'F'} (EMPLOYEE)$$

$$EMPNAMES \leftarrow \Pi_{Fname, Lname, SSN} (FAMALE\_EMPS)$$

$$ACTUAL\_DEPENDENTS \leftarrow EMPNAMES \times DEPENDENT$$

$$RESULT \leftarrow Fname, Lname, DependentName (ACTUAL\_DEPENDENTS)$$

- بیش‌ترین موجودی حساب‌های بانک را بیابید.

برای این کار ابتدا تمام موجودی‌هایی را می‌یابیم که حداقل یک حسـاب با موجودی بیش‌تر از آن‌ها وجود داشته باشد و سپس حـاصـــل را از مجموعه کل موجودی‌ها کم می‌کنیم. ابتدا جدول Account را در خودش ضـــرب می‌کنیم تا بتوانیم موجودی حسـاب‌ها را با هم مقایسـه کنیم. برای ایجاد تمایز دو جدولی که در هم ضـرب می‌شـوند یکی را تغییر نام می‌دهیم و سـپس حاصل را حساب می‌کنیم.

$$Account \times \rho_j (Account)$$

از این حاصـل ضـرب آن‌هایی را انتخاب می‌کنیم که حداقل یک حسـاب با میزان موجودی بیش از آن‌ها موجود باشد و سپس حاصل را بر روی ویژگی موجودی تصویر می‌کنیم.

$$\Pi_{account.amount} (\sigma_{account.amount < d.amount} (Account \times \rho_j (Account)))$$

۶-۳ عملگرهای رابطه‌ای دودویی

۶-۳-۱ الحاق

عملگرهای دیگری نظیر عملگر الحاق (Join) به قدرت جبر رابطه‌ای اضـافه نمی‌کنند، بلکه سـبب سـاده سـازی پرس‌وجوها می‌شوند. برای نمایش این عملگر از علامت ∞ استفاده می‌کنیم.

فرم کلی عملگر JOIN بر روی دو رابطه R و S به‌صورت زیر است.

$(B_1, B_2, \ldots, B_m)\ (A_1, A_2, \ldots, A_n)$

$R \infty_{\text{join condition}} S$

به‌طور کلی عملگر الحاق همان عملگر ضرب است که توسط یک شرط مقید شده است. برای اجرای این عملگر ابتدا عمل ضرب دکارتی بین دو رابطه اجرا شده و سپس چندتایی‌هایی انتخاب می‌شود که شرط برای آن‌ها درست باشد. بنابراین رابطه زیر برقرار است.

$R \infty_{<condition>} S \equiv \sigma_{<condition>} (R \times S)$

- نام و نام خانوادگی مدیران و اسم دپارتمانی که مدیریت می‌کنند، را بدهید.

$$\Pi_{Dname,\ Fname,\ Lname} (DEPARTMENT \infty_{MGRSSN=\ SSN} EMPLOYEE)$$

عملگر الحاق در سه نوع زیر قابل استفاده است.

۱) Θ_JOIN: وقتی شرط الحاق یکی از عملیات مقایسه‌ای باشد. $(<, \leq, >, \geq)$

۲) EQUIJOIN: وقتی در شرط الحاق تنها از عمل تساوی استفاده شود. این نوع الحاق پرکاربردترین است و در خروجی آن هر دو صفت مبنای تساوی شرکت می‌کنند.

۳) NATURALJOIN: شبیه EQUIJOIN است ولی تنها یک صفت مبنای تساوی را در خروجی قرار می‌دهد. در این الحاق نام صفات مبنای تساوی باید یکسان باشد. علامت این الحاق * است. وقتی از NATURALJOIN استفاده می‌کنیم باید دو رابطه شامل صفات هم‌نام باشند. بنابراین ممکن است که نیاز به تغییر نام صفات داشته باشیم.

۶-۳-۲ تقسیم

عملگر تقسیم با (÷) نشان داده می‌شود. غالباً در مورد درخواست‌هایی کاربرد دارد که شامل عبارات "به ازای هر" باشد. برای نمونه در نظر بگیرید که مشتریانی را که در همه شعبه‌های واقع در Brooklyn حساب دارند، را نیاز دارید. بنابراین دستورات زیر اجرا می‌شوند.

$r_1 = \Pi_{branch\_name} (\sigma_{branch\_city = \text{“Brooklyn”}} (BRANCH))$

$r_2 = \Pi_{customer\_name,\ branch\_name} (DEPOSITOR \infty ACCOUNT)$

$RESULT \leftarrow r_2 \div r_1$

فرض کنید s(S) و r(R) دو رابطه باشند، و R زیرمجموعه S باشد به این مفهوم که همه صفات موجود در شمای s در r نیز موجود باشد. رابطه r÷s رابطه‌ای است بر روی شمای y=R-S (شمایی شامل همه صفاتی از R که در S نیستند) که شرایط زیر برقرار باشد.

$T(y) = r(R) \div s(S)$

$T_1 \leftarrow \Pi_y (r)$

$T_2 \leftarrow \Pi_y ((S \times T_1) - r)$

$T \leftarrow T_1 - T_2$

نام و نام خانوادگی تمام کارمندانی که روی همه پروژه‌های آقای 'John Smith' کار می‌کنند را بدهید.

$SMITH \leftarrow \sigma_{Fname= \text{'John'} \wedge Lname= \text{'Smith'}} (EMPLOYEE)$

$SMITH\_PNOS \leftarrow \Pi_{PNO} (WORKS\_ON \infty SMITH)$

ESSN= SSN

$SSN\_PNO \leftarrow \Pi_{ESSN\_PNO} (WORKS\_ON)$

$SSNS (SSN) \leftarrow SSN\_PNOS \div SMITH\_PNOS$

$RESULT \leftarrow \Pi_{Fname, Lname} (SSNS * EMPLOYEE)$

۴-۶ عملگرهای رابطه‌ای اضافی

خلاصه‌ای از عملگرهای ارائه شده در جبر رابطه‌ای در شکل زیر آورده شده است. این عملگرها به‌عنوان عملگرهای مبنایی در جبر رابطه‌ای مطرح می‌باشند. علاوه‌بر این عملگرها، تعدادی عملگر دیگر برای کاربردهای خاص در جبر رابطه‌ای ارائه شده است که در این بخش مورد بررسی قرار خواهد گرفت.

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\langle \text{SELECTION CONDITION} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\langle \text{ATTRIBUTE LIST} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle \text{JOIN CONDITION} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{JOIN CONDITION} \rangle} R_2$, OR $R_1 \bowtie_{(\langle \text{JOIN ATTRIBUTES 1} \rangle), (\langle \text{JOIN ATTRIBUTES 2} \rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\langle \text{JOIN CONDITION} \rangle} R_2$, OR $R_1 *_{(\langle \text{JOIN ATTRIBUTES 1} \rangle), (\langle \text{JOIN ATTRIBUTES 2} \rangle)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

۱-۴-۶ توابع تجمعی و گروه‌بندی

توابع تجمعی توابعی هستند که مجموعه‌ای از مقادیر را به عنوان ورودی می‌گیرد و یک مقدار را به عنوان جواب بازمی‌گرداند. برای نمونه تابع Average میانگین مقادیر را باز می‌گرداند. حاصل تابع count تعداد مقادیر متمایز در یک جدول است. توابع

max و min نیز به ترتیب برای بازگرداندن بیش‌ترین و کم‌ترین مقدار از مقادیر ورودی به کار می‌روند. تابع sum مجموع مقادیر را برمی‌گرداند. توابع تجمعی با علامت f نشان داده‌می‌شوند.

فرض کنید جدول pt_works حاوی داده‌های کارمندان پاره وقت بانک باشد.

- مجموع حقوقی که به کارمندان پاره وقت پرداخت می‌شود، را ارائه کنید.

EMPLOYEE name	Branch_name	Salary
Johnson	Down town	1500
Lorena	Down town	1300
Peterson	Down town	2500
Sato	Austin	1600
Rao	Austin	1500
Gopal	Prryridge	5300
Adoma	Prryridge	1500
Brown	prryridge	1300

$f_{\text{sum salary}}$ (pt_works)

حاصل این پرس‌وجو رابطه‌ای است که دارای یک ستون می‌باشد و آن هم تنها یک چندتایی دارد. این جدول به‌صورت زیر است.

Sum salary
16500

مواردی وجود دارد که لازم است به جای اعمال توابع تجمعی بر روی یک مجموعه ورودی آن‌ها را بر روی چندین گروه اعمال کنیم که هر گروه شامل مجموعه‌ای از چندتایی است. برای انجام منظور از گروه‌بندی استفاده می‌شوند. ایده گروه‌بندی ابتدا چندتایی‌های موجود در یک جدول را بر مبنای مقدار یک صفت گروه‌بندی می‌کند. به‌عبارت دیگر چندتایی‌ها با مقدار مساوی برای صفت گروه، در یک گروه قرار می‌گیرند. بعد از گروه‌بندی، توابع تجمعی به‌طور جداگانه بر روی هر گروه اعمال شده و نتیجه آن در جدول خروجی ارائه می‌شود. بنابراین نتیجه اعمال توابع تجمعی به همراه گروه‌بندی می‌تواند بیش از یک چندتایی در جدول خروجی ایجاد نماید. مثال‌های زیر نحوه گروه‌بندی و کاربرد آن را مشخص می‌کند.

- مجموع حقوق کارمندان پاره وقت هر شعبه را به تفکیک به دست آورید.

(Branch_name) $f_{\text{sum salary}}$ (pt_works)

- نام و نام خانوادگی همه مدیران دپارتمان‌ها که بیش از ۳ فرزند دارند، را بدهید.

$R_1 \Pi_{Fname,\ Lname} \leftarrow (EMPLOYEE \infty_{SSN=\ MERSSN} DEPARTMENT)$

$R_2 \leftarrow (R_1 \infty_{SSN=\ ESSN} DEPENDENT)$

$R_3 \leftarrow (SSN)\ f_{count\ dependent\ name}\ (R_2)$

$\Pi_{Fname,\ Lname}\ (\sigma_{count\ dependetname>3}\ (R_3) \infty EMPLOYEE)$

۶-۴-۲ عملگر بازگشتی

برای اجرای برخی از پرس‌وجوها ممکن است که نیاز به اجرای بازگشتی برخی از عملیات روی یک جدول احساس شود. جبر رابطه‌ای امکان اجرای عملیات به‌صورت بازگشتی را فراهم نمی‌کند. برای نمونه در نظر بگیرید که می‌خواهید نام تمام زیردرست‌های آقای "James Borg" را بدست آورید. بدیهی است که زیردست‌ها می‌توانند تا چندین سطح تکرار شوند. برای نوشتن عملیات جبر رابطه‌ای باید بدانیم تا چند سطح ادامه دهیم. برای نمونه در زیر تا دو سطح این درخواست اجرا شده است.

$BORG\_SSN \leftarrow \Pi_{SSN}\ (\sigma_{Fname=" James" \wedge Lname=" Borg"}\ (EMPLOYEE))$

$SUPERVISION\ (SSN_1, SSN_2) \leftarrow \Pi_{ssn,\ superSSN}\ (EMPLOYEE)$

$RESULT_1\ (SSN) \leftarrow \Pi_{SSN1}\ (supervision \infty Borg\_SSN)$

$RESULT_2\ (SSN) \leftarrow \Pi_{SSN1}\ (supervision \infty_{SSN_2=\ SSN} RESULT_1)$

$RESULT \leftarrow RESULT_1 \cup RESULT_2$

۶-۴-۳ الحاق بیرونی

الحاق بیرونی تعمیمی از عملگر الحاق است که مقادیر اطلاعات از دست رفته حاصل از این عمل را مورد توجه قرار می‌دهد. این عملگر زمانی که ما بخواهیم تمام چندتایی‌های موجود در R یا همه چندتایی‌های موجود در S یا همه چندتایی‌های دو رابطه را نگه داریم مورد استفاده قرار می‌گیرد.

الحاق بیرونی به سه نوع چپ، راست و کامل تقسیم می‌شود. الحاق بیرونی چپ همه چندتایی‌های رابطه سمت چپ را نگه می‌دارد و صفاتی از S که ارتباطی با چندتایی‌های R نداشته باشند، با مقدار NULL در جدول حاصل قرار می‌گیرند. روش نمایش این الحاق به‌صورت زیر است.

The left outer join $\overset{\sqsupset\bowtie}{}$: (R $\overset{\sqsupset\bowtie}{}$ S)

الحاق بیرونی چپ همه چندتایی‌های رابطه سمت راست را نگه می‌دارد و صفاتی از R که ارتباطی با چندتایی‌های S نداشته باشند، با مقدار NULL در جدول حاصل قرار می‌گیرند. روش نمایش این الحاق به‌صورت زیر است.

The right outer join $\overset{\bowtie\sqsubset}{}$: (R $\overset{\bowtie\sqsubset}{}$ S)

در الحاق بیرونی کامل اجتماع الحاق چپ و راست انجام می‌شود. نمونه زیر را در نظر بگیرید.

- نام همه کارمندان و نام دپارتمان‌هایی که مدیریت می‌کنند.

The left outer join ⟕: (R ⟕ S)

SSN= MERSSN

RESULT ← $\Pi_{Fname,\ Minit,\ Lname,\ Dname}$ (TEMP)

RESULT	Fname	Minit	Lname	Dname
	John	B	Smith	Null
	Feranklin	T	Wong	Research
	Alicia	J	Zaiya	Null
	Jenifer	S	Wallace	Administration
	Ramesh	K	Narayan	Null
	Joyce	A	Borg	Null
	Ahmad	V	Jabbar	Null
	James	E	Borg	Head quartes

۵-۶ نمونه‌ای از چند پرس‌وجو

در این بخش چند مثال از پرس‌وجو با کمک جبر رابطه‌ای بر روی مدل رابطه‌ای Company ارائه می‌شود.

Query1- نام و آدرس همه کارمندان دپارتمان "research".

Research- depth ← $\sigma_{Dname=\ \text{“research”}}$ (Department)

Result ← $\Pi_{Fname,\ Lname,\ Address}$ (Research_depth ∞ Employee)

Dnumber= DNO

Query2- برای هر پروژه واقع در "Stanford"، لیستی از شماره پروژه‌ها، شماره دپارتمانی که پروژه را کنترل می‌کند و نام و آدرس و تاریخ تولید مدیر دپارتمان بدهید.

Stafford_projs← $\sigma_{Plocation=\ \text{“Stafford”}}$ (Project)

Contr_dept ← (Stafford_projs ∞ Department)

Dnum= Dnumber

Result ← $\Pi_{Pname,\ Dname,\ Lname,\ address,\ Bdute}$ (contr_dept ∞ Employee)

MGSSN= SSN

Query3- نام همه کارمندانی که روی همه پروژه‌ها تحت کنترل دپارتمان شماره ۵ کار می‌کنند.

