پوهنتون کابل

پوهنځی کمپیوترساینس

اساسات بانک معلومات

Database Fundamentals

تهیه کننده پوهنیار محمد شعیب "زرین خیل"
سال 1388


بسم الله الرحمن الرحيم

پیشگفتار

تکنالوژی کمپوتر در جهان امروزی تقریباً در تمام ساحه های زنده گی بشر محسوس میباشد. اگر عمومی تر فکر شود، کمپیوتر تنها ماشین الکترونیکی که اصطلاحاً به این نام مسمی است نبوده، بلکه در اکثر ماشین آلات، سیستم ها، شهرها، خانه ها و غیره استفاده میشود. به همان تناسبیکه ضرورت و استفاده به کمپیوتر و سیستم های کمپیوتری در زنده گی روزمره محسوس میشود، به همان تناسب علم کمپیوتر ساینس انکشاف و وسعت پیدا نموده است.

بانک معلومات (Database) یک از شقوق بزرگ و با اهمیت در کمپیوتر ساینس میباشد. استفاده از بانک های معلومات در نواحی مختلف زنده گی انسانها ضرورت بوده و در حال انکشاف است. بطور نمونه، تجارت های بزرگ و کوچک بانک معلومات استفاده مینمایند. کتابخانه ها بانک معلومات استفاده مینمایند. ممالک مختلف جهان جهت ثبت نمودن معلومات در مورد فرد فرد نفوس و تابعین خود بانک معلومات استفاده مینمایند. بصورت کلی بانک معلومات نظر به استفاده و اهمیت آن یکی از علوم و فنون معتبر در تکنالوژی کمپیوتر به شمار رفته و دانستن اساسی علم بانک معلومات یک شرط حتمی برای استفاده کننده گان بانک معلومات میباشد.

امروز اکثر مردم جهان استفاده از کمپیوتر را بلدیت دارند. یک شخص، طور مثال، میتواند با استفاده از یک پروگرام کمپیوتر یک یا چند صفحه را دیزاین و چاپ نماید، حاشیه های صفحات را تنظیم نماید، یک جدول را در کمپیوتر ایجاد نماید، بعضی محاسبات را اجرا نماید و غیره. در کنار این همه چه بهتر که این اشخاص اساسات بانک معلومات را نیز بیاموزند. این را بفهمند که چطور یک بانک معلومات ایجاد میشود، کدام قواعد در نظر گرفته میشود، کدام عناصر بیشتر اهمیت دارند، چگونه معلومات بهم ارتباط پیدا میکنند و غیره. سیستم های اداره‌ء بانک معلومات (DBMSs) عبارت از نرم افزارهای اند که توسط آنها بانک معلومات ایجاد و استفاده شده میتوانند. دانستن اساسات بانک معلومات استفاده کننده گان DBMSهای مدرن را قادر به استفاده کامل و وسیع از این نرم افزار میسازد.

نوشتن یک اثر تخنیکی کار نسبتاً دشوار میباشد. زمانیکه یک اثر یا کتاب نود در صد تکمیل شده باشد، باز هم نود در صد کار ضرورت دارد تا واقعاً و عملا تکمیل شود. در این عرصه بازهم تکنالوژی کمپیوتر کارها را به مراتب آسان ساخته است. طور مثال اگر یک اصطلاح تخنیکی ضرورت به اصلاح داشته باشد و آن در کتاب صد صفحه یی 120 بار استفاده شده باشد، با براه انداختن یک دستور در ظرف کمتر از چند ثانیه اصلاحات تطبیق میشوند. این اثر بحیث یک مبتدی در ساحه‌ء بانک معلومات، با استفاده از کتب با اعتبار، نوشته شده و بخاطر عناوین و

کارهای به سطح بالا در بانک معلومات استفاده شده میتواند. خواننده گان میتوانند این اثر را به شکل انفرادی، گروپی، و یا هم جهت تدریس استفاده نمایند.

در این اثر کوشش بعمل آمده تا تمام مفاهیم بصورت واضح و عام فهم ارایه شوند. در قسمت ترتیب نیز کوشش شده است تا این اثر بصورت درست و واضح تنظیم شود. عناوین مختلف با مثال ها و تفصیل لازمه شرح شده اند. اصطلاحات هر کدام در استعمال اول با ذکر اصل کلمه به لسان انگلیسی، توضیح شده اند. در مواقع ضرورت هر اصطلاح با تفصیل بیشتر ارایه شده است. بصورت عمومی، اصطلاحات رایج در لسان دری جستجو شده و در این اثر استفاده شده اند. اصطلاحات انگلیسی که تا حالا معادل آنها در لسان دری موجود نبودند با مشورهء استادان دیپارتمنت کمپیوتر ساینس پوهنتون کابل و بعضی اشخاص مسلکی کمپیوتر ساینس ترجمه گردیده و استفاده شده اند. لست مکمل اصطلاحات استفاده شده و ترجمهء آنها در این اثر ضمیمه میباشد.

یکی از اهداف اساسی این اثر ایجاد ضمینه بخاطر بحث بالای موضوعات و عناوین مربوطهء آن میباشد. دلایلی که این نظریه را تقویه مینمایند عبارت اند از:

- تمرکز بیشتر بالای طرح، دیزاین، تطبیق و اجراء بانک معلومات از صفر. یعنی بانک های معلومات ایجاد میشوند که قبلا موجود نبودند. راه های مختلف جهت ایجاد یک

بانک معلومات استفاده شده میتواند که این خود بحث برانگیز بوده و مهارت استفاده کننده گان را بالا میبرد.

- عناوین شرح شده در این اثر کار عملی در ساحه بانک معلومات را ایجاب مینماید. گروپ های استفاده کننده گان در زمان عملی ساختن آنها به بحث های پرداخته و مهارتهای خود را بالا میبرند. بیشتر عناوینیکه چنین اند، با تفصیل بیشتر توضیح شده اند.
- طرق مختلف دیزاین پیشنهاد شده اند. طور مثال، جهت ایجاد مودل معلومات اولیه دو اختیار توضیح شده اند: مودل E-R و مودل UML.

تصحیح آثار تخنیکی، بطور خاص در ساحهء کمپیوتر ساینس، مهارت و دقت زیادی ضرورت می داشته باشند. با اظهار سپاس و تشکر فراوان از استاد محترم پوهندوی محمد همایون "ناصری" که قبول زحمات فرموده مسولیت دشوار تصحیح این اثر را بعهده داشتند و با نظریات و ربهنمایی های پربار و عالمانهء خود این اثر را ارزشمند ساخته و بنده را قادر به تکمیل آن نمودند.

با عرض حرمت

پوهنیار محمد شعیب "زرینخیل"

فهرست و عناوین

صفحه عنوان

اساسات دیتابیس

(Database Fundamentals)

Data-Modeling and Database Implementation

این صنف یک صنف دیتابیس است که در جریان آن مفاهیم اساسی دیتابیس شرح شده و زمینه را جهت طرح، مودل سازي، ایجاد و اجراء دیتابیس فراهم میسازد. نوآموزان این رشته با استفاده از سمبولها، اشکال و مودل هاي شرح شده در این نوت قادر خواهند شد تا دیتابیس هاي فني و معیاري ایجاد نمایند. در قسمت عملی شاگردان و نوآموزان وسایل و پروگرام هاي دیزاین دیتابیس را استفاده خواهند نمود و قادر خواهند شد تا دیتابیسی را از ابتدا یا (Scratch) ایجاد نمایند. همچنان جهت تطبیق دیتابیس پروگرام ها یا DBMS[1]های از قبیل "Ms-Access" و "MySQL[2]" استفاده شده اوامر آنها اجراء میگردند.