$Dept\_projs\ (PNO) \leftarrow \Pi_{Pnumber}\ (\sigma_{Dnum=s}\ (project))$

$Emp\_proj\ (SSN, PNO) \leftarrow \Pi_{ESSN,\ PNO}\ (Works\_on)$

$Result_1 \leftarrow Emp\_proj \div Depts\_projs$

$Result \leftarrow \Pi_{Lname,\ Fname}\ (Result^{*}Employee)$

Query4- لیستی از شماره پروژه‌هایی شامل کارمندی به نام Smith که او به عنوان کارمند یا مدیر دپارتمانی که پروژه را کنترل می‌کند هم هست.

$Smith\ (SSN) \leftarrow \Pi_{SSN}\ (\sigma_{Lname=\ \text{“Smith”}}\ (EMPLOYEE))$

$Smith\_worker\_proj \leftarrow \Pi_{PNO}\ (Smith^{*}\ works\_on)$

$Mers \leftarrow \Pi_{Lname,\ Dnumber}\ (EMPLOYEE \underset{SSN=\ MegSSN}{\infty} DEPARTMENT)$

$Smith\_managed\_depts \leftarrow \Pi_{Dnumber}\ (\sigma_{Lname=\ \text{“Smith”}}\ (Megs))$

$Smith\_meg\_projs \leftarrow \Pi_{Pnumber}\ (Smith\_managed\_depts\ ^{*}\ Project)$

$Result \leftarrow (Smith\_worker\_projs\ U\ Smith\_mer\_projs)$

Query5- نام همه کارمندانی که ۲ یا بیش‌تر عائله‌مندی دارند.

$T_1\ (SSN, No\_of\_depts) \leftarrow ESSN\, f_{\ amount\ dependent\_name}\ (Dependent)$

$T_2 \leftarrow \sigma_{No\_of\_dept \geq 2}\ (T_1)$

$Result \leftarrow \Pi_{Lname,\ Fname}\ (T_2^{*}\ Employee)$

Query6- نام همه کارمندانی که هیچ عائله‌مندی ندارند.

$All\_emps \leftarrow \Pi_{SSN}\ (Employee)$

$Emps\_with\_depts_{\ (SSN)} \leftarrow \Pi_{SSN}\ (Dependent)$

$Emps\_without\_deps \leftarrow (All\_emps\_Emps\_with\_deps)$

$Result \leftarrow \Pi_{Lname,\ Fname}\ (Emps\_without\_deps^{*}\ Employee)$

Query7- نام همه مدیرانی که حداقل یک عائله‌مندی دارند.

$Mgrg\ (SSN) \leftarrow \Pi_{MgrSSN}\ (department)$

$Emps\_with\_deps\ (SSN) \leftarrow \Pi_{ESSN}\ (Department)$

$Mgrs\_with\_deps \leftarrow (Mgrs \cap\ Emps\_with\_deps)$

$Result \leftarrow \Pi_{Fname,\ Lname}\ (Mgrs\_with\_deps\ Employee)$

۷ نگاشت ER و EER به رابطه‌ای

مدل داده‌ای که با استفاده شمای ER تشکیل شده است، قابل تبدیل به مجموعه‌ای از جدول‌هاست. به ازای هر مجموعه موجودیت و به ازای هر مجموعه رابطه موجود در نمودار ER یک جدول منحصر به فرد وجود دارد که نام آن مجموعه موجودیت یا مجموعه رابطه به جدول مزکور تخصیص می‌یابد. هر جدول دارای چندین ستون است که هر ستون نام یکتایی دارد.

پایگاه داده رابطه‌ای و مدل ER هر دو نمایش منطقی و انتزاعی از یک مساله واقعی هستند. از آن‌جا که هر دو مدل مذکور از اصول طراحی مشابهی پیروی می‌کنند، می‌توانیم یک طراحی ER را به یک پایگاه داده‌ای رابطه‌ای تبدیل نماییم. تبدیل نمایش یک DB از مدل ER به قالب جداول مبنا و اساس مساله به دست آوردن پایگاه داده رابطه‌ای از مدل ER است.

ما در این‌جا از مثال پایگاه داده‌ای Company برای تشریح الگوریتم نگاشت استفاده می‌کنیم. برای این منظور مدل ER و مدل رابطه‌ای ارائه شده برای Company را در نظر بگیرید. این مدل‌های در فصول قبلی ارائه شده‌اند. ابتدا الگوریتم تبدیل مدل ER به مدل رابطه‌ای در ۷ مرحله ارائه شده و سپس این الگوریتم برای تبدیل مدل EER به مدل رابطه‌ای توسعه داده می‌شود.

۷-۱ تبدیل ER به رابطه‌ای

همان‌طور که گفته شد الگوریتم تبدیل ER به مدل رابطه‌ای در ۷ مرحله اجرا می‌شود و در هر مرحله یکی از اجزای مدل ER به مدل رابطه‌ای تبدیل می‌شود. تبدیل اجزای ER باید به ترتیب ارائه شده اجرا شود.

۷-۱-۱ موجودیت قوی

در این مرحله هر موجودیت قوی به یک جدول در مدل رابطه‌ای تبدیل می‌شود. مجموعه موجودیت قوی E را با صفات $a_1, a_2, \ldots a_n$ در نظر می‌گیریم. این مجموعه موجودیت را با جدولی با نام E نشان می‌دهیم که دارای n ستون جدا از هم و هر ستون متناظر با یکی از صفات مجموعه نهاد E است. هر سطر در این جدول هم متناظر با یک نمونه موجودیت از مجموعه موجودیت E می‌باشد.

صفات ساده و مرکب را هم ضمیمه جداول می‌کنیم. در مورد صفات مرکب، اجزای صفات به طور مجزا در جدول اصلی قرار می‌گیرند. باید توجه داشت که هر جدول باید یک کلید اصلی داشته باشد.

دراین مرحله، برای مدل Company جداول EMPLOYEE، DEPARTMENT و PROJECT در مدل رابطه‌ای ایجاد شده است. ما در این مرحله تنها SSN و DNumber و PNumber را به عنوان کلیدهای اصلی برای رابطه‌های EMPLOYEE و DEPARTMENT و PROJECT در نظر می‌گیریم.

۷-۱-۲ موجودیت ضعیف

فرض کنید W یک مجموعه موجودیت ضعیف با ویژگی‌های $a_1, a_2, \ldots a_m$ باشد. فرض کنید E هم مجموعه موجودیت قوی‌ای باشد که W به آن وابسته است. در این صورت یک رابطه مانند R تعریف کنید شامل کلیه ویژگی‌های ساده و اجزای صفات مرکب از مجموعه موجودیت ضعیف W و کلید اصلی مجموعه موجودیت غالب یعنی E را به همراه کلید مجموعه موجودیت ضعیف اگر وجود داشت، به عنوان کلید اصلی جدول R در نظر بگیرید.

در مثال مورد بحث، رابطه DEPENDENT را در این مرحله از مجموعه موجودیت ضعیف DEPENDENT ایجاد کردیم که شامل کلید اصلی SSN از رابطه EMPLOYEE (که مجموعه موجودیت غالب است) می‌باشد و آن را به عنوان کلید خارجی رابطه DEPENDENT در نظر گرفتیم و آن را به ESSN تغییر نام دادیم گرچه لازم نیست حتماً این کار را انجام دهیم. پس کلید اصلی رابطه DEPENDENT ترکیبی از ESSN و DEPENDENT_name شد زیرا DEPENDENT_name کلید اصلی رابطه DEPENDENT بود.

۷-۱-۳ ارتباط دوتایی یک به یک

برای هر نوع ارتباط دوتایی یک به یک مانند R در شمای ER، روابط S و T را که متناظر با نوع موجودیت‌های شرکت کننده در T می‌باشند، در نظر بگیرید. یکی از رابطه‌های S یا T را (مثلاً S) انتخاب کنید. بهتر است از بین S و T موجودیتی انتخاب شود که به صورت کلی در رابطه شرکت کرده است. کلید اصلی رابطه T را به‌عنوان کلید خارجی در S قرار داده و در صورتی که ارتباط صفاتی دارد، آن صفات را به رابطه S اضافه کنید.

در مثال مورد بحث، در مورد نوع ارتباط MANAGES 1:1 از مدل ER مجموعه موجودیت DEPARTMENT در نقش رابطه S می‌باشد زیرا به صورت کلی شرکت کرده است. (هر دپارتمان یک مدیر دارد) کلید اصلی رابطه EMPLOYEE را به عنوان کلید خارجی رابطه DEPARTMENT به آن اضافه می‌کنیم و آن را از SSN به MGESSN تغییر نام می‌دهیم. ما هم‌چنین صفت ساده StartDate از ارتباط MANAGES را هم به رابطه DEPARTMENT اضافه می‌کنیم و آن را به MGRStartDate تغییر نام می‌دهیم.

۷-۱-۴ ارتباط دوتایی یک به چند

برای هر نوع ارتباط دوتایی یک به چند مانند R، روابط S و T را در نظر بگیرید به طوری که S مجموعه موجودیت شرکت کننده در طرف N نوع ارتباط و T طرف 1 آن باشد. کلید اصلی T را به عنوان کلید خارجی در S قرار داده و صفات ارتباط را هم به S اضافه کنید.

در مثال مورد بحث، ارتباطات works_for، CONTRLOS، SUPERVISION از نوع 1: N می‌باشند. برای works_for کلید اصلی DNumber از DEPARTMENT را به عنوان کلید خارجی EMPLOYEE در نظر گرفته و آن را به DNO تغییر نام می‌دهیم و به جدول EMPLOYEE اضافه می‌کنیم.

برای SUPERVISION کلید اصلی رابطه EMPLOYE (SSN) را به عنوان کلید خارجی خودش در نظر می‌گیریم زیرا ارتباط بازگشتی بوده است و آن را به SuperSSN تغییر نام می‌دهیم.

برای CONTRLOS کلید اصلی DEPARTMENT را بعد از تغییر نام به DNUM به PROJECT اضافه می‌کنیم.

۷-۱-۵ ارتباط دوتایی چند به چند

برای نوع ارتباط دوتایی M:N مانند R یک رابطه جدید مانند S ایجاد می‌کنیم و کلیدهای اصلی روابط متناظر با موجودیت‌های ارتباط‌ها را به عنوان کلید خارجی S منظور می‌کنیم. کلید اصلی جدول S ترکیبی از کلیدهای خارجی می‌باشد. صفات ارتباط R را نیز به S اضافه می‌کنیم.

در مثال مورد بحث، ارتباط Works_ON یک نوع ارتباط M:N است که برای آن یک رابطه‌ای به نام Works_ON ایجاد کرده و کلیدهای اصلی رابطه‌های PROJECT و EMPLOYEE را به (ESSN, PNO) تغییر نام داده و به این رابطه اضافه کردیم.

کلید اصلی رابطه Works_ON ترکیبی از PNO و ESSN شده است.

۷-۱-۶ صفات چندمقداری

مشاهده کردیم که متناظر با هر صفت در یک موجودیت و یا ارتباط یک ستون در جدول مربوطه قرار می‌گیرد. اما صفات چندمقداری از این قاعده مستثنی هستند. برای یک صفت چندمقداری جدول جدیدی ایجاد می‌کنیم که یک ستون آن نشان دهنده مقادیر این صفت و ستون دیگر آن کلید اصلی رابطه‌ای است که این صفت به آن تعلق دارد.

برای هر صفت چندمقداری A رابطه جدید R را اضافه می‌کنیم. این رابطه یک صفت متناظر با A دارد. کلید اصلی رابطه متناظر با موجودیت اصلی را به عنوان کلید خارجی در جدول جدید اضافه می‌کنیم. کلید اصلی جدول R ترکیب A و کلید خارجی است.

در مثال مورد بحث ما یک رابطه با نام DEPT_LOCATIONS ایجاد می‌کنیم. چون صفت DLOCATION یک صفت چندمقداری است. صفت کلید اصلی DNUMBER هم به عنوان کلید خارجی این رابطه می‌شود. پس ترکیب DLOCATION و DNUMBER به عنوان کلید اصلی رابطه DEPT_LOCATION خواهد شد.

۷-۱-۷ ارتباط Nتایی

برای هر نوع ارتباط Nتایی مانند R که (N>2)، رابطه جدید S را تولید کنید و کلید اصلی روابط متناظر با موجودیت‌های شرکت کننده در R را به عنوان کلید خارجی در S اضافه کنید. کلید اصلی رابطه جدید S برابر ترکیب کلیدهای خارجی اضافه شده به این جدول است.

۲-۷ تبدیل تخصیص یا تعمیم

دو روش برای تبدیل یک نمودار ER شامل تعمیم به مدل رابطه‌ای وجود دارد. برای سادگی بحث تنها یک سطح از سطوح تعمیم را برای دو نوع حساب پس‌انداز و جاری در سیستم بانک در نظر می‌گیریم.

- ابتدا جدولی برای مجموعه موجودیت سطح بالاتر ایجاد می‌کنیم. برای هر یک از مجموعه موجودیت‌های سطح پایین‌تر نیز جدولی ایجاد خواهد شد. این جدول به ازای هر ویژگی مجموعه موجودیت متناظر یک ستون دارد. علاوه بر آن ویژگی‌های کلید اصلی مجموعه موجودیت سطح بالاتر نیز دارای یک ستون در جدول مزبور هستند. بنابراین برای سیستم بانکی مورد مثال سه جدول خواهیم داشت:
  - جدول account با صفات account_number و balance
  - جدول saving_account با صفات account_number و overdraft_amount
- اگر تعمیم تام و جدا از هم باشد (به این مفهوم که اگر موجودیتی هیچ یک از مجموعه موجودیت‌های سطح پایین‌تر متعلق ندارد، عضو مجموعه موجودیت سطح بالاتر نیز نباشد و نیز هر عضو از مجموعه موجودیت سطح بالاتر تنها متعلق به یکی و فقط یکی از مجموعه موجودیت‌های سطح پایین‌تر باشد) در آن صورت می‌توان نمایش جدولی دیگری نیز ارائه نمود. در این روش هیچ جدولی برای مجموعه موجودیت سطح بالاتر در نظر نمی‌گیریم. به جای آن برای هر مجموعه نهاد سطح پایین‌تر یک جدول در نظر می‌گیریم.
- این جدول به ازای هر صفت مجموعه موجودیت متناظر خود دارای یک ستون است.
- علاوه بر آن به ازای هر صفت مجموعه موجودیت سطح بالای خود نیز یک ستون دارد. با این روش برای مثال مورد بحث دو جدول خواهیم داشت:
  - جدول saving_account با صفات balance و account_number و insert_rate
  - جدول checking_account با صفات balance، account_number و overdraft_amount

اگر از روش دوم برای یک تعمیم هم‌پوشان استفاده کنیم برخی مقادیر مانند balance بدون آن‌که نیازی باشد دوباره ذخیره می‌شوند. اگر تعمیم تام نباشند برخی از حساب‌ها در هیچ یک از جدول‌ها ذخیره نمی‌شوند.

۸ زبان SQL

در این فصل پرکاربردترین و مهم‌ترین زبان پرس‌وجوی ساختیافته در سیستم‌های پایگاه داده رابطه‌ای یعنی SQL صحبت خواهد شد. ترکیبی از جبر رابطه‌ای و حساب رابطه‌ای در ساخت SQL به کار رفته است. این زبان شامل بخش‌هایی برای تعریف ساختار داده‌هاست. امکان تغییر در داده‌های موجود در پایگاه داده را دارد و برای مشخص نمودن محدودیت‌های امنیتی نیز جنبه‌هایی را در نظر گرفته است. پیاده‌سازی‌های مختلف از SQL ممکن است در جزئیات با هم تفاوت‌هایی داشته باشند.

زبان‌های دیگری نیز در ارتباط با پرس‌وجو و کار با پایگاه داده وجود دارند که آن‌ها هم دارای مشخصه‌های زبان‌های تجاری می‌باشند. از جمله این زبان‌ها می‌توان به QBE و QUEL اشاره کرد. همچنین Datalog زبانی از همین خانواده است که در سیستم‌های تحقیقاتی مورد استفاده قرار می‌گیرد. هر سه زبان در سیستم‌های پایگاه داده‌ای تحقیقاتی کاربرد دارند.

۸-۱ مقدمه

زبان SQL امروزه به عنوان زبان استاندارد پایگاه داده‌ای رابطه‌ای شناخته می‌شود. نگارش‌های فراوانی از این زبان وجود دارد. نگارش اصلی آن در آزمایشگاه تحقیقاتی IBM تهیه شد. این زبان که در اصل sequel نام دارد، به عنوان بخشی از پروژه سیستم R در اوایل دهه ۱۹۷۰ پیاده‌سازی شد. این زبان (Sequel) به تدریج تکامل یافت و نام آن به SQL، به معنای زبان پرس‌وجوی ساختار یافته تغییر یافت. برخی از قابلیت‌های این زبان عبارتست از:

- زبان تعریف داده (DDL): این بخش از SQL دستوراتی برای تعریف شمای رابطه‌ها، حذف رابطه‌ها، ساختن اندیس‌ها و تغییر شمای رابطه‌ها فراهم می‌نماید.
- زبان محاوره‌ای دستکاری داده: شامل یک زبان پرس‌وجوی مبتنی بر جبر رابطه‌ای و حساب رابطه‌ای است. علاوه بر آن دستوراتی برای اضافه کردن یک رکورد به جداول پایگاه داده، حذف رکورد از آن‌ها و تغییر مقادیری از رکوردهای آن‌ها را دارد.
- زبان دستکاری داده جانشان شده (Embedded DML): فرم جانشان شده از SQL به منظور استفاده شدن درون یک زبان برنامه‌نویسی همه منظوره مانند C، Delphi، ... ارائه شده است.
- تعریف دیدگاه: بخش DDL از SQL که شامل دستوراتی برای تعریف دیدگاه‌ها می‌باشد.
- اعتبارسنجی (Authorization): DDL شامل دستوراتی است که قوانین دسترسی به دیدگاه‌ها و جداول را مشخص می‌کنند و بر صحت آن‌ها نظارت می‌نمایند.
- جامعیت: تغییراتی (به روزرسانی‌هایی) از پایگاه داده که محدودیت‌های جامعیت را نقض می‌کنند، اجازه انجام شدن ندارند.
- کنترل تراکنش: SQL دستوراتی برای مشخص کردن آغاز و انتهای تراکنش‌ها دارد.

در این فصل در ارتباط با DML و مفاهیم پایه‌ای DDL از زبان SQL بحث خواهیم کرد.

محیط‌هایی که در مثال‌های این فصل استفاده می‌شوند یا محیط بانکی‌اند یا COMPANY که شمای رابطه‌های آن‌ها در ادامه آمده است. شمای رابطه‌ای محیط بانکی به‌صورت زیر است.

BRANCH_SCHEMA= (branch_name, branch_city, assets)

CUSTOMER_SCHEMA= (customer_name, customer_street, customer_city)

LOAN_SCHEMA= (branch_name, Loan_number, amount)

BORROWER_SCHEMA= (customer_name, loan_number)

ACCOUNT_SCHEMA= (branch_name, account_number, balance)

DEPOSITOR_SCHEMA= (customer_name, account_number)

شمای رابطه‌ای COMPANY به‌صورت زیر است.

EMPLOYEE_SCHEMA= (FNAME, MINT, LNAME, BDATE, ADDRESS, SEX, SALARY, SUPERSSN, DNO)

DEPARTMENT_SCHEMA= (DNAME, DNUMBER, MGRSSN, MGRSTART DATE)

DEPT_LOCATION_SCHEMA= (DNUMBER, PNUMBER, PLOCATION, DNUM)

PROJECT_SCHEMA= (ESSN, PNO, HOURS)

WORKS_ON_SCHEMA= (ESSN, PNO, HOURS)

DEPENDENT_SCHEMA= (ESSN, DEPARTMENT_NAME, SEX, BDATE, RELATIONSHIP)

۲-۸ تعریف شما در SQL

برای تعریف یک مدل رابطه‌ای در SQL از دستور CREATE TABLE استفاده می‌کنیم. فرم کلی این دستور به‌صورت زیر است.

```
CREATE TABLE r (A1D1, A2D2, …, AnDn)
<Integrity constraint1>
...
<Integrity constraintk>
```

در این تعریف r نام جدول و هر یک از Aiها نام یکی از صفات جدول r هستند. همچنین Di دامنه نوع مقادیر معتبر در دامنه A_i است یعنی مقادیری که می‌توانند به صفت A_i نسبت داده شوند همان مقادیر موجود در دامنه D_i می‌باشند.

محدودیت‌های ممکن مجاز <integrity constraint> شامل موارد زیر است.

```
Primary key (Aj1, Aj2, …, Ajm)
Check (p)
```

کلید اصلی (Primary key) خاطر نشان می‌سازد که ویژگی‌های Aj1, Aj2, …, Ajm با هم کلید اصلی جدول را می‌سازند. مشخص نمودن یک کلید اصلی در جدول مشخص می‌کند که شرط p باید توسط همه رکوردهای جدول تأمین و رعایت شود.

تعاریف زیر با استفاده از DDL، پایگاه داده بخشی از سیستم بانکی مورد مثال این فصل و در قسمت بعد مثال COMAPNY را تعریف می‌کنند.

```
CREATE TABLE customer
            (Customer_name char (20) not null,
            Customer_street char (30)
            Customer_city char (30)
            Primary key (customer_name)
CREATE TABLE branch
            (Branch_name char (15) not null,
            Branch_city char (30),
            Assets integer,
            Primary key (branch_name),
            Check (assets>= 0))
CREATE TABLE account
            (Account_number char (10) not null,
            Branch_name char (15),
            Balance integer,
            Primary key (account_number),
            Check (balance>= 0))
CREATE TABLE depositor
            (Customer_name char (20) not null,
            Account_number char (10) not null,
            Primary key (customer_name, customer number))
```

ویژگی‌هایی که به عنوان کلید اصلی معرفی می‌شوند باید دارای مقادیر یکتا و not null باشند. یکتایی به این مفهوم که دو رکورد متمایز از جدول وجود نداشته باشند که مقادیر آن‌ها برای همه ویژگی‌های اصلی یکسان باشد. هر تغییری که باعث شود یکی از دو شرط null نبودن و یکتا نبودن در مورد کلید اصلی نقض گردد، موجب بروز خطا می‌شود و سیستم پایگاه داده از انجام و اعمال آن تغییر بر روی جدول جلوگیری می‌کند.

یکی از کاربردهای متداول check این است که تضمین می‌نماید که مقادیر ویژگی‌ها شرایط مشخص شده‌ای را دارا باشند و این سبب قدرتمندتر شدن سیستم کنترل نوع‌های داده‌ای می‌شود. به عنوان نمونه در ساخت جدول account وجود check مانع از منفی شدن مقدار موجودی یک حساب می‌شود.

```
CREATE TABLE EMPLOYEE
            (FNAME VARCHAR (15) NOT NULL,
            MINT CHAR,
            LNAME VARCHAR (15) NOT NULL,
            SSN CHAR (9) NOT NULL,
            BDATE DATE,
            ADDRESS VARCHAR (30),
            SEX CHAR,
            SALARY DECIMAL (10, 2),
            SUPERSSN CHAR (9),
            DNO INT NOT NULL,
            PRIMARY KEY (SSN)
            FOREIGN KEY (SUPER SSN) REFERENCES EMPLOYEE (SSN),
            FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNUMBER));
CREATE TABLE DEPARTMENT
            (DNAME VARCHAR (15) NOT NULL,
            DNUMBER INT NOT NULL,
            MGRSSN CHAR (9)
            MGRSTORDATE DATE,
            PRIMARY KEY (DNUMBER),
            UNIQUE (DNAME),
            FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE (SSN));
CREATE TABLE DEPT_LOCATIONS
            (DNUMBER INT NOT NULL,
            DLOCATION VARCHAR (15) NOT NULL,
            PRIMARY KEY (DNUMBER, DLOCATION),
            FOREIGN KEY (DNUMBER REFERENCES DEPARTMENT (DNUMBER));
CREATE TABLE PROJECT
            (PNAME VARCHAR (15) NOT NULL,
            PNUMBER INT NOT NULL,
            PLOCATION VARCHAR (15),
```

```
            DNUM INT
            PRIMARY KEY (PNUMBER),
            UNIQUE KEY (PNAME),
            FOREIGN KEY (DNUM) REFERENCES DEPARTMENT (DNUMBER));
CREATE TABLE WORKS_ON
            (ESSN CHAR (9) NOT NULL,
            PNO INT NOT NULL,
            HOURS DECIMAL (3, 7) NOT NULL,
            PRIMARY KEY (ESSN, PNO),
            FOREIGN KEY (ESSN) REFERENCES EMPLOYEE (SSN),
            FOREIGN KEY (DNO) REFERENCES PROJECT (PNUMBER),
CREATE TABLE DEPENDENT
            (ESSN CHAR (9) NOT NULL,
            DEPENDENT_NAME VARCHAR (15) NOT NULL,
            SEX CHAR,
            BDATE DATE,
            RELATIONSHIP VARCHAR (18),
            PRIMARY KEY (ESSN), DEPARTMENT_NAME)
            FOREIGN KEY (ESSN) REFERENCES EMPLOYEE (SSN));
```

یک جدول وقتی ایجاد می‌شـود در ابتدا خالی اسـت و فاقد رکورد می‌باشـد. برای قرار دادن داده‌ها درون جدول می‌توانیم از دسـتورات MDL از جمله دسـتور Insert اسـتفاده نماییم. علاوه‌بر رکوردهای جداول، ممکن اسـت که طراح بخواهد شـمای برخی از جداول و یا شمای پایگاه داده را تغییر دهد. در زبان SQL برای این منظور نیز دستوراتی ارائه شده است.

برای حذف یک جدول از یک پایگاه داده دسـتور DROP TABLE در SQL وجود دارد. این دسـتور همه اطلاعات مرتبط با جدول مورد نظر را از پایگاه داده حدق می‌نماید.

دسـتور DROP TABLE r بسـیار گسترده‌تر و کلی‌تر از دستور DELETE FROM r می‌باشد چرا که دستور DELETE بـا حذف همه رکوردهای موجود در جدول r تنها جدول خالی را باقی می‌گذارد. یعنی شـمای جدول و اطلاعات مرتبط با آن از پایگاه داده حذف نمی‌شوند.

بلکه تنها داده‌های موجود در جدول حذف می‌گردند. این در حالی اسـت که با دستور drop همه آن‌چه که در جدول r می‌باشد از قبیل شـمای جدول از بین می‌رود. پس از دستور drop دیگر امکان اضافه کردن رکورد از طرق insert به جدول وجود ندارد مگر آن‌که با استفاده از دستور create table جدول مجدداً ساخته شود.

دسـتوری به نام ALERT TABLE امکان افزودن یک یا چند صـفت به مجموعه صـفات جدول را فراهم می‌نماید. تعداد صفات اضافه شده با این دستور برای همه رکوردهای موجود در جدول null می‌شود.

فرم کلی دستور به صورت زیر است:

ALERT TABLE r add A D

که در آن r نام جدول، A نام ویژگی افزوده شده به جدول و D دامنه ویژگی افزوده شده است. با این دستور (ALERT TABLE) هم چنین می‌توانیم یک صفت را از مجموعه صفات یک جدول حذف کنیم. برای این کار این دستور را به همراه کلمه drop به صورت زیر به کار می‌گیریم:

ALERT TABLE r drop A

که r نام جدول و A نام یکی از صفات جدول r است.

در ضمن چناچه کلید خارجی از جدول مرجع حذف شود و یا تغییر دیگری کند با استفاده از دو عبارت [on delete cascade] و یا [on update cascade] تغییرات به جداول مراجعه کننده سرایت می‌کنند.

۸-۳ دامنه‌های پیش‌فرض

دامنه‌های زیر به‌طور پیش‌فرض در SQL موجود می‌باشد و در تعریف صفات یک جدول می‌تواند استفاده شود.

- Char (n): یک رشته کاراکتری با طول ثابت که این طول (n) توسط کاربر مشخص می‌شود.
- Varchar (n): رشته‌های با طول متغیر حداکثر به طول n
- Int: اعداد صحیح
- Small int: اعداد صحیح کوچک
- Numeric (p,d): اعداد اعشاری، P حداکثر تعداد رقم‌ها و d تعداد رقم‌های اعشاری.
- Floot (n): اعداد اعشاری با دقت حداقل n رقم.
- NOT NULL: چنانچه در انتهای تعریف یک فیلد به کار رود صفت مورد نظر نمی‌تواند مقدار null داشته باشد.
- Time, Date: بیانگر تاریخ و زمان می‌باشند.

۸-۴ ساختار پایه دستور Select

یک پایگاه داده‌ای رابطه‌ای شامل مجموعه‌ای از رابطه‌هاست که هر یک از آن‌ها نام منحصر به فردی دارند. جهت تعریف پرس‌وجو در زبان SQL، دستور Select ارائه شده است. این دستور توانمندی جبر رابطه‌ای برای تعریف پرس‌وجوهای مختلف را پوشش می‌دهد. علاوه‌بر آن می‌تواند پرس‌وجو را بر روی دید (View) تعریف نماید.

ساختار پایه‌ای یک عبارت Select در SQL شامل سه شبه جمله Select، From و Where می‌باشد که به‌صورت زیر توصیف می‌شود.

- شبه جمله select معادل عملگر تصویر (Project) در جبر رابطه‌ای است. برای آن‌که صفات مطلوب خود را در نتیجه پرس‌وجو ببینیم از select استفاده می‌کنیم.