هدف عمده و اساسی تهیه این نوت بکارگیری مهارتها در استفاده از دیتابیس های رابطه اي (Relational-Databases) برای انجام کارهای عملی در تدریس، تحقیق، معاملات تجارتي، و غیره موارد میباشد. همچنان استفاده کننده گان این نوت در ختم پروگرام، توانایی خواندن، تفسیر کردن و ترجمه دیتامودل های ساده یا (Simple Data-Models)،

[1] DBMS (Database Management System)
[2] SQL (Structured Query Language

تطبیق مراحل نارمل سازی یا (Normalization) بالای جداول دیتابیس و اجراء Relational-Databaseها را خواهند داشت.

خصوصا در ختم این پروگرام هر استفاده کننده توانایی های ذیل را خواهد داشت:

-خواندن و ترجمه دیاگرامها ی Entity-Relationship(E-R)

-اجراء مراحل نارمل سازي بالای یک مودل Relational که آنرا به درجه بالایی Normalization میرساند.

-تبدیل نمودن یک Relational-Model در دیتابیس، به دستورهای SQL تا در DBMSها قابل اجراء باشند.

موضوعات ذیل شامل این نوت بوده و تدریس میشوند:

- The Data-Modeling Process
- Basic Relational Concepts
- The Process of Normalization
- Relational-Algebra
- SQL

- Guidelines for Mapping a Data-Model Into a Relational-Database

مقدمه

کمپیوتر اساسا یکنوع ماشین الکترونیکی استکه معلوماتی را در خود ذخیره نموده و با استفاده از این معلومات ذخیره شده کارهای بی نهایت عمده را در اندک زمانی بر طبق هدایاتیکه به او داده میشود بانجام میرساند. باید دانست که این دستگاه عام المنفعه تخنیکی یک ماشین ساده نیست. این دستگاه الکترونیکی یا باصطلاح کمپیوتر امروز نتیجه سالها و قرن ها پژوهش های علمی استکه دانشمندان تحقیقات و پژوهش های خود را یکی بعد از دیگری بدسترس دانشمندان و پژوهشگران نسلهای مابعد خویش قرار داده بالاخره مجموعه تمام این پژوهش های علمی این اعجوبه عالم را بمیان آورده است.

دانشیکه همه این مطالعات و تحقیقات علمی را انجام داده و هنوز هم مطالعات خویش را در راه پیشرفت و انکشاف بیشتر این اعجوبه معاصر ادامه داده و حتی عصری بنام عصر کمپیوتر را برای خویش اختصاص داده است بنام دانش کمپیوتر یا Computer Science یاد میشود.

این علم دلچسپ و عام المنفعه که هر نوع موضوعات تخنیکی، اوپراتیفی پیشرفت ها و انکشافات معاصر را بسرعت سرسام آوری تحلیل، مطالعه و تدقیق مینماید از سه خصوصیت ممتاز برخوردار است:

1. این دانش بسرعت انکشاف نموده و انکشاف آن بسرعت ادامه دارد.

2.صورت استفاده از این دانش ساحه بی نهایت وسیعی را در تمام امور زنده گی معاصر احتواء مینماید.

3.کسب و آموزش این دانش برای هرنوع استعداد فکری و دارنده گان عقل سلیم میسر است.

ولی باید متوجه بود که آموزش این دانش کار پیگیر و مطالعه دوامدار را توصیه مینماید.

بطور خاص، استفاده و دانستن تکنالوژی دیتابیس، امروز یک جزء مهم اکثر امور زندگی را تشکیل میدهد. دیتابیس ها در بسیاری موارد استفاده میشوند. طور مثال در قسمت تجارت از طریق انترنت (E-Commerce) ، پیدا نمودن اشیاء مورد نیاز، پیدا نمودن کتابها در کتابخانه ها بصورت Online و یا Offline ، پیدا نمودن وظیفه، اشخاص و دیگر ضروریات از طریق شبکه های محلی، ملی و بین المللی، دیتابیس ها وسیعا استفاده میشوند.

دیتابیس ها توسط هزاران شرکت بزرگ و کوچک و ملیون ها فرد به سطح بین المللی استفاده میشوند. یک سروی تعداد دیتابیس های فعال در سال 2002 به سطح تمام جهان را اضافه از 10 ملیون نشان میدهد]4، ص. 3[.

در این نوت علم دیتابیس بصورت مبتدی توضیح و جهت تدریس آماده شده است. همچنان کوشش زیاد بعمل آمده است تا مفهوم دیتابیس، تکنالوژی دیتابیس و تخنیک هائیکه

توسط آنها دیزاین و استفاده دیتابیس صورت میگیرد، توضیح شود.

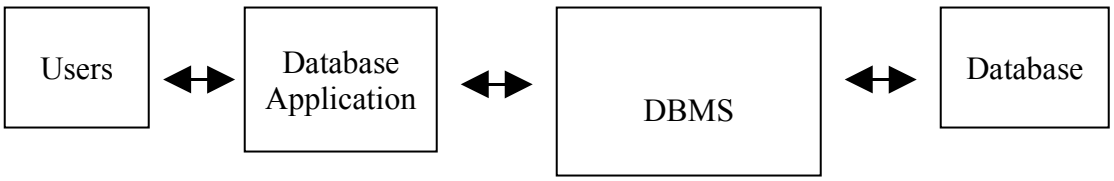
چون دیتابیس ساحه وسیع در کمپیوتر ساینس بوده و بحث هاي زيادي را احتواء مي نمايد. بناً این نوت در بر گیرنده تمام مسایل در مورد دیتابیس شده نمیتواند. با آنهم در ختم این پروگرام، استفاده کنندگان قادر خواهند بود تا دیتابیس های شخصی خود شان را ایجاد نمایند و خواهند توانست با اشتراک در گروپ ها به دیزاین دیتابیس های بزرگ که نیازمندي موسسات و شرکت هاي بزرگ را برآورده ساخته بتوانند، نیز بپردازند.

یک سیستم دیتابیس چی است؟

سیستم دیتابیس اساسا یک سیستم ثبت معلومات به شکل کمپیوتری یا "Computerized Record-Keeping System" است [3، ص. 6].

یک سیستم دیتابیس بصورت عمومی متشکل از چهار قسمت ذیل بوده که در شکل (1) نیز نشان داده شده است [4، ص. 12-13].