- شبه جمله from معادل عملگر ضرب کارتزینی یا join در جبر رابطه‌ای است. لیست جداولی که باید در عبارت مورد نظر مورد بررسی قرار گیرند، توسط دستور from مشخص می‌گردد.
- شبه جمله where معادل شرط انتخاب در جبر رابطه‌ای است. Where شامل شرط‌هایی است که بر روی صفات روابط موجود در from گذاشته می‌شود.

مفهوم select و انتخاب در SQL و جبر رابطه‌ای متفاوت است و این کلمه در دو زبان فوق دو معنا و عملکرد متفاوت دارد.

یک پرس‌وجو متعارف در زبان SQL به فرم زیر است.

```
Select <attribute list>
From <table list>
Where <condition>
```

این پرس‌وجو معادل عبارت جبر رابطه‌ای زیر است.

$$\Pi_{A1, A2, \ldots, An} (\sigma_p (r_1 \times r_2 \times \ldots \times r_n))$$

هر A_i نشان دهنده یک ویژگی و هر r_i نشان دهنده یک رابطه است. عبارت p شرطی است که برای انتخاب رکوردها از جداول بر روی آن‌ها اعمال می‌شود. اگر شبه جمله where را حذف کنیم فرض می‌شود که شرط p همیشه درست باشد. یعنی عمل انتخاب برای همه رکوردها انجام می‌پذیرد. برخلاف جبر رابطه‌ای نتیجه پرس‌وجوی SQL ممکن است شامل رکوردهای تکراری باشد.

۸-۴-۱ شبه جمله Select

نتیجه عمل select یک رابطه است. به عنوان مثال می‌خواهیم نام همه شعبه‌های پرداخت کننده وام را بدانیم.

```
Select     branch_name
From  loan
```

نتیجه این عمل رابطه‌ای است مشتمل بر یک صفت با نام branch_name.

در زبان‌های فرمال پیشین نظیر جبر رابطه‌ای مبنای کار بر آن بود که رابطه به عنوان مجموعه‌ای از رکوردها در نظر گرفته می‌شود. بدیهی بود که در چنین شرایطی هیچ رکورد تکراری‌ای در رابطه حاصل وجود ندارد. زبان SQL به دلیل این که در عمل حذف تکرارها در یک رابطه عملی نسبتاً زمان‌گیر است و هزینه زمانی تقریباً زیادی دارد، اجازه می‌دهد که در روابط حاصل از اجرای عبارت SQL، تکرار وجود داشته باشد.

در مواردی که می‌خواهیم تکرارها را از جدول نتیجه یک select حذف کنیم کلمه کلیدی distinct را پس از select می‌نویسیم.

```
Select     distinct branch_name
From  loan
```

به این ترتیب می‌توانیم با کلمه کلیدی all به صراحت مشخص کنیم که می‌خواهیم در نتیجه یک عبارت SQL تکرارها حذف نشوند.

```
Select     all branch_name
From  loan
```

چون به صورت پیش فرض تکرارها از یک جدول حذف نمی‌شوند به کارگیری all در عبارات select ضرورتی ندارد. برای آن‌که همه ویژگی‌های یک جدول در جدول حاصل از عبارات select قرار داشته باشند، می‌توانیم به جای نوشتن نام تک تک ویژگی‌ها از نماد "*" استفاده کنیم.

شبه جمله select می‌تواند شامل عبارات محاسباتی مشتمل بر عملگرهای +, –, *, /, باشد که بر روی مقادیر و صفات چندتایی‌ها عمل می‌کنند.

```
Select     branch_name, loan_number, amoun*loo
From  loan
```

این پرس‌وجو جدولی مانند loan را به عنوان نتیجه برمی‌گرداند که میزان وام‌ها در آن ۱۰۰ برابر شده است.

۸-۴-۲ شبه جمله Where

برای روشن شدن عملکرد where پرس‌وجوی زیر را در نظر می‌گیریم.

فرض کنید که می‌خواهیم شماره وام‌هایی که در شعبه perrydige پرداخت شده و میزان آن‌ها بیش از ۱۲۰۰ بوده را پیدا کنیم.

```
Select     Loan_number
      From  Loan
      Where       branch_name= “perrydige” and amount> 1200 $
```

در SQL به جای علامت‌های ∧ و ∧ و ¬ به ترتیب از رابط‌های منطقی and و or و not استفاده می‌شود. عملگرهای مقایسه‌ای می‌توانند شامل < و ≤ و = و ≥ و > و <> باشند. عملگر between نیز یک عملگر مقایسه‌ای دیگر است که برای ساده‌سازی whereهایی که شرط آن‌ها بیانگر کوچکتر مساوی بودن از مقداری و بزرگتر مساوی بودن از مقدار دیگر باشد، کاربرد دارد.

- شماره وام‌هایی که میزان آن‌ها بین ۱۰۰ تا ۵۰۰ دلار بوده است.

```
Select     Loan_number
From  Loan
Where amount between 100 and 500
```

این پرس‌وجو معادل پرس‌وجوی زیر است.

```
Select     Loan_number
From  Loan
```

```
Where      amount ≤500 and amount≥100
```

۸-۴-۳ شبه جمله From

شبه جمله from به خودی خود به معنای ضرب کارتزینی رابطه‌های موجود در بخش from است. چون الحاق طبیعی عمل ترکیبی از ضرب کارتزینی، انتخاب و تصویر است، لذا پیاده‌سازی الحاق طبیعی توسط پرس‌وجوهای SQL کاری نسبتاً ساده است. مثال‌های زیر را در نظر بگیرید.

- نام مشتریانی را که از بانک وام گرفته‌اند، به همراه شماره وام آن‌ها بیابید.

```
Select     customer_name, Loan.Loan_number
From borrower, Loan
Where      borrower.Loan_number = Loan.Loan_number
```

- تاریخ تولد و آدرس کارمندانی که نام آن‌ها John B.smith می‌باشد را بیابید.

```
SELECT     BDATE, ADDRESS
FROM            EMPLOYEE
WHERE      FNAME= "JOHN" AND MINT= "B" AND LNAME="SMITH";
```

نام و آدرس همه کارمندانی که برای دپارتمان Research کار می‌کنند.

```
SELECT     FNAME, LNAME, ADDRESS
FROM       EMPLOYEE, DEPARTMENT
WHERE      DNAME=" RESEARCH" AND DNO= DNUMBER;
```

برای هر پروژه واقع در stafford، لیست شماره پروژه و شماره دپارتمانی که آن پروژه را کنترل می‌کند و نام خانوادگی و آدرس و تاریخ تولد مدیر آن دپارتمان را بیابید.

```
SELECT    PNUMBER, DNUMBER, LNAME, ADDRESS, BDATE
FROM       PROJECT, DEPARTMENT, EMPLOYEE
WHERE    PLOCATION="STAFFORD" AND DNUM=DNUMBER AND MGRSSN=SSN;
```

۸-۴-۴ عملگر تغییر نام

زبان SQL امکان تغییر نام جداول و صفات را فراهم نموده است. برای این کار از کلمه AS به فرم زیر استفاده می‌شود.

Old_name AS new_name

که از AS می‌توانیم هم در قسمت select و هم در قسمت from استفاده کنیم.

به عنوان مثال حاصل پرس‌وجوی زیر جدولی با دو صفت به نام‌های customet_name و Loan_id خواهد بود.

```
Select     distinct customer_name, borrower.Loan_number AS Loan_id
From borrower AS T, Loan AS S
Where      T.Loan_number = S.Loan_number
```

مشاهده شد که تعریف یک کلمه متغیر چندگانه‌ای مانند T یا S در بخش from با قرار دادن نام متغیر پس از نام جدولی که با آن مرتبط می‌گردد انجام گرفت.

وقتی عباراتی به صورت relation_name.attribute_name می‌نویسیم نام جدول در واقع متغیر چندگانه‌ای است که به صورت ضمنی تعریف شده است.

یکی از مهم‌ترین کاربردهای متغیر چندگانه‌ای، مقایسه دو رکورد از یک جدول است.

- نام همه شعبه‌هایی را بیابید که دارایی آن‌ها دست کم از یکی از شعبه‌های واقع در Brooklyn کم‌تر باشد.

```
Select      distinct T.branch_name
From  branch AS T, branch AS S
Where       T.assets > S.assets and S.branch_city= "Brooklyn";
```

توجه داشته باشید که در این پرس‌وجو نمی‌توانستیم از branch.Assets استفاده کنیم زیرا در آن صورت مشخص نبود که ارجاع به کدام جدول branch مد نظر ما بوده است.

- نام و نام خانوادگی کارمندان و نام و نام خانوادگی مدیران آن‌ها.

```
SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM        EMPLOYEE AS E, EMPLOYEE AS S
WHERE       E.SUPERSSN = S.SSN;
```

پرس‌وجوی زیر یک پرس‌وجوی نادرست است زیرا دو جدول با هم JOIN شده‌اند اما شرط JOIN آورده نشده یعنی شبه جمله WHERE ندارد.

- شماره شناسنامه همه کارمندان و نام دپارتمان‌هایی که در آن کار می‌کنند.

```
SELECT      SSN, DNAME
FROM        EMPLOYEE, DEPARTMENT;
```

۵-۸ عملگرهای مجموعه‌ای

عملگرهای اجتماع (Union)، اشتراک (Intersect) و تفاضل (Except) عملگرهای مجموعه‌ای SQL متناظر با عملگرهای $\cup$ و $\cap$ و $-$ در جبر رابطه‌ای می‌باشند. بدیهی است که رابطه‌هایی که این عملگرها بر روی آن‌ها اعمال می‌شود باید سازگار باشند، به این مفهوم که دارای تعداد یکسانی از صفات باشند.

دو پرس‌وجوی زیر را در نظر بگیرید.

- نام همه مشتریان یکه در بانک دارای حساب باشند.

```
Select      customer_name
From  depositor;
```

- نام مشتریانی که از بانک وام دریافت کرده‌اند.

```
Select     customer_name
From  borrower;
```

جدول حاصل از پرس‌وجوی اول را d و جدول حاصل از پرس‌وجوی دوم را b می‌نامیم و در بخش‌های بعد با آن‌ها کار می‌کنیم.

۸-۵-۱ عملگر اجتماع

می‌خواهیم نام مشتریانی را بدانیم که از بانک وام گرفته‌اند و یا این‌که در بانک حساب دارند. پرس‌وجوی زیر این پاسخ را به دست می‌آورد:

```
(Select    customer_name
From  depositor)
Union
(Select    customer_name
From borrower);
```

بر خلاف دستور select عملگر union به طور خودکار تکرارها را در جدول نتیجه حذف می‌کند. اگر بخواهیم همه تکرارها در نتیجه پرس‌وجو وجود داشته باشند (یعنی تکرارها حذف نشوند) از عملگر union all استفاده می‌کنیم.

تعداد تکرارها در جدول نتیجه برابر مجموع تعداد تکرارهای جداول d و b خواهد بود. مثلاً نام مشتری‌ای که دارای سه حساب بانکی است و وام دریافت کرده است، پنج بار در جدول نتیجه تکرار خواهد شد.

۸-۵-۲ عملگر اشتراک

اگر به جای union در پرس‌وجوی بخش قبل intersect قرار دهیم نام مشتریانی در حاصل پرس‌وجو قرار می‌گیرد که هم در بانک دارای حساب باشند و هم از بانک وام گرفته باشند.

```
(Select    distinct customer_name
From  depositor)
Intersect
(Select    distinct customer_name
From  borrower);
```

اشتراک نیز به طور اتوماتیک تکرارها را حذف می‌کند و برای باقی ماندن تکرارها باید از intersect all به جای intersect استفاده کنیم.

در اشتراک‌گیری بدون حذف تکرارها تعداد تکرارهای موجود در جدول برابر مینیمم تعداد تکرارها در d و b خواهد بود. مثلاً نام مشتریانی با سه حساب بانکی و دو وام دریافتی در جدول نتیجه اشتراک دو بار تکرار می‌شود.

۸-۵-۳ عملگر تفاضل

نام مشتریان صاحب حسابی که از بانک وام نگرفته‌اند از پرس‌وجوی زیر به دست می‌آید:

```
(Select    distinct customer_name
From depositor)
Except
(Select    distinct customer_name
From borrower);
```

مانند دو عملگر جتماع و اشتراک، در عملگر تفاضل هم برای عدم حذف تکرارها در جدول حاصل از except all استفاده می‌شود.

تعداد تکرارها در جدول نتیجه برابر تعداد تکرارها در جدول d منهای تعداد تکرارها در b خواهد بود. البته با این شرط که حاصل تفریق عددی مثبت باشد. به عنوان مثال نام مشتریانی که دارای سه حساب بانکی بوده و دو وام دریافتی دارند تنها یک بار در جدول حاصل خواهد آمد.

۸-۶ عملیات رشته‌ای

متداول‌ترین عمل مورد استفاده بر روی رشته‌ها، تطابق الگو با استفاده از عملگر like است. الگوها با استفاده از ۲ کاراکتر خاص تعریف می‌شوند.

- کاراکتر (%) که هر زیر رشته را مطابقت می‌دهد.
- کاراکتر (_) که برای تطابق کاراکترها به کار می‌رود.

الگوها به حروف کوچک و بزرگ حساس هستند به این مفهوم که حروف کوچک با حروف بزرگ تطابق نمی‌یابند.

برای روشن شدن بحث مثال‌های زیر را در نظر بگیرید.

- "perry%" با هر رشته‌ای که با زیررشته "perry" آغاز شود و مطابقت دارد.
- "%idge%" با همه رشته‌هایی که زیررشته "idge" در آن‌ها موجود باشد تطابق دارد مانند "Ridgeway" یا "perryridge".
- "---" (سه تا کاراکتر زیرخط) با هر رشته‌ای که دقیقاً شامل ۳ کاراکتر باشد مطابقت دارد.
- "%---" مطابق با هر رشته‌ای است که حداقل سه کاراکتر داشته باشد.

اگر در الگوی ما کاراکترهای ویژه یعنی % و _ وجود داشته باشند برای آن‌که بدانیم این‌ها جزئی از الگوی ما هستند و می‌خواهیم از آن‌ها مانند کاراکترهای معمولی استفاده کنیم، از یک کاراکتر مجزاکننده استفاده می‌کنیم. یعنی پس از بخش like، کاراکتر "\" به عنوان کاراکتر مجزا کننده استفاده شده است.

Like "ab \ % c % except " \" با همه رشته‌هایی مطابقت دارد که با "ab%c" آغاز شده باشند.