1. Database
2. DBMS (Database Management System)
3. Database Applications
4. Users



شکل(1) اجزاء یک سیستم دیتابیس

تاریخچه

اوایل دهه 1960 سیستم های پروسس فایل ها که دیتا در آنها ذخیره میشد در سیستم های کمپیوتر مروج گردید. اواخر دهه 1960 کوشش های ابتدایی جهت ایجاد سیستم های دیتابیس آغاز گرید. در دهه 70 اولین دیتابیس ها به اشکال Hierarchical و Network بمیان آمدند. همچنان در این دهه دیتا مودل های Relational توسط شخصي بنام E. F. Codd و بعضی کسان دیگر طرح و انکشاف داده شدند. در دهه 80 دیتابیس های تجارتی (Commercial) توسط شرکت های Sybase ، Oracle و غیره تولید شدند. در دهه 90 سیستم های کمپیوتری Client-Server بوجود آمدند. در همین زمان استکه نوع دیتا (Data-Type) های مختلف، طور مثال text جدا از number ، در ساحه دیزاین دیتابیسها استفاده شدند. برعلاوه دهه 90 شاهد بوجود آمدن دیتابیس های با ذخیره نمودن مقدار های بزرگ دیتا و دیتابیس های شی گرا (Object-Oriented) میباشد. در دهه اخیر (بعد از سال 2000) انکشافاتیکه در ساحه دیتابیس صورت گرفته اند طور ذیل میباشند:

-تعریف نوع دیتا توسط استفاده کننده (User-defined Data-Types)

-وسعت استفاده از دیتابیس های Object-Oriented

-استفاده دیتابیس در سیستم های Cline-Server

-دیتابیس هاي توزیعي (Distributed-Databases)

-استفاده از Data-Warehouseها جهت ذخیره نمودن مقادیر زیاد دیتا (Tera-Bytes)

دیتابیس (Database)

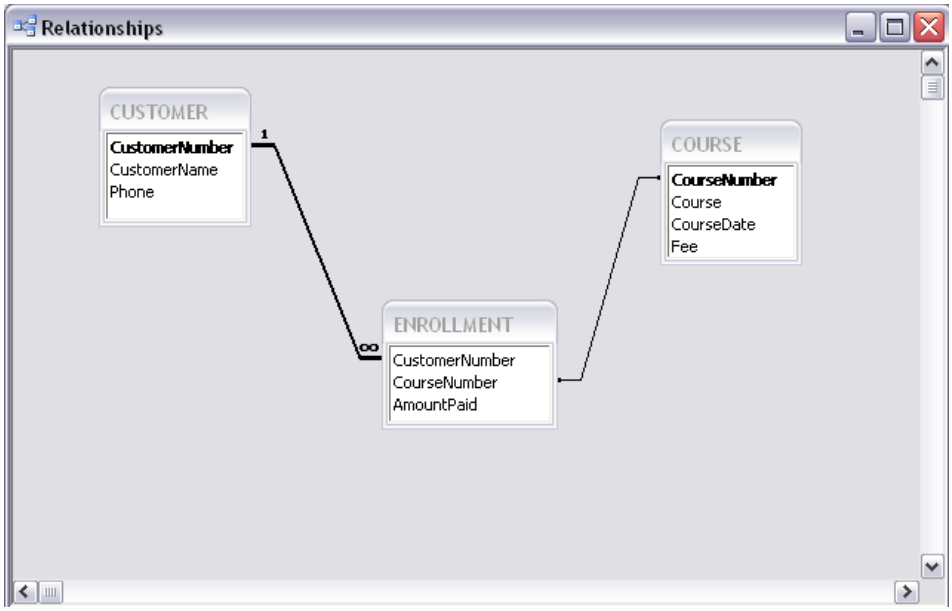
دیتابیس عبارت از مجموعه تعریف شده داخلی (Self-Describing) جداول مرتبط (Related-Tables) و ساختمان های دیگر میباشد]4، ص. 13[.

دو نکته اساسی در تعریف دیتابیس عبارت از تعریف شده داخلی (Self-Describing) و جدول های مرتبط (Related-Tables) اند که در ذیل توضیح مي گردند.

-مفهوم تعریف شده داخلی عبارت از توضیح ساختمان یا Structure هر جدول (Table) در خود دیتابیس است، یعنی استفاده کننده برای دانستن مشخصات یک Table در داخل دیتابیس ضرورت به معلومات بیرونی ندارد.

-به عین ترتیب اصطلاح جدول های مرتبط (Related-Tables) ، ارتباط بین جدول ها (فیلدهای جدول ها)، از جمله مشخصات دیگر دیتابیس میباشد.

تمام DBMSها یک سیستم برای نمایش ساختمان دیتابیس یا (Database-Structure) را دارند. طور مثال شکل(2) دیاگرام تولید شده توسط پروگرام Ms-Access میباشد که نشان دهنده روابط (Relationships) بین جدول های دیتابیسی به نام "ArtCourse" است.



شکل(2) نمایش Relationshipها بین سه جدول در یک دیتابیس

-دیتا که ساختمان یک دیتابیس را نشان میدهد به نام "Metadata" یاد میشود [2، ص. 23]، [3، ص. 47]، [4، ص. 13]. مثال های Metadata عبارت اند از: نام جدول ها، نام ستونها و جدول های که به آن ارتباط

دارند، مشخصات یا (Property) جدول ها و غیره.

-دیتا که توسط استفاده کننده (Users) در داخل جدول ها جابجا میشود، بنام " Userdata" یاد میشود. مثال های Userdata عبارت از Recordها یا سطرهای موجود در جداول میباشد.

دیتابیسها نظر به ساختمان شان به چهار نوع ذیل اند:

-Hierarchical یا شجری (شکل Original دیتابیسها)

-Network یا شبکه ای (نمونه جدید)

-Relational (بیشتر استفاده میشود، در این نوت نیز استفاده شده است)

-Object-Oriented (از جمله انکشافات جدید در دیزاین دیتابیس میباشد)

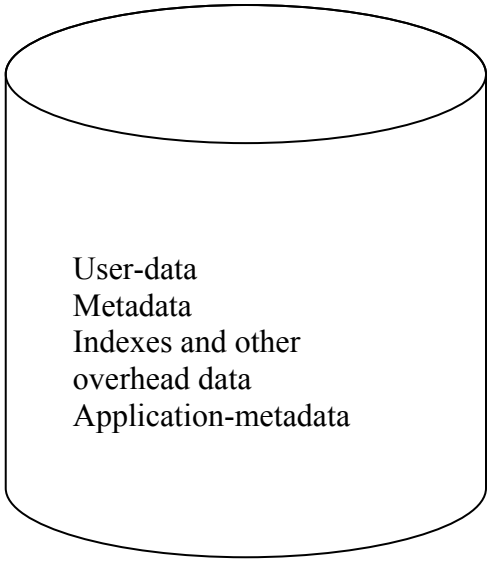
دیتابیسها نظر به سایز یا اندازه شان به سه نوع ذیل تقسیم بندی میشوند:

- Personal- (دیتابیسهای کوچک تهیه شده در Ms-Access)

- Departmental / Workgroup- (دیتابیسهای متوسط تهیه شده در MySQL و SQL_Server)

- Enterprise- (دیتابیسهای بزرگ تهیه شده در Informix، Sybase، Oracle و SQL-Server)

شکل (3) اجزاء دیتابیس را نشان میدهد]4، ص. 14[. از جمله Userdata و Metadata توضیح شد. علاوتا یک دیتابیس دارای Indexها و دیگر ساختمان ها میباشد که در اجراء، تهیه و استفاده دیتابیس نقش دارند. قسمت اخیر عبارت از Application-Metadata است که متشکل از فورم ها (Forms) و راپورها (Reports) میباشد. پروگرام Access این قسمت را نیز جزء دیتابیس حساب نموده و استفاده مینماید.



شکل(3) محتوا یک دیتابیس

DBMS (Database Management System)

در بین دیتابیس و استفاده کنندگان سیستم بصورت فزیکی یک طبقه از Software موجود است که 'Database-Manager' یا 'Database-Server' و یا هم DBMS یا سیستم اداره دیتابیس گفته میشود]3،ص. 44[. به عبارت دیگر، DBMS عبارت از یک پروگرام کمپیوتر بوده که توسط آن یک دیتابیس ایجاد، اجراء و اداره میگردد. DBMS دستورها را ذریعه لسان SQL یعنی (Structured Query Language) دریافت نموده و آنها را بالای دیتابیس اجراء می نماید. DBMS یک سیستم یا پروگرام وسیع و پیچیده

میباشد که توسط یکی از کمپنی های تولید کننده Software تولید میشود. طور مثال پروگرام Ms-Access یک DBMS است. مثال های دیگر آن عبارت اند از: Oracle تولید شرکت Oracle، DB2 تولید شرکت IBM و 'SQL-Server' مربوط شرکت مایکروسافت. ده ها DBMS دیگر نیز وجود دارند، اما این چهار بطور وسیع استفاده میشوند.

کارهایرا که یک DBMS قادر است اجراء کند عبارت اند از:

-اداره دکشنری دیتا

-اداره ذخیره دیتا

-ایجاد دیتابیس

-ایجاد جداول (Tables)

-ایجاد ساختمان های کمکی

-خواندن دیتا از یک دیتابیس

-تغییر آوردن در دیتا یک دیتابیس

-حفظ و نگهداشت ساختمان های داخلی دیتابیس

-اجراء اوامر

-کنترول Concurrency

-امنیت (Security)

-تهیه Backup و مطمئن بودن دیتا

DBMS مسئول جابجا سازی دیتا در مورد عناصر و ساختمان دیتابیس یعنی (Metadata) میباشد که بنام دکشنری دیتا نیز یاد میشود. DBMS، دکشنری دیتا را جهت پیدا نمودن دیتا مورد نظر و Relationshipها در داخل دیتابیس مورد استفاده قرار میدهد. یعنی برای پیدا نمودن یک Relationship ضرورت به داشتن Code نبوده، از طریق دکشنری دیتا پیدا میشود.

DBMS ساختمان های پیچیده اي را جهت حفظ نمودن دیتا ایجاد مینماید. این ساختمان ها ضرورت توضیح و پروگرام نمودن دیتا، جهت حفظ شدن، را مرفوع ساخته است. یک DBMS مدرن برعلاوه، اشکال مختلف دیتا را از قبیل Report Definitions، Screen Definitions، اوامر Data Validation، ساختمان های برای فایل های ویدیویی/تصویری و غیره را نیز حفظ مینماید.

توسط DBMS، یک دیتابیس ایجاد شده، و جداول به آن علاوه شده میتوانند. همچنان امکان علاوه نمودن ساختارهای کمکی چون Indexها نیز میباشد. طور مثال، اگر

در یک دیتابیس یک Table با "1000" سطر موجود باشد و یکی از ستون های Table عبارت از "DepartmentName" باشد و ستون متذکره دیپارتمنت اعضا لست شده در Table را مشخص نماید. دفعتا ضرورت میشود تا دیتا مشخص شده این Table نظر به "DepartmentName" پیدا شود. چون دیتابیس ذکر شده بسیار بزرگ میباشد، پیدا نمودن دیتا مورد نظر، فرضا از دیپارتمنت Accounting، وقت زیاد را خواهد گرفت. بنا ضرورت می افتد تا یک Index برای DepartmentName ایجاد شود و نشان دهد که کدام شخص عضو کدام دیپارتمنت است. ایجاد چنین Indexها مثال خوبی برای ساختمان های کمکی میباشد که توسط DBMS ایجاد و اجراء میشوند.

از جمله وظایف دیگر DBMS، خواندن و تغییر آوردن در دیتا یک دیتابیس است. طوریکه قبلا نیز گفته شد DBMS درخواست ها یا دستورها را ذریعه SQL از استفاده کننده گرفته و به وهله اجراء میگذارد. وظیفه بعدی DBMS عبارت از حفظ و نگهداشت ساختمان های داخلی دیتابیس است. طور مثال، ضرورت می افتد تا شکل یا فارمت جداول و یا دیگر ساختمان های کمکی در داخل یک دیتابیس، وقتا فوقتا تغییر داده یا Update شود، جهت اجراء این کار DBMS بصورت اتوماتیک اجرات نموده و پروسه Update شدن را تکمیل مینماید.

بسیاری از DBMSها، امکان اجراء بعضی اوامر را به استفاده کننده میدهد که این اوامر در استفاده از دیتابیس یک امر مهم به شمار میرود. طور مثال اگر در

یک دیتابیس دو Table به اساس یک ستون (Field) با هم ارتباط (Relationship) داشته باشند و دیتا ستون متذکره در Table اولی اعداد (1-7) باشند. اگر اشتباها استفاده کننده در Table دومی در ستونیکه به اساس ستون متذکره از Table اولی ایجاد شده است، عدد (9) را وارد نماید، در دیتابیس غلطی های ایجاد خواهد شد. زیرا عدد (9) در ستون Table اولی موجود نیست. پس برای رفع چنین اشتباهات، DBMS اوامری را برای استفاده کننده اجازه میدهد که جلو اشتباهات را بگیرد. اوامر متذکره در DBMS بنام "Referential Integrity Constraint (RIC)" یاد شده توسط DBMS تطبیق میشوند.

سه وظیفه لست شده اخیر DBMSها بیشتر ارتباط میگیرد به قسمت اداره یا Administration دیتابیس. DBMS تصادف یا Concurrency را کنترول مینماید بدین معنی که کار یک استفاده کننده در عین دیتابیس با کار استفاده کننده دیگرتصادف یا تداخل نه نماید. این وظیفه مهم و پیچیده DBMS مربوط بخش اداره دیتابیس یا (Database Administration) میباشد.

همچنان، DBMS وظیفه امنیت (Security) دیتابیس را بعهده دارد.طور مثال کارکنان هر بخش تنها اجازه کار در یک قسمت معین دیتابیس را دارا میباشند. ویا استفاده کننده گان تنها بخش های از دیتا را می بینند که برای کارهای مربوطه آنان تعیین شده است، در حالیکه

مدیران یا دیزاینرهای دیتابیس قادر به دیدن قسمت های دیگر و تغییر آوردن در آن قسمت ها نیز میباشند.

در اخیر باید علاوه شود اینکه دیتابیس یک سرمایه مهم و پر ارزش، خصوصا برای کمپنی های بزرگ چون Amazon.com و نظیر آن میباشد. پس برای دیزاین و کار با دیتابیس باید از دقت زیاد کار گرفته شود که تا حد ممکنه دیتا بصورت مکمل در دیتابیس جابجا شود. این کار مستقیما ارتباط میگیرد به رفع پرابلم های Hardware و Softwareسیستم. DBMSها تا حد زیاد جلو چنین مشکلات را گرفته و با مهیا نمودن امکانات وسیع و گرفتن Backup دیتا، دقت و صحت کار را تضمین نموده اند.