- نام مشتریانی مد نظر قرار دارد که در نام خیابان آن‌ها کلمه Main وجود داشته باشد:

```
Select      customer_name
From  customer
Where       customer_street like "% Main%";
```

- نام و نام خانوادگی کارمندانی که در آدرس آن‌ها عبارت Houston, Texas وجود داشته باشد.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE ADDRESS LIKE "%Houston, Texas%":
```

- نام و نام خانوادگی کارمندانی که در طول دهه ۱۹۵۰ متولد شده‌اند

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE BDATE LIKE "--5------";
```

عملگر not like هم در SQL وجود دارد که برای یافتن عدم تطابق به کار می‌رود.

۷-۸ مرتب‌سازی نمایش رکوردها

در SQL می‌توان کنترل ترتیب رکوردهای موجود در رابطه را در اختیار کاربر گذاشت. با کمک دستور order by در یک پرس‌وجو می‌توان چندتایی‌های موجود در جدول حاصل را به صورت مرتب چاپ کرد.

- لیست الفبایی مرتبی از نام مشتریانی که از شعبه perridge وام گرفته‌اند.

```
Select          distinct customer_name
From        borrower, Loan
Where       borrower.Loan_number= Loan. Loan_number and branch_name=
"perridge"
Order by    customer_name;
```

به صورت پیش‌فرض دستور order by رکورها را بر حسب فیلد مشخص شده به طور صعودی مرتب می‌کند.

برای مشخص کردن مرتب‌سازی می‌توان از کلمه asc برای مرتب‌سازی صعودی و از dec برای مرتب‌سازی نزولی استفاده کرد. هم‌چنین می‌توان مرتب‌سازی را بر روی چندین ویژگی اعمال کرد.

چون هزینه زمانی عمل مرتب‌سازی رکوردها بالاست بهتر است تنها در زمان نیاز از order by استفاده شود.

- همه وام‌های موجود در جدول Loan را به ترتیب نزولی میزان وام و اگر میزان چند وام یکسان بود آن وام‌ها را به ترتیب صعودی شماره وام نمایش دهید.

```
Select      *
From        Loan
Order by    amount dec, Loan_number asc;
```

۸-۸ زیر پرس‌وجوهای تودرتو

زیر پرس‌وجو، یک عبارت select_from_where است که درون یک پرس‌وجوی دیگر قرار می‌گیرد. یک کاربرد متداول زیرپرس‌وجوها، مشخص نمودن عضویت مجموعه‌ها، مقایسه مجموعه‌ها و کاردینالیتی مجموعه‌هاست.

۸-۸-۱ عضویت در مجموعه

برای تست عضویت در مجموعه از رابط in و تست عدم عضویت در یک مجموعه از رابط not in استفاده می‌شود. پیش از این پرس‌وجوی یافتن همه مشتریانی که هم در بانک دارای حساب هستند و هم از بانک وام دریافت کرده‌اند را با استفاده از عملگر اشتراک نوشتیم. در رویکرد دیگر می‌توانیم ابتدا همه صاحبان حساب را انتخاب کرده و به عنوان یک مجموعه در نظر بگیریم و بعد از میان آن‌ها تنها کسانی را که عضو مجموعه borrower هستند برگزینیم.

انتخاب همه صاحبان حساب با پرس‌وجوی زیر انجام می‌شود:

```
Select   customer_name
From     depositor;
```

حال برای یافتن مشتریانی از borrower که در جدول حاصل از پرس‌وجوی بالا قرار داشته باشند، پرس‌وجوی بالا را به عنوان زیرپرس‌وجو در یافتن مشتریان borrower قرار می‌دهیم.

```
Select      customer_name
From  borrower
Where customer_name in (select customer_name from depositor);
```

و این نشان می‌دهد که یک پرس‌وجو را به اشکال مختلف می‌توان در SQL نوشت.

- همه مشتریانی را بیابید که هم دارای حساب باشند و هم از شعبه perrydge وام دریافت کرده باشند.

```
Select      distinct customer_name
From  borrower, Loan
Where borrower.Loan_number = Loan.Loan_number
      And branch_name= "perrydge" and
      (branch_name, customer_name) in
            (Select    branch_name, customer_name
            From  depositor, account
            Where depositor.Account_number= account.Account_number)
```

برای روشن شدن کاربرد not in هم پرس‌وجوی زیر را در نظر می‌گیریم.

- همه مشتریانی را بیابید که از بانک وام گرفته‌اند اما در بانک حساب ندارند.

```
Select     distinct customer_name
From  borrower
Where customer_name not in (Select     customer_name
                  From depositor);
```

عملگرهای in و not in را می‌توانیم در ارتباط با مجموعه‌هایی که با اعضایشان هم مشخص شده‌اند به کار ببریم.

به عنوان مثال اگر بخواهیم مشتریانی را بیابیم که از بانک وام گرفته‌اند، ولی نام آن‌ها Smith یا Jones نباشد می‌توانیم پرس‌وجوی زیر را بنویسیم.

```
Select     distinct customer_name
From  borrower
Where customer_name not in ("smith", "jones");
```

شماره پرونده‌هایی را بیابید که یا آقای Smith مدیر دپارتمانی باشد که آن پروژه را کنترل می‌کند و یا خودش روی آن پروژه کار می‌کند.

```
SELECT      DISTINC PNUMBER
FROM        PROJECT
WHERE PNUMBER IN (SELECT    PNUMBER
                      FROM        PROJECT, DEPARTMENT, EMPLOYEE
                      WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME="
SMITH")
OR
PNUMBER IN (SELECT    PNO
            FROM      WORKS_ON, EMPLOYEE
            WHERE     ESSN = SSN AND LNAME = "SMITH");
```

۹-۸ توابع تجمعی

توابع تجمعی توابعی هستند که یک مجموعه از مقایر را به عنوان ورودی می‌گیرند و یک عدد را به عنوان خروجی باز می‌گردانند. در SQL پنج تابع تجمعی پیش ساخته وجود دارند: avg (میانگین)، min (مینیمم)، max (ماکزیمم)، sum (مجموع) و count (تعداد).

ورودی تابع sum و avg باید مجموعه‌ای از اعداد باشد اما ورودی سایر توابع می‌تواند مجموعه‌های غیرعددی هم باشد.

- میانگین موجودی حساب‌های شعبه perridge را بدهید.

```
Select      avg (balance)
From  account
Where branch_name = "perridge";
```

که نتیجه این پرس‌وجو رابطه‌ای با یک صفت است که تنها دارای یک سطر است.

- مجموع حقوق همه کارمندان دپارتمان Research و ماکزیمم و مینیمم و میانگین حقوق در این دپارتمان را محاسبه کنید.

```
SELECT      SUM (SALARY), MAX (SALARY), MIN (SALARY), AVG (SALARY)
FROM        EMPLOYEE, DEPARTMENT
WHERE DNO = DNUMBER AND DNAME = "RESEARCH";
```

- نام و نام خانوادگی تمامی کارمندانی که دو یا بیش‌تر وابسته دارند.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE  (SELECT    COUNT (*)
             FROM       DEPOSITOR
             WHERE      SSN = ESSN) >= 2;
```

گاهی لازم است توابع تجمعی را نه تنها بر روی یک مجموعه از مقادیر بلکه بر روی یک گروه از مجموعه‌ها (شامل چندین مجموعه مقادیر) اعمال نمائیم. این نیازمندی در زبان SQL با به کارگیری GROUP BY تحقق می‌یابد.

صفت یا صفاتی که در بخش group by مشخص می‌شوند، برای تشکیل گروه‌های مربوطه به کار می‌روند به این ترتیب که رکوردهایی که مقادیر همه صفات موجود در group by در آن‌ها یکسان باشند در یک گروه قرار می‌گیرند.

- میانگین موجودی هر شعبه را بیابید.

```
Select           branch_nam, avg (balance)
From        account
Group by    branch_name;
```

در محاسبه میانگین، نگه‌داشتن (حذف نکردن) تکرارها اهمیت اساسی دارد. زیرا در صورت حذف عناصر تکراری جواب متفاوتی به دست می‌آید.

اما مواردی هم وجود دارند که در آن‌ها مجبوریم پیش از انجام محاسبات تجمعی تکراری‌ها را حذف کنیم که در این موارد باید از کلمه کلیدی distinct درون توابع تجمعی استفاده کنیم.

- تعداد مشتریان صاحب حساب در هر شعبه را بدهید.

```
Select      branch_name, count (distinct customer_name)
From        depositor, account
Where       depositor.Account_number = account.Account_number
Group by    branch_name;
```

اگر ما تنها شعبه‌هایی را مورد بررسی قرار دهیم که میانگین موجودی آن‌ها بیش از ۱۲۰۰ دلار باشد، این شرط بر روی یک رکورد اعمال نمی‌شود بلکه باید بر روی گروه‌هایی که توسط بخش group by به وجود می‌آیند اعمال می‌گردد. برای چنین کاری در SQL از having استفاده می‌شود. شرط موجود در بخش having پش از ایجاد و شکل‌گیری گروه‌ها (توسط group by) بر روی آن‌ها اعمال می‌گردد. بنابراین می‌توان در بخش having هم از توابع تجمعی استفاده کرد. استفاده از توابع تجمعی در بخش where در پرس‌وجوهای SQL غلط است.

```
Select      branch_name, avg (balance)
From        account
Group by    branch_name
Having      avg (balance) > 120$;
```

از تابع count معمولاً برای شمارش تعداد چندگانه‌ها (رکوردهای) جدول استفاده می‌کنیم. نماد این تابع در SQL به صورت count (*) است.

- تعداد مشتریان بانک را بدهید.

```
Select      count (*)
From  customer;
```

در پرس‌وجوهای SQL نمی‌توان به همراه count (*) از distinct استفاده کرد.

اگر پرس‌وجویی هم شامل where و هم having باشد در ابتدا شرط موجود در بخش where بر روی رکوردها اعمال می‌شود و رکوردهایی را که شرط را برآورده می‌کنند انتخاب می‌کند. پس از where، group by قرار می‌گیرد که رکوردهای انتخاب شده توسط بخش where را گروه‌بندی می‌کند و در پایان having قرار می‌گیرد و شرط موجود در آن بر گروه‌های ایجاد شده اعمال می‌گردد و گروه‌هایی که شرط having را ارضا نکنند حذف می‌شوند. گروه‌های باقی مانده در بخش select مورد استفاده قرار می‌گیرند و دستور select با استفاده از آن‌ها نتیجه پرس‌وجو را تولید می‌کند.

- میانگین موجودی حساب‌های مشتریان ساکن Harrison را که حداقل دارای سه حساب هستند محاسبه نمایید.

```
Select      depositor. Customer_name, avg (balance)
From        depositor.Account, customer
Where       depositor.Account_number = account.Account_number and
            depositor.Customer_name = customer.Customer_name and
            customer_city = "Horrison"
Group by    depositor.Customer_name
```

```
Having      count (distinct depositor.Account_number) > 3;
```

- برای هر دپارتمان، شماره دپارتمان، تعداد کارمندان دپارتمان و میانگین حقوق آن‌ها را بیابید.

```
SELECT      DNO, COUNT (*), AVG (SALARY)
FROM        EMPLOYEE
GROUP BY    DNO;
```

- برای هر پروژه، شماره پرونده، نام پرونده و تعداد کارمندانی که روی پروژه کار می‌کنند را بیابید.

```
SELECT      DNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKES_ON
WHERE DNUMBER= PNO
GROUP BY    PNUMBER, PNAME;
```

- برای هر پروژه که بیش‌تر از ۲ کارمند روی آن کار می‌کند، شماره پرونده، نام پروژه و تعداد کارمندان که روی پروژه کار می‌کنند را بیابید.

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKES_ON
WHERE DNUMBER = PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT (*) > 2;
```

- برای هر پروژه، شماره پروژه، نام پروژه و تعداد کارمندانی از دپارتمان شماره ۵ که روی پروژه کار می‌کنند.

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKES_ON, EMPLOYEE
WHERE PNUMBER = PNO AND SSN = ESSN AND PNO =5
GROUP BY    PNUMBER, PNAME;
```

برای هر دپارتمان که بیش‌تر از ۵ کارمند دارد، شماره دپارتمان و تعداد کارمندانی که بیش از 40000 دریافت می‌کنند را بیابید.

```
SELECT      DNUMBER, COUNT (*)
FROM        DEPARTMENT, EMPLOYEE
WHERE DNUMBER = PNO AND SALARY > 40000 AND
                DNO IN (SELECT   DNO
                FROM        EMPLOYEE
                GROUP BY    DNO
                HAVING      COUNT (*) > 5)
GROUP BY    DNUMBER;
```

با توجه داشــت که توابع تجمعی در SQL قابلیت ترکیب شــدن ندارند، مثلاً برای یافتن بیش‌ترین میانگین نمی‌توان از max (avg (...)) استفاده نمود.

۸-۱۰ مقادیر NULL

گفتیم که در SQL به جای مقدار صــفاتی که به هر دلیل فاقد مقدار هســتند، NULL قرار می‌گیرد. برای آن‌که مشــخص کنیم مقدار صفتی NULL است یا خیر می‌توانیم در شرط خود از کلمه کلیدی IS NULL استفاده کنیم.

- وام‌هایی را بیابید که میزان آن‌ها مشخص نیست.

```
Select      Loan_number
From  Loan
Where amount  is  NULL;
```

متناظر با is null، is not null نیز برای تست null نبودن یک مقدار وجود دارد.

استفاده از null در عبارات محاسباتی باعث ایجاد مشکل می‌شود، چون اگر هر یک از مقادیر عبارات محاسباتی null باشد حاصل کل محاسبه null خواهد بود و هر مقایسه‌ای که شامل null باشد مقدار false برمی‌گرداند. علاوه بر این وجود مقادیر null باعث پیچیده شدن عملیات عملگرهای تجمعی می‌شود.

به طور کلی توابع تجمعی بر اســاس قانون زیر با مقادیر null برخورد می‌کنند. همه توابع تجمعی به جز count (*) از همه مقادیر null موجود در مجموعه ورودی چشم‌پوشی می‌کنند.

۸-۱۱ تهی بودن رابطه

در SQL با کمک مکانیزم‌هایی می‌توان مشخص کرد که پس از اجرای یک پرس‌وجو آیا رکوردی در جدول حاصل وجود دارد یا نه. اگر حاصــل یک زیرپرس‌وجو تهی نباشــد، شــبه دســتور Exit به ازای آن مقدار true را برمی‌گرداند. مثلاً اگر بخواهیم پرس‌وجوی "یافتن مشتریانی که هم دارای حساب‌اند و هم وام دارند" را با استفاده از exit دوباره بنویسیم این‌گونه خواهد بود.