Application-Programs

یک Application دیتابیس عبارت از یک پروگرام کمپیوتر و یا دسته از پروگرام های کمپیوتر میباشد که بحیث رابط بین استفاده کننده و DBMS استفاده میشوند [2، ص. 20] ، [4، ص. 15]. این پروگرام ها، طور مثال، دیتا را در اشکال فورمها و راپورها در دسترس استفاده کننده قرار میدهد. در حقیقت Application-Programها، دیتا موجود در دیتابیس را توسط دستورهای SQL، که از طرف استفاده کننده صادر میشود، خوانده و اجراء مینماید. Applicationها هم توسط کمپنی های تولید Software تولید میشوند و هم خود استفاده کننده میتواند آنها را بنویسد. این نوت استفاده کننده گان را قادر میسازد تا خود پروگرام هاي Applicationها را بنویسند.

Applicationهای دیتابیس وظایف ذیل را اجراء کرده میتوانند:

-ایجاد و پروسس فورم ها (Froms)

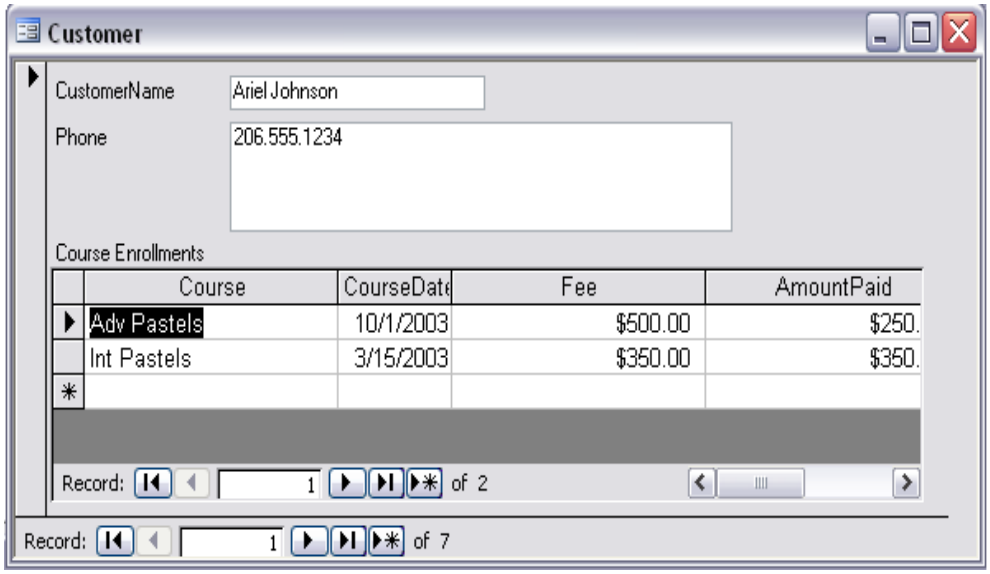
-اجراء کیوري ها (Queries)

-ایجاد و پروسس راپورها (Reports)

-اجراء Applicationهای منطقی یا Logic

-کنترول Applicationها

اولتر از همه، یک پروگرام Application، فورم ها را ایجاد نموده و تحت پروسس قرار میدهد. شکل (4) مثال یک فورم تهیه شده در Access میباشد. اگر دقت شود، دیده میشود که ساختمان Tableها یا جداول استفاده شده در این فورم پنهان میباشند یعنی نشان داده نشده اند. معلومات مورد نظر به شکل بسیار ساده در دسترس استفاده کننده بوده و نشان داده شده اند. با استفاده از فورم سیستم داخل نمودن دیتا یا Data Entry به دیتابیس ساده تر شده میتواند.

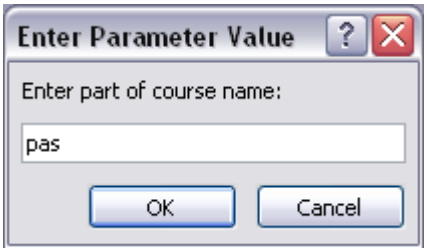


شکل (4) مثال یک فورم تهیه شده از یک Relation

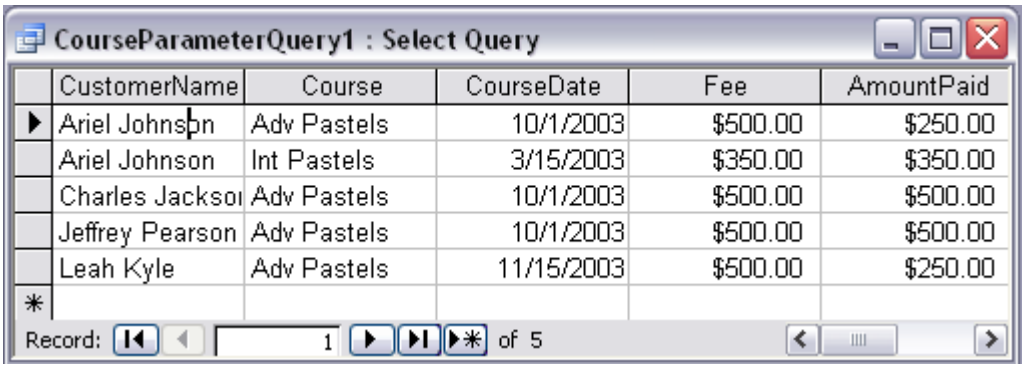
در عقب فورم، Application اکسس بالای دیتابیس عملیات را نظر به درخواست استفاده کننده اجراء مینماید. Application دستورهای SQL را بخاطر داخل کردن دیتا، تغییر آوردن در دیتا و یا از بین بردن دیتا در یکی از جدول های موجود در دیتابیس، ایجاد و اجراء مینماید.

وظیفه دوم پروگرام های Applicationها عبارت از براه انداختن کیوري ها میباشد. Application اولا یک درخواست کیوري را ایجاد نموده به DBMS می فرستد. نتیجه دوباره به شکلیکه توسط استفاده کننده از طریق Application مشخص شده است، بروی وسیله خروجی نشان داده

میشود. شکل(5) گفته های بالا را تصدیق مینماید. در شکل (5-الف) Application، نام کورس و یا قسمتی از نام کورس را از استفاده کننده میخواهد. استفاده کننده سه حرف کلیدی 'pas' را داخل نموده ، OK می نماید. بعدا Application با استفاده از دستور SQL، ستون Course را در دیتابیس جستجو نموده و ریکوردهای تمام کورس ها را که سلسله حروف pas جزء نام آنها باشد، در شکل (5-ب) نشان میدهد.