```
Select      customer_name
From  borrower
Where exit  (Select *
        From depositor
        Where      depositor.Customer_name= borrower.Customer_name);
```

Not exit هم تهی بودن رابطه را بررســی می‌کند. هم‌چنین می‌توان با کمک not exit زیرمجموعه بودن دو مجموعه را بررسی نمود. البته در مورد رابطه‌ها به جای لفظ زیرمجموعه بودن از واژه شمول استفاده می‌کنیم.

در مورد درخواست "یافتن مشتریانی که در همه شعبه‌های واقع در Brooklyn حساب دارند" ابتدا باید ببینیم به ازای هر مشتری آیا مجموعه همه شعبه‌هایی که این مشتری در آن‌ها حساب دارد شامل مجموعه همه شعبه‌های مجموعه همه شعبه‌های موجود Brooklyn هست یا نه.

```
Select     distinct s.customer_name
From  depositor AS s
Where not exit ((Select     branch_name
             From branch
             Where     branch_city y= "Brooklyn")
            Exept
             (Select   R.branch_name
             From depositor as T.account as R
             Where T.account_number= R.account_number
             And s.customer_name= T.customer_name));
```

زیر پرس‌وجوی آخر این پرس‌وجو، همه شعبه‌هایی را پیدا می‌کند که مشتری در آن‌ها حساب دارد (مجموعه A) و زیر پرس‌وجوی پیش از آن همه شعبه‌های واقع در Brooklyn را می‌یابد (مجموعه B).

تهی بودن B-A توسط non exit بررسی می‌شود و مشتریانی که به ازای آن‌ها B-A خالی باشد توسط select اصلی انتخاب می‌شوند.

اگر متغیر چندگانه‌ای در پرس‌وجوی اصلی تعریف شود، در همه پرس‌وجوهای آن پرس‌وجو هم تعریف شده و قابل استفاده می‌باشد. اگر یک متغیر چندگانه‌ای را هم در یک زیر پرس‌وجو و هم در پرس‌وجوی اصلی تعریف کنیم، هنگام استفاده از آن متغیر در زیرپرس‌وجوی مزبور تعریف انجام شده در زیرپرس‌وجو مورد نظر و ملاک قرار می‌گیرد.

۱۲-۸ نمایش (view)

نمایش در حقیقت یک جدول مجازی است که چندتایی ندارد اما می‌توان روی آن پرس‌وجو انجام داد. در SQL دستور create view برای تعریف یک نمایش به کار می‌رود. برای این کار لازم است نام نمایش و پرس‌وجویی را که نمایش را محاسبه می‌کند مشخص نماییم.

ساختار دستور create view به‌صورت زیر است.

```
Create     view v as < query expression>
```

در تعریف فوق V نام نمایش و <query expression> یک پرس‌وجوی صحیح می‌باشد.

نمایشی ایجاد کنید که شامل نام و شعبه مشتریانی باشد که از بانک وام گرفته‌اند و یا در بانک حساب دارند (یا هر دو).

```
Create     view all_Customer as
```

```
(Select    branch_name, customer_name
From  depositor, account
Where depositor.Account_number= account.Account_number)
Union
(Select    branch_name, customer_name
From  borrower, loan
Where borrower.Loan_number = loan.Loan_number);
```

صفات یک نمایش را نیز می‌توانیم صریحاً بیان و نام‌گذاری کنیم.

- نمایشی ایجاد کنید شامل نام هر شعبه و کل میزان وام پرداخت‌شده توسط آن شعبه:

```
Create     view branch_total_loan (branch_name, total_loan) as
Select          branch_name, sum (amount)
From       loan
Group by   branch_name;
```

از نمایش‌ها می‌توان مشابه جداول استفاده کرد و هر جا که در یک پرس‌وجو بتوان نام یک جدول را آورد می‌توان نام یک نمایش را نیز قرار داد.

وقتی یک نمایش ایجاد می‌شود، در پایگاه داده باقی می‌ماند مگر آن‌که آن را با استفاده از دستور drop view view_name از بین ببریم. البته در نگارش‌های جدیدتر SQL نمایش‌هایی وجود دارند که در پایگاه داده باقی نمی‌مانند.

۸-۱۳ تغییر در پایگاه داده

تغییرات در پایگاه داده توسط دستورات DML از زبان SQL انجام می‌شود. زبان SQL برای این منظور دستورات DELETE، INSERT و UPDATE را ارائه نموده است.

۸-۱۳-۱ حذف

در عملیات درخواست حذف در یک پایگاه داده، تنها حذف یک رکورد کامل از پایگاه داده مقدور است و نمی‌توان تعداد از صفات یک رکورد را حذف کرد.

فرم دستور حذف در SQL به‌صورت زیر است.

```
Delete     from r
Where p;
```

که r نام جدولی است که حذف از آن انجام می‌شود و p شرط حذف یک رکورد را مشخص می‌کند.

دستور delete ابتدا همه چندگانه‌های t از جدول r را که در شرط p صدق می‌کنند می‌یابد و بعد آن‌ها را از r حذف می‌کند. اگر بخش where در پرس‌وجو نباشد همه چندتایی‌ها از جدول حذف می‌شوند.

دستور delete تنها بر روی یک جدول عمل می‌کند و اگر بخواهیم داده‌هایی را از جداول مختلف حذف کنیم بایستی برای هر یک از جداول دستور delete را جداگانه به کار ببریم.

- همه حساب‌های شعبه perryridge را حذف کنید.

```
Delete      from loan
Where branch_name = "perrylidge";
```

- اطلاعات همه وام‌هایی که میزان آن‌ها بین ۱۳۰۰ تا ۱۵۰۰ دلار بوده حذف شود.

```
Delete      from loan
Where amount between 1300 and 1500;
```

- حساب‌هایی که در شعبه‌های شهر needham وجود دارند حذف شوند.

```
Delete      from loan
Where branch_name in (Select branch_name
                      From branch
                      Where branch_city = "needham");
```

- کارمندانی با نام خانوادگی brown را حذف کنید.

```
DELETE      FROM EMPLOYEE
WHERE LNAME=" Brown";
```

- تمام کارمندانی را که در دپارتمان Research کار می‌کنند، حذف کنید.

```
DELETE FROM EMPLOYEE
Where              DNO IN (SELECT DNumber
                   From Department
                   Where DName = ="Research");
```

۸-۱۳-۲ درج

برای درج داده به یک رابطه می‌توان به دو صورت عمل کرد: ۱- یک رکورد را با اعضا (مقادیر صفاتش) مشخص نموده تا به جدول اضافه شود. ۲- پرس‌وجویی نوشت که جدول حاصل از آن به مجموعه رکوردهای جدول مورد نظر افزوده شود.

مقادیر ضفات رکورد (رکوردهای) اضافه شده به جدول بایستی مجاز باشند، یعنی در دامنه مقادیر هر ویژگی وجود داشته باشند. هم‌چنین باید تعداد و نوع صفات رکورد اضافه شده با تعداد و نوع صفات جدول مورد نظر مطابقت داشته باشد.

مثلاً اگر قرار باشد حسابی با شماره A-9732 در شعبه perryridge با موجودی ۱۲۰۰ دلار افتتاح شود از دستور INSERT به صورت زیر استفاده می‌کنیم.

```
Insert into      account
Values           ("perriridge", "A-9732", 1200);
```

در ضمن مقادیر متناظر باید با همان ترتیب درون رکورد قرار گیرند که صفات متناظرشان در شمای جدول مورد نظر قرار دارد.

می‌توانیم رکوردهایی را به جدول بیفزاییم که تنها تعدادی از صفات آن‌ها دارای مقدار باشد.

```
Insert into      account
Values           (null, "A_A01, 1200);
```

این نشان می‌دهد که حسابی به شماره A_A01 و با موجودی 1200 دلار به جدول account اضافه می‌شود. مقدار صفت branch_name برابر null بوده، لذا شعبه‌ای که این حساب در آن افتتاح شده نامشخص است.

۸-۱۳-۳ به‌روز رسانی

از دستور update برای این منظور استفاده می‌کنیم که در برخی از موارد ممکن است بخواهیم بدون تغییر همه مقادیر یک رکورد، تنها مقادیر برخی از ویژگی‌های رکورد را تغییر دهیم.

- به عنوان پرداخت سود سالانه، میزان موجودی هر حساب را ۵% افزایش دهید.

```
Update      account
Set         balance= balance* 1.05;
```

- به افرادی که موجودی آن‌ها بیش از ۱۰۰۰ دلار می‌باشد، ۶% سود سالانه تعلق می‌گیرد و بقیه افراد مشمول همان ۵% سود شوند.

```
Update      account
Set         balance= balance * 1.06
Where       balance> 1000;
Update      account
Set         balance= balance * 1.05
Where       balance<= 1000;
```

باید توجه داشت که ترتیب نوشت عملیات بالا اهمیت دارد، زیرا اگر پرس‌وجوی دوم را اول انجام دهیم ممکن است برخی از حساب‌ها که موجودی آن‌ها کم‌تر از ۱۰۰۰ دلار می‌باشد با افزایش سود سالانه ۵% مقدارشان به بیش از ۱۰۰۰ دلار برسد و بعد با پرس‌وجوی بعد ۶% دیگر نیز به موجودی آن‌ها افزوده شود.

بخش where در دستور update همانند بخش where در دستور select می‌باشد. به‌کارگیری دستور select در where هم مجاز است. همانند دستورات delete و insert در بخش where از دستور update هم می‌توان یک دستور Select قرار داد که رکوردهایی را از جدولی که update می‌شوند انتخاب نماید.

۸-۱۴ جمع‌بندی

با توجه به توضیحات ارائه شده در مورد زبان SQL نکات زیر را می‌توان به‌عنوان جمع‌بندی ارائه نمود.

- SQL یک زبان پرس‌وجوی ساخت‌یافته است.
- یک بخش از SQL، DDL (زبان تعریف داده) می‌باشد که SQL با استفاده از آن می‌تواند انواع دامنه و شماهای داده را تعریف کند.
- DML که زبان دستکاری داده است، بخش دیگری از SQL می‌باشد که توسط نوع دستوراتی مانند delete, update, insert داده‌ها را عوض می‌کند. هم‌چنین قوانین جامعیت هم در این زبان رعایت می‌شود و اجازه دسترسی هر کاربر هم تعیین می‌شود.
- تابع extract جهت به دست آوردن فیلدهای خاص از نوع داده‌های موجود در SQL استفاده می‌شود. مثلاً اگر d_1 نام یک داده از نوع تاریخ (Date) باشد و آن‌گاه دستور extract (year from d_1) تنها سال را از این نوع داده برمی‌گرداند.
- چنانچه کلید خارجی (FK) از جدول مرجع حذف شود و یا تغییر دیگری کند با استفاده از دو عبارت [on delete cascade] و یا [on update cascade] تغییرات به جداول مراجع کننده سرایت می‌کنند.
- بخش group by جزو آخرین بخش‌های یک دستور است و فقط order by می‌تواند بعد از آن بیاید.
- ویژگی که گروه‌بندی روی آن انجام می‌شود حتماً باید در خروجی (دستور select) بیاید.
- اگر بخواهیم بر روی توابع تجمعی شرط بگذاریم با دستور having این کار انجام می‌شود که شبیه where عمل می‌کند.
- الحاق رشته در SQL توسط عملگر || انجام می‌شود.
- تمام توابع تجمعی به جز (*) count، چندتایی‌هایی با مقادیر Null را نادیده می‌گیرند.
- عملگرهای all, some, any, in روی عدد کار می‌کنند و عملگرهای Exists و Not exists روی سطرها عمل می‌کنند.
- برای پیاده‌سازی تقسیم در SQL مستقیماً دستوری وجود ندارد. این عملگر زمانی به کار می‌رود که قید همه در پرس‌وجو به کار نرفته باشد. در SQL این عملگر را می‌توان با پرس‌وجوهای تودرتو پیاده‌سازی کرد.
- در SQL سه نوع جدول وجود دارد:
  - جداول اصلی (base table) که همان جداولی هستند که می‌توان به آن‌ها دسترسی داشت و یا آن‌ها را تغییر داد و با دستور create table ایجاد می‌شوند.
  - جداول میانی (inter mediate table) که نه قابل دسترسی‌اند و نه قابل تغییر و در طول انجام عملیات توسط خود سیستم ساخته شده و سپس هم از بین می‌روند.
  - جداول مجازی (views) که اصلاً وجود خارجی و فیزیکی هم ندارند و هدف از ایجاد آن‌ها ایجاد اطلاعات خلاصه از اطلاعات موجود و محدود کردن نمایش کاربران است.
- جداول مجازی را هم مانند جداول حقیقی می‌توان در پرس‌وجوهای مختلف به کاربرد و هم آن‌ها را با دستور Drop view از بین برد.

۹ وابستگی تابعی و نرمال‌سازی

در حالت کلی هدف از طراحی پایگاه داده رابطه‌ای، ساخت مجموعه‌ای از شماهاست که امکان ذخیره داده‌ها بدون افزونگی‌های غیر ضروری را فراهم نماید. هم‌چنین بر اساس آن طراحی بتوانیم داده‌های ذخیره شده را به آسانی بازیابی نماییم. یک رویکرد در طراحی صحیح یک پایگاه داده با این مشخصات این است که شماهای طراحی شده در فرم نرمال مناسب قرار داشته باشند.

یکی از پارامترها برای مقایسه دو مدل رابطه‌ای این است که هر کدام که فضای کم‌تری بگیرند بهترند.

توصیه‌های طراحی بهینه برای کیفیت مدل رابطه‌ای عبارتند از:

۱) معنای صفت را رعایت کنید و صفاتی را در جدول نگذارید که ربطی به آن نداشته باشد.
۲) وجود اطلاعات تکراری حداقل شود.
۳) کاهش مقادیر Null چون هم حافظه زیادی اشغال می‌کنند (به اندازه یک ویژگی حافظه می‌گیرد و ابهام هم ایجاد می‌کند زیرا حالت‌هایی هم که ممکن است Null اتفاق بیفتد ۱) قابل اعمال نبودن ۲) نامشخص بودن (مقدار نداشتن) ۳) در دسترس نبودن (نداشتن مقدار) می‌باشد) همچنین مقادیر NULL در توابع تجمعی مشکل ایجاد می‌کنند.
۴) جلوگیری از امکان تولید چندتایی‌های غلط که ممکن است بر اساس یک join به وجود آیند.

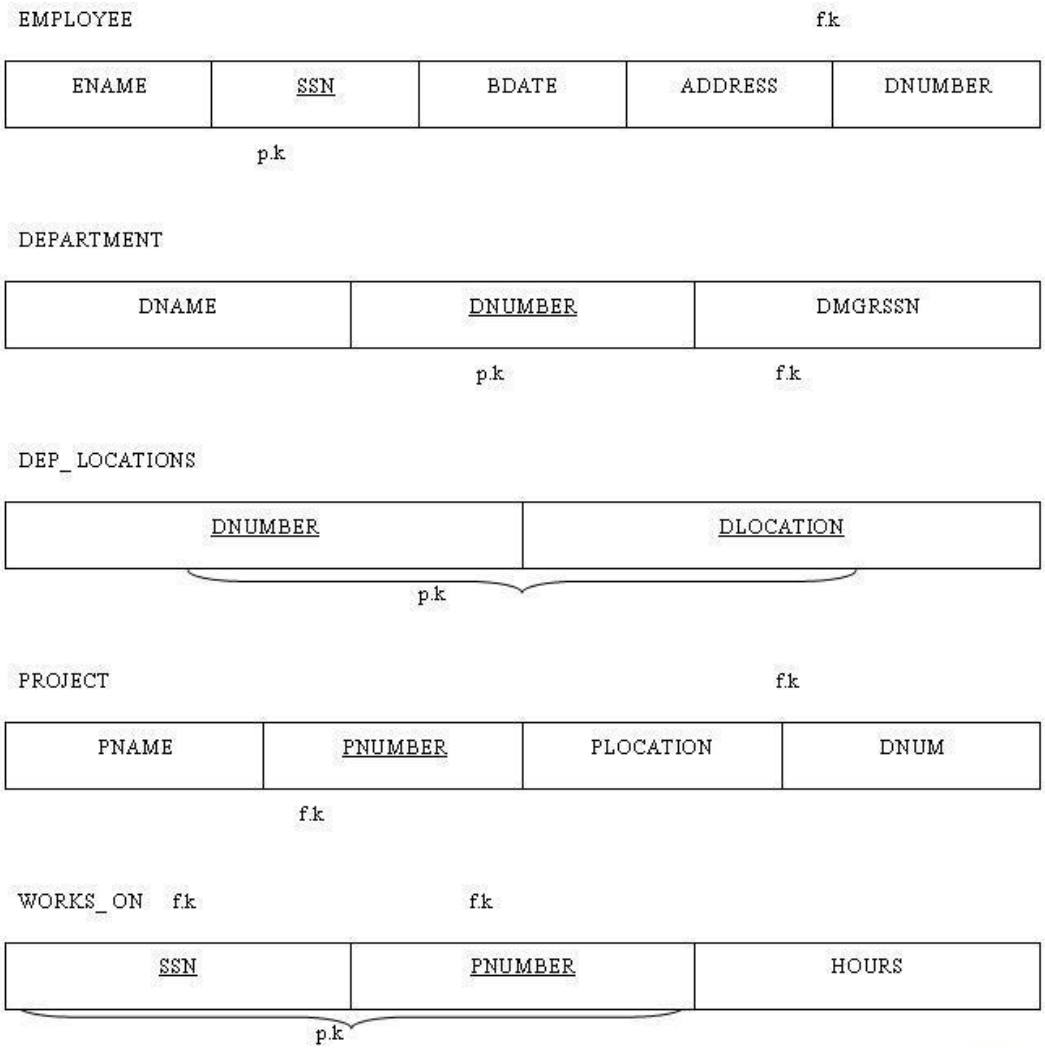
جداولی که به وسیله پرس‌وجو می‌سازیم ممکن است دارای اشکالاتی مانند افزونگی (Redundancy)، وابستگی و نرمال‌سازی باشند. افزونگی یعنی تکرار بی‌رویه داده‌ها که مشکلات زیر را به وجود می‌آورد:

- Redundancy Storage: نیاز به حافظه بیشتر
- Update Anomaly: اگر یک کپی از جدول داشته باشیم برای update کردن بایستی یک بار این کار را بر روی جدول اصلی و یک بار نیز بر روی کپی آن انجام دهیم.
- Insertion/ Deletion Anomaly: برای اضافه یا حذف کردن داده‌ای باید این کار را هم روی جدول اصلی و هم روی جدول کپی انجام دهیم.

که این مشکلات ممکن است به دلیل ادغام نادرست جداول به وجود آمده باشد. هم‌چنین با ادغام جداول مقادیر تهی در جدول زیاد می‌شود. که مقادیر تهی حجم زیادی اشغال می‌کنند و مشکلات دیگری نیز به همراه دارند.

ایده اصلی نرمال‌سازی رابطه‌ها بر مبنای رفع ناهنجاری‌ها در رابطه‌هاست.

در این فصل یک نسخه جدیدی از شمای پایگاه داده رابطه‌ای COMPANY ارائه شده که به صورت زیر می‌باشد.



۹-۱ وابستگی تابعی

نوع خاصی از قیود به نام وابستگی تابعی (FD[۱۹]) وجود دارد و مفهوم آن در واقع تعمیم‌یافته مفهوم کلیدهاست و محدودیتی است بین دو مجموعه از صفات یک رابطه در یک پایگاه داده رابطه‌ای. وابستگی تابعی در واقع مفهوم سوپرکلید را تعمیم و گسترش می‌دهد.

رابطه $R(A_1, A_2, \ldots, A_n)$ مفروض است و $r(R)$ مجموعه‌ای از چندتایی‌های رابطه R است. دو مجموعه x و y را به عنوان دو زیرمجموعه از صفات R در نظر بگیرید. $(x \subseteq R, y \subseteq R)$ وابستگی تابعی $x \rightarrow y$ (می‌خوانیم x، determine می‌کند

[۱۹] Functional Dependency

y را، یا x تعیین می‌کند y را) بر روی شـمای رابطه R برقرار اسـت، اگر برای هر حالت r (R)، به ازای هر دو چندتایی t_1 و t_2 رابطه زیر برقرار باشد.

$t_1 (x) = t_2 (x) \Rightarrow t_1 (y) = t_2 (y)$

با اسـتفاده از نمادهای وابسـتگی تابعی می‌توان سـوپرکلید را به این صـورت تعریف نمود که مجموعه k سوپرکلید رابطه R است اگر $k \rightarrow R$ و می‌خوانیم k ابرکلید است اگر R وابسته تابعی به k باشد و بدیهی است که رابطه زیر برقرار است:

$t_1[k]= t_2[k] \Rightarrow t_1[R]= t_2[R]$

یعنی اگر ۲ چندتایی t_1 و t_2 بر روی مجموعه صـفات k دارای مقادیر یکسـانی باشـند در آن صورت با توجه به سوپرکلید بودن k و وابستگی تابعی R به k، این دو چندتایی در R هم نباید متمایز باشند.

شمای زیر را در نظر بگیرید:

Loan_info_schema = (branch_name, loan_number, customer_name, amount)

مجموعه وابستگی‌های تابعی که انتظار می‌رود بر روی این شما برقرار باشند به شرح زیرند.

{loan_number} → {amount}

{loan_number} → {branch_name}

اما نباید انتظار داشته باشیم وابستگی تابعی زیر برقرار باشد.

{loan_number} → {customer_name}

زیرا در حالت کلی هر وام لزوماً به یک نفر تعلق نمی‌گیرد و ممکن است ۲ نفر با هم گیرنده وام باشند.

با در نظر گرفتن شمای رابطه EMPLOYEE از مثال COMPANY وابستگی‌های تابعی زیر برقرارند.

{SSN} → {All attributes}

{FName, Lname} → {Fname}

یکی از کاربردهای وابسـتگی تابعی در تشـخیص این موضـوع این اسـت که آیا یک جدول تحت مجموعه‌ای از وابسـتگی‌های تـابعی مجزا اسـت یـا خیر. اگر جـدول، تحـت مجموعـه وابسـتگی‌هـای تـابعی F مجاز باشـد گوییم که آن جدول مجموعه وابستگی‌های F را ارضا می‌کند.

جدول r در شکل زیر در نظر گرفته و وابستگی‌های تابعی را که توسط این جدول ارضا می‌شوند از آن استخراج کنید.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2

a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

مشاهده می‌شود که $A \rightarrow C$ برقرار است. دو رکورد در جدول وجود دارند که مقدار ویژگی A برای آن‌ها برابر a_1 است اگر این دو رکورد را t_1 و t_2 بنامیم می‌بینیم که $t_1(A) = t_2(A) = a_1$ و از طرفی مقدار صفت C برای هر دو رکورد فوق c_1 است، یعنی $t_1[c] = t2[c] = c_1$ و به همین ترتیب دو رکورد در جدول موجودند که مقدار ویژگی A برای آن‌ها a_2 است و مقدار ویژگی C در هر رکورد مزبور c_2 است. بنابراین به ازای هر دو رکورد دلخواهی که مقدار ویژگی A در آن‌ها یکسان باشد، مقدار ویژگی c نیز یکسان است یعنی:

$$\forall\ t_1, t_2 \in r\ t_1(A) = t_2(A) \Rightarrow t_1(C) = t_2(C)$$

به همین ترتیب وابستگی‌های تابعی دیگری را نیز می‌توان مشاهده کرد که توسط جدول r ارضا می‌شوند مانند $B \rightarrow C$ و $AB \rightarrow D$. برخی از وابستگی‌های تابعی بدیهی هستند مانند $A \rightarrow A$ زیرا توسط هر جدولی که شمای آن شامل ویژگی A باشد و $AB \rightarrow A$. بنابراین موارد زیر را داریم.

- در حالت کلی اگر $y \supseteq x$ آن‌گاه همواره $x \rightarrow y$. این وابستگی تابعی را بدیهی (tririal FD) می‌نامیم.
- اگر $x \rightarrow y$ لزوماً نداریم $y \rightarrow x$ (رابطه وابستگی تقارنی نیست.)

رابطه $x \rightarrow y$ را وابستگی تابعی کامل (FDD[۲۰]) گوییم، اگر y به هیچ زیرمجموعه محض از x وابسته نباشد.

FDD را می‌توان این گونه هم تعریف کرد:

صفت خاصه y از رابطه R با صفت خاصه x از آن FDD دارد هرگاه x با y، FDD داشته باشد اما با هیچ کدام از اجزای تشکیل دهنده آن FD نداشته باشد.

ممکن است برخی از وابستگی‌های تابعی را از وابستگی‌های تابعی دیگر نتیجه گرفت.

فرض کنید داریم:

$R = (S, T, U, V, W)$

$F = \{S \rightarrow T, V \rightarrow SW, T \rightarrow U\}$

آن‌گاه منطقاً می‌توان وابستگی‌های تابعی زیر را نتیجه گرفت.

$S \rightarrow U \qquad (S \rightarrow T, T \rightarrow U)$

---

[۲۰] Full Functional Dependency

$V \rightarrow S$

$V \rightarrow W$

$V \rightarrow T$ $(V \rightarrow S, S \rightarrow T)$

$V \rightarrow U$ $(V \rightarrow S, S \rightarrow U)$

و می‌توان نتیجه گرفت V کلید کاندید R است زیرا همه صفات R را نتیجه می‌دهد. باید توجه داشت که مجموعه‌ای از صفات کلید کاندید است که تمام صفات رابطه توسط آن مجموعه تعیین می‌شود و آن مجموعه نیز حداقل است. یک جدول ممکن است بیش از یک کلید کاندید داشته باشد که یکی از آن‌ها کلید اصلی خواهد بود.

۹-۲ بستار وابستگی تابعی

فرض کنید F یک مجموعه از وابستگی‌های تابعی باشد. مجموعه تمام FDهایی که از F قابل استنتاج هستند یا مجموعه‌ای از همه وابستگی‌های تابعی که منطقاً توسط F ایجاب می‌شوند، را بستار (پوششی) F می‌نامند و با F^+ نشان می‌دهند. مجموعه F^+ مستقیماً از تعریف وابستگی‌های تابعی قابل محاسبه است. اگر F خیلی بزرگ باشد این فرآیند بسیار طولانی و مشکل خواهد شد. مجموعه‌ای از قواعد استنتاجی (IR[۲۱]) داریم که ناظر بر مفهوم FD هستند و به کمک این قواعد می‌توان به تمام وابستگی‌های منتج دست یافت.

فرض کنید α مجموعه‌ای از صفات باشد، مجموعه همه صفات را که بر اساس مجموعه F از وابستگی‌های تابعی به صورت تابعی از α مشخص می‌شوند، بستار α تحت F می‌نامیم و آن را با α^+ نمایش می‌دهیم.

الگوریتم محاسبه α^+ به صورت شبه کد پاسکال:

ورودی‌های الگوریتم: مجموعه وابستگی‌های تابعی F و مجموعه صفات α:

خروجی در متغیر RESULT ذخیره می‌شود.