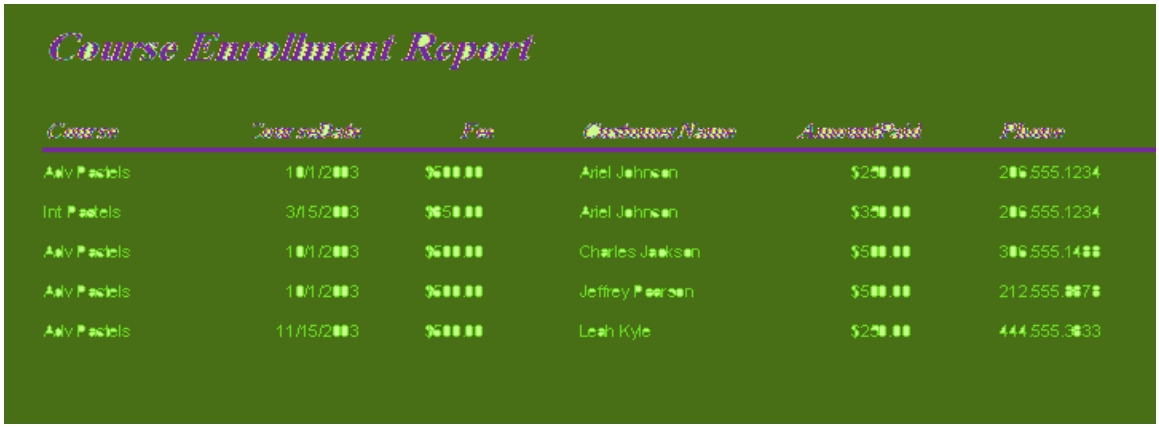
شکل (5-الف) شکل پارامتری یک کیوري



شکل (5-ب) نتیجه براه انداختن کیوري

شکل (5) مثال یک کیوري

وظیفه بعدی (سوم) یک Application-Program شباهت زیاد به وظیفه قبلی (دوم) آن دارد. Application-Program اولا دیتابیس را بخاطر پیدا نمودن دیتا در شکل کیوري (با استفاده از SQL) تنیظم مینماید. بعدا نتایج را در شکل راپور (Report) نشان میدهد. یعنی اول کیوري ایجاد شده و بعد راپور به اساس آن ایجاد میشود. شکل (6) راپور تهیه شده توسط پروگرام Application را نشان میدهد.



شکل (6) مثال راپور

وظیفه بعد ی Application-Programها عبارت از اجراء عملیه های منطقی یا (Application Logic) است. در بسیاری موارد این وظیفه بسیار مهم بوده، جلو اشتباهاتی را میگیرد. طور مثال استفاده کننده درخواست ده عدد پیراهن را از یک فروشگاه لباس مینماید، در حالیکه تعداد پیراهن های موجود در فروشگاه هشت عدد باشد، چی اتفاق خواهد افتاد؟ - در این حالت Application-Program باید نظر به ضرورت تصمیم بگیرد، یعنی عوض اجراء دستور، باید پیام یا Message با مفهوم به وسیله خروجی صادر شود.

وظیفه بعدی یا اخیر پروگرام های Application عبارت از کنترول Applicationها میباشد. کنترول Application به دو طریقه صورت میگیرد. اول اینکه در وقت نوشتن یک Application استفاده کننده تنها میتواند از اختیارات با Optionهای منطقی استفاده نماید. Application شاید یک مینوی دستورات را در اختیار استفاده کننده بدهد. در

اینصورت استفاده کننده باید توسط Application مطلع ساخته شود تا کدام دستورها قابل انتخاب و یا قابل اجراء میباشند. طریقه دوم کنترول توسط Application طوریست که Application باید دیتا را به کمک DBMS کنترول نماید. Application طوری عمل مینماید تا DBMS هدایت و راهنمایی شود، طور مثال تغیرات در تمام دیتا آورده شود و یا یک قسمت آن تغییر داده شود و غیره.

Users

استفاده کننده گان (Users) دیتابیس به سه گروپ تقسیم میشوند]3، ص. 10[.

- Database Administrators (DBAs)
- Application Programmers (Developers)
- End Users

1. اداره کنندگان دیتابیس

اداره کنندگان دیتابیس (Database-Adiministrators) یا DBAs وظیفه کنترول و اداره دیتابیس در داخل سیستم را به عهده دارند. آشنايي با لسان هاي كيوري براي

این گروپ از استفاده کنندگان ضروري مي باشد. ایشان برعلاوه بخش های دیگر SQL توسط بخش SQL-DCL[3] میتوانند دیتابیس را اداره و کنترول نمایند.

اداره کننده یك دیتابیس باید اهمیت دیتا ذخیره شده و اهمیت استفاده از دیتا ذخیره شده در آن دیتابیس را بداند. با درك این همه اداره کننده دیتابیس میبتواند منحیث یك شخص مسول بصورت درست و کامل یك دیتابیس را اداره نماید.

وظایف یك اداره کننده دیتابیس عبارت اند از:

-داشتن معلومات در مورد ساختمان ها و اجزاء داخلي دیتابیس و روابط منطقي بین آنها

-داشتن معلومات در مورد ساختمان فزیکي دیتابیس

-داشتن معلومات در مورد مسایل امنیتي (Security) و تطبیق قوانین و Integrityها

-داشتن معلومات در مورد پالیسي هاي Backup و Recovery دیتابیس

-توضیح مسایل اداره دیتابیس به استفاده کنندگان عمومي (End-Users) در موارد:

[3] SQL-DCL (SQL Data Control Language)

-معلومات تخنيكي

-تحليل معلومات مورد ضرورت استفاده كنندگان

-نگه داشتن توازن جهت جلوگيري از مشكلات عدم دسترسي استفاده كنندگان عمومي به ديتا

-كنترول، ارزيابي و پاسخ دهي به ذخيره كردن ديتا، تغير دادن ديتا، پاك كردن ديتا و گرفتن ديتا از ديتابيس

2.پروگرامرهای Applicationهای ديتابيس

اين گروپ استفاده كننده گان مسوليت نوشتن پروگرام های Application را در داخل ديتابيس به عهده دارند. پروگرامرهاي ديتابيس نظريات و پيشنهادات اداره كننده ديتابيس (Database-Administrator) و استفاده كنندگان عمومي ديتابيس را در نظر گرفته و در ديتابيس تطبيق نمايند. اشخاص شامل در اين كتگوری بايد مهارت كافی در ديتابيس به سطح Coding داشته باشند. اين اشخاص مي توانند يكی از لسان های پروگرام نويسی سطح بالايی (4GL)[4] را استفاده نمايند. لسان های پروگرام نويسی قابل استفاده برای نوشتن

[4] 4GL (Fourth Generation Languages)

Application-Programها در دیتابیس عبارت اند از: Visual-Basic، COBOL، PL/1، C++، Java و غیره. پروگرامهای یاد شده با استفاده از اوامر SQL در داخل DBMS با استفاده از شبکه ها اجراء میشوند. در بسیاری از DBMSهای جدید ، Application-Programها به شکل Online دیزاین شده و اجراء مي شوند.