```
Result: = α
While (change to result) do
      For each function dependency D→Y in F do
      Begin
            If B ⊆ result then result: = result ∪ y;
      End;
```

مجموعه وابستگی‌های تابعی F را که بر روی F = (A, B, C, G, H, I) برقرار است در نظر بگیرید.

---

[۲۱] Inference Rule

$F= (A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

برای روشـن شـدن کار الگوریتم سـعی می‌کنیم با اسـتفاده از آن (AG^+) را با وابسـتگی‌های تابعی که داریم محاسـبه کنیم. منظور از AG مجموعه $\alpha= \{A,B\}$ است.

ابتدا مقدار result را برابر AG قرار می‌دهیم. بعد از اجرای نخستین دور حلقه داریم:

- وجود قانون $A \rightarrow B$ باعث می‌شود که به B به result اضافه شود.

$A \rightarrow B \in F, A \subseteq result \Rightarrow result: = result \cup B$

- وجود $A \rightarrow C$ سبب می‌شود که هم به result اضافه شد. تا این مرحله result به صورت ABCG در آمده است.
- وجود $CG \rightarrow H$ باعث می‌شود که result به صورت ABCGH درآید.
- وجود $CG \rightarrow I$ باعث می‌شود که result به صورت ABCGHI در آید.
- در اجرای دور دوم حلقه while هیچ ویژگی جدیدی به result اضافه نمی‌شود و الگوریتم خاتمه می‌یابد.

قوانین استنتاجی زیر برای توسعه مجموعه FDها از یک مجموعه قابل استفاده است.

- قانون بازتابی: هر ویژگی به خودش وابسـتگی تابعی دارد و هر مجموعه از ویژگی‌ها نیز به همان مجموعه وابسـته تابعی است.

IR1: if $x \supseteq y$ then $x \rightarrow y$

- قانون جذب (افزایش) Augmentation rule:

IR2: $\{x \rightarrow y\} \Rightarrow xz \rightarrow yz$ OR $xz \rightarrow y$

- قانون تعدی (تراگذاری) transitive rule:

IR3: $\{x \rightarrow y, y \rightarrow z\} \Rightarrow x \rightarrow z$

این قوانین مانع هستند یعنی هیچ وابستگی تابعی نادرستی تولید نمی‌کنند و کامل (جامع) هستند یعنی به ازای هر مجموعه F از وابسـتگی‌های تابعی ما را قادر می‌سـازند که همه وابستگی‌های موجود در F^+ را تولید نماییم. این قوانین جامع و کامل به نام ارائه‌دهنده این قوانین آرمسترانگ نامیده می‌شوند.

- قانون تجزیه پذیری decomposition rule

IR4: $\{x \rightarrow yz\} \Rightarrow x \rightarrow y$

- قانون اجتماع Union or additive rule:

IR5: $\{x \rightarrow y, x \rightarrow z\} \Rightarrow x \rightarrow yz$

- قانون شبه تراگذاری Pseudotrasitivity rule:

IR6: $\{x \rightarrow y, wy \rightarrow z\} \Rightarrow xw \rightarrow z$

دو مجموعه وابستگی تابعی F_1 و F_2 را معادل نامند اگر مجموعه پوششی آن‌ها برابر باشد $F^+_1 = F^+_2$

با اعمال قوانین فوق ممکن است وابستگی‌های تکراری و اضافی به وجود آید که باید مجموعه وابستگی را بهینه کنیم.

با کمک ۳ قاعده زیر می‌توان یک مجموعه وابسته را به مجموعه بهینه معادل آن تبدیل کرد:

- سمت راست هر وابستگی فقط یک صفت باشد.
- هر صفتی که F^+ را تغییر نمی‌دهد از سمت چپ حذف شود.
- وابستگی‌های تکراری و اضافی حذف شوند.

با توجه به رابطه R مجموعه وابستگی پوششی بهینه را بیابید.

$R= (U, V, W, X, Y, 2)$

$F= \{U \rightarrow XY, X \rightarrow Y, XY \rightarrow ZV\}$

$F^+= \{U \rightarrow XY, X \rightarrow Y, XY \rightarrow ZV, U \rightarrow ZV\}$

اعمال قاعده اول روی وابستگی اول:

$U \rightarrow XY \Rightarrow U \rightarrow X, U \rightarrow Y$

$F^+= \{ U \rightarrow X, U \rightarrow Y, X \rightarrow Y, XY \rightarrow ZV, U \rightarrow ZV\}$

اعمال قاعده دوم روی $XY \rightarrow ZV$

$XY \rightarrow ZV \Rightarrow X \rightarrow ZV$

اعمال قاعده اول روی سایر وابستگی‌ها:

$U \rightarrow ZV \Rightarrow U \rightarrow Z, U \rightarrow V$

$X \rightarrow ZV \Rightarrow X \rightarrow Z, X \rightarrow V$

$F_{OPT} = \{U \rightarrow X, U \rightarrow Y, U \rightarrow Z, U \rightarrow V, X \rightarrow Y, X \rightarrow Z, X \rightarrow V\}$

اگر قوانین استنتاجی مذکور را بر روی مجموعه وابستگی‌های تابعی F که بر روی $R \cong \{A, B, C, G, H, I\}$ برقرار است اعمال نماییم.

$F= \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

در این صورت وابستگی‌های تابعی زیر عضو F^+ خواهند بود.

- $A \rightarrow B, B \rightarrow H \Rightarrow A \rightarrow H$
- $CG \rightarrow H, CB \rightarrow I \Rightarrow CG \rightarrow HI$
- $A \rightarrow C, CG \rightarrow I \Rightarrow AG \rightarrow I$

دترمینان: دترمینان به یک وابستگی تابعی به صفت و یا گروهی از صفات گفته می‌شود که طرف چپ یک وابستگی هستند. مثلاً در $x \rightarrow y$، x دترمینان می‌باشد.

بستار صفت A را با A^+ نشان داده و می‌توان آن را با محاسبه F^+ و انتخاب آن FDهایی که در آن A دترمینان است بدست آورد.

الگوریتم محاسبه مجموعه صفات که به صورت تابعی از attr مشخص می‌شوند به عبارت دیگر این الگوریتم زیرمجموعه تمام صفت‌های وابسته به atr را می‌دهد.

```
1.  attr+= attr
2.  repeat
3.    old attr+= attr
4.    for each FD such as x→y in F do
5.       if x⊆attr+ then attr+= attr+ ∪y
6.  until (attr+= old attr+)
```

با استفاده از این الگوریتم می‌توان کلیدهای کاندید را به دست آورد.

- هر کلید کاندیدی سوپر کلید است ولی هر سوپر کلیدی کلید کاندید نیست.
- کلید کاندید مجموعه صفت‌هایی هستند که در طرف چپ پیکان‌ها می‌آیند.
- ممکن است چندین کلید کاندید داشته باشیم، کلیدهای کاندید ممکن است در یک یا چند صفت مشترک باشند.

همیشه به سمت چپ FD می‌توان یک صفت اضافه کرد (قاعده چپ افزایی) اما عکس این قاعده صادق نیست.

یک صفت از وابستگی تابعی اضافی (extraneous) نامیده می‌شود و به صورت رسمی این طور تعریف می‌شود: اگر مجموعه وابستگی‌های تابعی F و وابستگی تابعی $x \rightarrow y$ از مجموعه F مفروض باشد:

صفت A از مجموعه X اضافی است اگر $A \in X$ و F منطقاً وابستگی زیر را ایجاب کند:

$$(F - \{X - Y\} \rightarrow \{(x - A) \rightarrow Y\}$$

و صفت A از مجموعه Y اضافی است اگر $A \in Y$ و F منطقاً وابستگی زیر را ایجاب می‌کند:

$$(F - \{X \rightarrow Y\}) \rightarrow \{X \rightarrow (Y - A)\}$$

نکته: با فرض وجود یک مجموعه از FDها به نام F همیشه می‌توان حداقل یک مجموعه دیگر پیدا کرد که کاهش‌ناپذیر باشد. این مجموعه ۳ خاصیت دارد:

۱) سمت راست هر FD صفت ساده باشد.

۲) سمت چپ هر FD کاهش‌ناپذیر باشد (FD کامل باشد)

وابستگی تابعی مانند $A \rightarrow B$ را کاهش‌ناپذیر گوییم هرگاه نداشته باشیم

$$C \rightarrow B \mid C \subset A$$

یعنی B با A وابستگی تابعی داشته ولی با هیچ زیرمجموعه‌ای از A، FD نداشته باشد.

اگر سمت چپ FD صفت ساده باشد به طور خودکار کامل است و کاهش‌ناپذیر.

۳) هیچ FD در آن افزونه نباشد. FDای افزونه است که اگر آن را از مجموعه F حذف کنیم، F^+ تغییر نکند.

با توجه به ۳ خاصیت فوق می‌توان الگوریتم زیر را برای تولید مجموعه کاهش‌ناپذیر از FDهای F را ارائه نمود.

```
Algorithn: Finding a minimal cover G from F
Set G := F
Replace each FD such x →{Aa, A2,…, An} in ∈ by n FD x→A1, x→A2, …, x→An
For each FD x→A in G for each attribute B that element of x
        If ((G−{x−A}) ∪ {x −{B} −A}) ≡ G then replace with (x −{B}) →A in G
For each remaining FD x→A in G
         If (G−{x→A}) ≡ G then Remove X→A from G
* F≡G, iff ∀x∈F, G, x+ in F= x+ in G
```

۹-۳ نرمال‌سازی

به‌طور کلی فرایند نرمال‌سازی مبتنی بر مفهوم FD تعریف و اجرا می‌شود. برای این منظور در نظر بگیرید که مجموعه تمام صفات موجود در مدل رابطه‌ای در یک رابطه قرار داده شده است و مجموعه FDهای این رابطه نیز در اختیار است. البته نرمال‌سازی روی مجموعه حداقل FDها اعمال می‌شود. بنابراین باید مجموعه کاهش‌ناپذیر FDها تولید شود. در ادامه عملیات نرمال‌سازی را تعریف نموده و با ذکر مثال شرح می‌دهیم.

۹-۳-۱ نرمال اول

رابطه R در 1NF[۲۲] است اگر و فقط اگر تمام صفات آن تک‌مقداری (اتمیک) باشد. فرم 1NF در واقع مفروضات مدل رابطه‌ای را بیان می‌کند. بنابراین با قبوا محدودیت‌های مدل رابطه‌ای، می‌توان گفت که هر رابطه‌ای در نرمال اول می‌باشد.

۹-۳-۲ نرمال دوم

نرمال دوم (2NF[۲۳]) مبتنی بر وابستگی تابعی کامل (FFD) تعریف می‌شود. همچنین قبل از تعریف نرمال دوم نیاز به تعاریف زیر می‌باشد.

وابستگی x→y را یک FDD گوییم اگر حذف هر صفت A از x به معنای قابل اعمال نبودن آن FD باشد.

یک صفت را صفت اولیه می‌گوییم اگر عضو یک کلید کاندید باشد. همچنین صفتی را غیر اولیه گوییم اگر عضو هیچ کلید کاندیدی نباشد.

---

[۲۲] First Normal Form

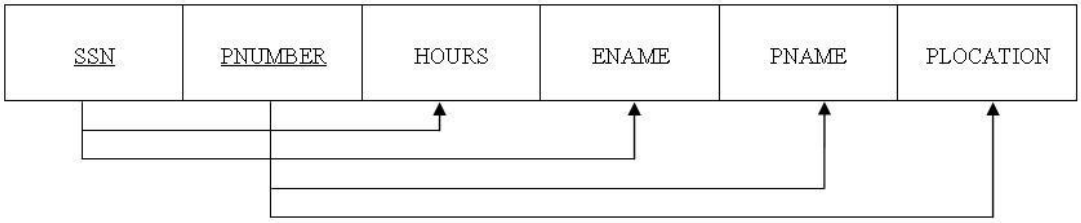
[۲۳] Second Normal Form

اکنون می‌توانی نرمال 2NF را تعریف نماییم. یک رابطه در 2NF است اگر داشته باشیم.

- 1NF باشد و
- هر صفت غیر اولیه با کلید اصلی وابستگی کامل داشته باشد.

جهت شـرح دقیق‌تر نرمال دوم مثال زیر را در نظر بگیرید. رابطه WORKS_ON به همراه صـفات و مجموعه FDهای آن ارائه شده است.

WORKS_ON



در این مثال ما ۳ صفت غیر اولیه داریم که FDD به کلید اصلی نیستند پس رابطه در 2NF نیست.

{ SSN, Pnumber} → Ename

{ SSN, Pnumber} → Pname

{ SSN, Pnumber} → Plocation

زمـانی کـه یـک رابطه که در نرمال دوم نیســت، باید آن را به روابطی در 2NF تبدیل نمود. برای این منظور جدول را به جداول کوچک‌تر می‌شکنیم که آن‌ها 2NF باشند. تجزیه یک جدول به جدول دیگر نباید FDها را حذف کند و اگر FD تولید می‌شود باید با توجه به قوانین آرمسـترانگ باشد. بنابراین رابطه فوق به روابط زیر شکسته می‌شود. روابط جدید همگی در 2NF هستند.

SSN	Ename

PNumber	Pname	Plocation

SSN	Pnumber	Hours

الگوریتم زیر جدول 1NF را به جدول 2NF تجزیه می‌کند:

۱) هر بخش از کلید اصلی را که ایجاد وابستگی جزئی کرده است با همه وابسته‌های آن کنار هم بگذار.

۲) کل کلید اصلی را با صفات باقی‌مانده کنار هم بگذار.

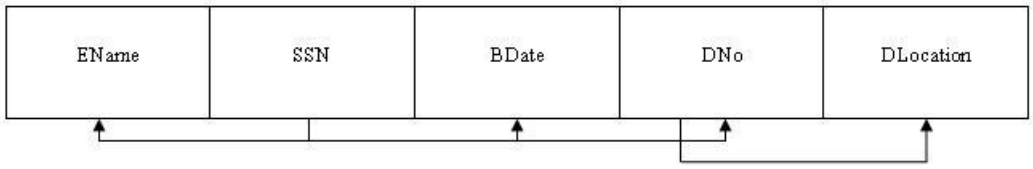
۳) صفات کلیدی را به عنوان کلید خارجی به ۲ اضافه کن.

۹-۳-۳ نرمال سوم

نرمال سوم (3NF[۲۴]) مبتنی بر یک نوع خاصی از وابستگی تابعی تعریف می‌شود که وابستگی تعدی[۲۵] نامیده می‌شود. بنابراین قبل از تعریف 3NF، وابستگی تعدی تعریف می‌شود.

وابستگی $x \rightarrow y$ را یک وابستگی تعدی گوییم اگر صفت z به همراه وابستگی‌های $x \rightarrow z$ و $z \rightarrow y$ وجود داشته باشد که وابستگی $x \rightarrow y$ از آن‌ها استخراج شده است.

به‌عنوان مثال رابطه زیر به‌همراه صفات و وابستگی‌های آن را در نظر بگیرید. در این رابطه وابستگی $SSN \rightarrow DLocation$ یک وابستگی تعدی است که از طریق وابستگی‌های $SSN \rightarrow DNo$ و $DNo \rightarrow DLocation$ بدست آمده است.



رابطه‌ای در نرمال سوم (3NF) است اگر شرایط زیر را داشته باشد.

- 2NF باشد و
- هر صفت غیر اولیه، وابستگی بی‌واسطه به کلید اصلی داشته باشد. (وابستگی تعدی به کلید اصلی نداشته باشد)

مثال قبل نشان می‌دهد که رابطه در نرمال سوم نیست، چون صفت غیر اولیه DLocation نسبت به کلید اصلی (SSN) وابستگی تعدی دارد. بنابراین باید آن را تبدیل به روابط 3NF کنیم. برای این منظور رابطه به نحوی به روابط دیگر شکسته می‌شود که روابط نتیجه همگی 3NF باشند. روال زیر برای این منظور قابل استفاده است.

- صفتی را که وابستگی تعدی ایجاد کرده با همه وابسته‌های آن کنار بگذار.
- کلید اصلی را با صفات باقی‌مانده کنار بگذار.
- صفت‌های کلیدی را به عنوان کلید خارجی به مرحله اول اضافه کن.

در 3NF باید همه FDها از کلید اصلی ناشی شود. نتیجه شکست برای رابطه قبل به‌صورت زیر است.

---

[۲۴] Third Normal Form

[۲۵] Transitive Dependecy

EName	SSN	Bdate	DNo

DNo	DLocation

BCNF ۴-۳-۹

نرمال 3NF با جداولی که هر سه شرط زیر را دارند ممکن است مشکل داشته باشد. در این صورت باید این نوع جداول را تا سطح بالاتری نرمال کرد.

- جدول دارای حداقل دو کلید کاندید باشد.
- این کلیدهای کاندید ترکیبی باشند.
- این کلیدهای کاندید ترکیبی، صفت‌های مشترکی داشته باشند.

برای رفع مشکل در شرایط فوق نرمال‌سازی BCNF[۲۶] ارائه شده است. رابطه‌ای BCNF است که در آن هر دترمینان کلید کاندید باشد. (صفات سمت چپ باید کلید کاندید باشد) جدولی در BCNF هست که صفات آن فقط به کلیدهای کاندیدش وابستگی تابعی داشته باشد.

نرمال BCNF از 3NF سخت‌گیرانه‌تر است و جامع‌ترین تعریف نرمال‌سازی بر مبنای وابستگی تابعی را به طور مستقل ارائه می‌دهد.

[۲۶] Boyce Codd Normal Form