وظایف یك پروگرامر دیتابیس یا Database-Developer عبارت اند از:

-ایجاد کردن دیتابیس با تمام جزئیات آن

-ایجاد ساختمان ها جهت تطبیق نظریات اداره کننده دیتابیس و پالیسي هاي استفاده از دیتابیس

-حصول اطمینان از اجراء درست Application-Programها

-ارتباط با استفاده کنندگان عمومي دیتابیس جهت گرفتن نظریات شان

-تعریف و تطبیق Business-Ruleها و Constraintها

-دیزاین و توضیح اشکالي که جهت Backup و Reload نمودن دیتابیس ها استفاده مي شوند

-نظارت بر اجراي ديتابيس ها و اصلاح مشکلاتيکه در زمان اجراء بخش هاي ديتابيس بروز مي نمايند

3. End User ها

اين گروپ استفاده کننده گان شامل اشخاص عمومی بوده که ديتابيس را بصورت مسقيم توسط DBMS و يا بشکل Online از طريق شبکه ها و انترنت استفاده مينمايند. Interfaceها يا محيط های که از طريق صفحه ويب (Internet-Explorer) يک End-User، ديتابيس را استفاده مينمايد با پروگرام های Application تفاوت دارند. در اين محيط ها يک End-User نميتواند Application-Program را ديزاين نمايد، بلکه Applicationها از قبل ايجاد شده و بشکل Built-in موجود ميباشند. در چنين حالات End-Userها به دو شکل ميتوانند ديتابيس را استفاده نمايند. يکی با استفاده از دستورهای SQL-DML[5] و ديگر با استفاده از Iconها و مينوها برروی صفحه Web (Web Browser).

استفاده کننده گان نوع اول بايد اشخاص مسلکی ديتابيس و يا IT[6] باشند و استفاده کننده گان نوع دوم ميتوانند اشخاص عادی که مهارت مسلکی ديتابيس ندارند، باشند. واضحتر گفته شود اين که استفاده

[5] SQL-DML (SQL Data Manipulating Language)
[6] IT (Information Technology)

نمودن دیتابیس به کمك دکمه ها و مینوها نظر به نوشتن دستورها (Coding) به مراتب ساده تر میباشد.

یادداشت: بعضی از DBMSها طوری دیزاین شده اند که از جمله چهار قسمت در سیستم دیتابیس یعنی (Users ,Database Applications ,Database ,DBMS) ، سه قسمت آن متمایز میباشد. یعنی قسمت های 'Database Applications' و 'DBMS' یکی میباشند. مثال آن عبارت از Ms-Access است. در پروگرام اکسس تفکیک دستورهای Applicationها با دستورهای DBMS مشکل میباشد. تمام دستورها (Coding) که در جریان به راه انداختن یک کیوري اجراء میگردند، پنهان میباشند. بدین مفهوم که استفاده کننده نمیتواند دستورهای SQL را که برای اجراء Applicationها در اکسس استفاده شده اند، مستقیما ببیند. شکل(7) کود SQL را که در جریان براه انداختن کیوري شکل(5) توسط پروگرام اکسس ایجاد شده است، نشان میدهد.

```
SELECT CUSTOMER.CustomerName, COURSE.Course,
COURSE.CourseDate, COURSE.Fee, ENROLLMENT.AmountPaid

FROM CUSTOMER INNER JOIN
        (COURSE INNER JOIN ENROLLMENT
        ON COURSE.CourseNumber = ENROLLMENT.CourseNumber)
        ON CUSTOMER.CustomerNumber = ENROLLMENT.CustomerNumber

WHERE (((COURSE.course)
        Like "*" & [Enter part of course name:] & "*"))

ORDER BY CUSTOMER.CustomerName;
```

شکل(7) کود SQL تولید شده توسط اکسس

باید یادآور شد اینکه کود ذکر شده خیلی ها هم مشکل نیست. با دانستن SQL تفکیک این نوع جملات ساده میباشد.

مودل رابطه اي (The Relational-Model)

طوريكه قبلا نيز گفته شد ديتابيس ها نظر به ساختمان شان به چهار شكل ايجاد شده ميتوانند:

1.شجري (Hierarchical)

2.شبكه اي (Network)

3.رابطه اي (Relational)

4.شي گرا (Object-Oriented)

موضوع درس اين صنف اساساً نوع سوم (ديتابيس هاي رابطه اي) بوده، علاوتاً از سه مودل ديگر (اول، دوم و چهارم) نيز استفاده مي شود.

مودل رابطه اي ديتابيس ها براي بار اول توسط محقق امريكايي در شركت IBM داكتر ا. ف. كود (E. F. Codd) در دهه 1970 پيشنهاد گرديد. بعد از گذشت مدتي در سال 1985 همين شخص قانون 13 فقره اي (1+12) مودل رابطه اي را پيشكش نمود كه اين قانون تا به امروز عملي بوده و استفاده مي شود.

در پايين لست مكمل قانون **كود** همراه با نام هاي اصلي (Original) هر بند و توضيح آنها ذكر شده است:

0.اداره ديتابيس (Database Management)

در مودل رابطه اي يك ديتابيس بايد بصورت مكمل به اساس ساختمان رابطه اي آن استفاده شده و قابل دسترسي باشد.

1.معلومات (Information)

تمام معلومات بايد به شكل دو بعدي (Table) در حجره ها (Cells) به شكل محتواء هاي جداگانه در ديتابيس ثبت شده بتواند.

2.دستيابي تضمين شده به معلومات (Guaranteed Access)

تمام معلومات ذخيره شده در ديتابيس بدون كدام ابهام بايد قابل دسترسي باشد. اين دسترسي ميتواند با استفاده از نام جدول، كليد اوليه جدول و نام ستون جدول صورت پذيرد.

3.قبول كردن محتواء هاي ناشناخته (Systematic Null Value Support)

محتواء هاي ناشناخته (Null-Values) عبارت از معلوماتي اند كه نوعيت و محتواء آن هيچكدام معلوم نباشد. البته محتواء ناشناخته به ستون هاي اوليه (Primary-Keys)، ستون هاي اندكس (Index-Fields) و ستون هايكه (Not-Null) تعريف شده باشند وارد شده نمي توانند. اما

در ستون هاي عادي جداول در مودل رابطه اي بايد داخل شده بتوانند.

نوت: Null معادل صفر يا خانه خالي نيست!

4. دستيابي آني به ساختمان ديتابيس (Active, Online Relational Catalog)

ساختمان ديتابيس هاي رابطه اي بايد قابل دسترسي و فهرست يابي لحظوي (Online) باشند. يعني ديتابيس هاي رابطه اي بايد از طريق شبكه ها و در نهايت از طريق انترنت اكسس شده بتوانند. چون ساختمان هاي داخلي ديتابيس هاي رابطه اي (Metadata) نيز به شكل دو بعدي (Tabular) ذخيره مي باشند، بناً استفاده از آنها بصورت آني و توسط انترنت نيز ممكن بوده مي تواند.

5. استفاده از لسان فرعي (Comprehensive Data Sublanguage)

ديتابيس رابطه اي بايد حد اقل يك لسان جامع را حمايه نمايد كه در آن لسان دساتير تعريف ديتا، اداره ديتا و دقت ديتا شامل باشند. تمام ديتابيس هاي رابطه اي تجارتي لسان SQL را كه دساتير ياد شده در آن استفاده شده مي تواند، حمايه نموده و استفاده مي نمايند.

6.تازه کردن Viewها (View Updating)

معلومات شامل يك ديتابيس با استفاده از Viewها يا نماهاي كه به شكل منطقي ترتيب شده باشند قابل نمايش است. هر View در ديتابيس رابطه اي بايد قابليت تازه شدن (Updating) را داشته باشد.

7.داخل كردن، تازه كردن و پاك كردن معلومات به سطح بالايي (Set-Level Insert, Update, and Delete)

داخل كردن، تازه كردن و پاك كردن معلومات ديتابيس رابطه اي به چندين ستون و به چندين جدول بطور همزمان و بشكل گروپي بايد ممكن باشد.

8.استقلال فزيكي معلومات (Physical Data Independence)

در ديتابيس هاي رابطه اي استفاده كنندگان بايد از شكل فزيكي معلومات و شكل ذخيره سازي معلومات بكلي مجزا باشند. هر زمانيكه در شكل فزيكي ثبت معلومات، موقعيت معلومات ذخيره شده، و يا هم وسايل ذخيره معلومات كدام تغيري وارد شود در ميتود استفاده از معلومات بايد كدام تغيري وارد نشود.

9.استقلال منطقي معلومات (Logical Data Independence)

در ديتابيس هاي رابطه اي طريقه هاي نمايش معلومات با بميان آمدن تغيرات در اشكال منطقي (Logical) معلومات بايد تغير نه نمايند. طور مثال، تغيرات در

ساختمان جداول و نوع معلومات ستون ها باعث ایجاد تغیرات در نمایش معلومات نشود.

10. استقلال درستي ديتابيس (Integrity Independence)

لسان ديتابيس (مانند SQL) بايد درست بودن دستورهاي وارد شده به ديتابيس را حمايه نمايد و در زمان اجراء دساتير قوانين وضع شده توسط استفاده كننده را بصورت درست تطبيق نمايد. تا فعلاً اين قانون بصورت كامل در ديتابيس ها تطبيق نشده است. ولي حد اقل مسايل ذيل بر مبناي اين قانون در ديتابيس ها تطبيق شده اند:

-كليدهاي اوليه (Primary-Keys) جداول ديتا ناشناخته (Null Value) را قبول نمي نمايند.

-محتواء كليد خارجي (Freign-Key) در جدول پايين(Child) بايد ست فرعي كليد اوليه جدول بالايي (Parent) باشد.

11. استقلال توزیع دیتابیس (Distribution Independence)

پخش و توزیع یك دیتابیس در اضافه از یك موقعیت باید مشكلي ایجاد نه نماید. در صورت نصب قسمت هاي دیتابیس در چندین كمپیوتر با موقعیت هاي مختلف، استفاده كننده باید مشكلي در استفاده از دیتابیس، داخل كردن دیتا، تازه كردن دیتا، پاك كردن دیتا و خواندن دیتا نداشته باشد.

12. انحصار به لسان مخصوص (Nonsubversion)

وارد كردن هر نوع تغیرات در ساختمان دیتابیس باید تنها از طریق دساتیر لسان مخصوص دیتابیس (مانند SQL) امكان پذیر باشد. در این اواخر اكثر پروگرامهاي دیتابیس امكانات وارد كردن تغیرات در دیتابیس ها را به سطح عالي دارند. این تغیرات نیز بكمك لسان SQL صورت مي گیرند. دساتیر لسانهاي سطح پایین باید در ساختمان دیتابیس كدام تغیري وارد نه نمایند.

رابطه (Relation)

طوریکه قبلا تذکر داده شد، یک دیتابیس اساسا متشکل از جدول ها میباشد. برعلاوه دیتابیس منحصر به جدول ها نبوده بلکه دارای اجزاء اساسی دیگر نیز است]4، ص.3[. در مجموع تمام اجزاي ديتابيس به نحوي باهم در ارتباط بوده و یا متکی به هم اند.

یک Relation عبارت از دیتا ذخیره شده در یک جدول یا Table دو بعدی با مشخصات ذیل میباشد]4، ص. 26[:

-سطرها (Rows) ، دیتا در مورد Entity میداشته باشند.

-ستون ها (Columns) ، دیتا در مورد Attributeها یا مشخصه هاي Entity را نشان میدهند.

-خانه ها (Cells) ، یک جدول هرکدام یک قسمت یا جزء مشخص دیتا را دارا میباشند.

-هر ستون دارای یک نام مشخص میباشد.

-ترتیب قرار گرفتن ستون ها در جدول اختیاری میباشد.

-ترتیب قرار گرفتن سطرها در جدول اختیاری میباشد.

-د و سطر دارای عین محتویات بوده نمیتوانند.

هر سطر، دیتا در مورد یک Entity یا قسمتی از Entity را دارا است. محتویات ستون در یک جدول، نمایانگر یک مشخصه از Entity میباشد. مثلا در یک Relation یا جدول بنام 'EMPLOYEE' هر سطر معلومات در مورد یک فرد مشخص را نشان میدهد و محتواء هر ستون یک صفت یا مشخصه همان فرد را نشان میدهد، مانند Name، Phone و یا Email-Address.

همچنان برای اینکه یک Relation یا جدول بوجود آید، خانه های جدول هرکدام باید یک جزء مشخص از دیتا (معلومات) را داشته باشند. تکرار عین دیتا در یک Cell مجاز نمیباشد و دیتا در یک ستون باید از یک نوع مشخص باشد. یعنی در ستون Email تمام دیتا باید در فارمت آدرس الکترونیکی (Email) باشد. هر ستون باید دارای یک نام باشد و نام متذکره برای ستون دیگر تکرار نشده باشد. ترتیب در قرار گرفتن ستون ها و سطرها مشخص نمیباشد، یعنی ستون Name میتواند در اول، وسط و یا اخیر ستونها واقع شود. بهمین ترتیب یک سطر میتواند در هر موقعیت جا گیرد. همچنان هیچ دو سطری دارای عین دیتا بوده نمیتوانند.

تفکیک Relation و NonRelation

شکل (8) یک جدول بنام EMPLOYEE را نشان میدهد. اگر شرایط یک Relation از درس قبلی بالای این جدول تطبیق شوند، طور ذیل نتیجه گیری میشود:

هر سطر، معلومات در مورد یک شخص را دارا است و هر ستون یک مشخصه را در مورد هر شخص از جدول دارا میباشد. بنا دو شرط اول در مورد بودن Relation درست اند. هر خانه دارای یک جزء دیتا است و تمام دیتا در یک ستون از یک نوع مشخص میباشد. نام های ستون ها تکرار نمیباشند و موقعیت ستون ها و سطرها، بدون اینکه در دیتا کدام تغیری وارد شود، تغییر شده میتوانند. بالاخره هیچ سطری مشابه به سطر دیگر در جدول متذکره نمیباشد. به این ترتیب، تمام شرایط ذکر شده برای یک Relation در این جدول قابل تطبیق بوده و گفته میشود که جدول متذکره یک رابطه Relation است.

FirstName	LastName	Department	Email	Phone
Jerry	Johnson	Accounting	JJ@somewhere.com	236-9987
Mary	Abernathy	Finance	MA@somewhere.com	444-8898
Liz	Smathers	Finance	LS@somewhere.com	777-0098
Tom	Caruthers	Accounting	TC@somewhere.com	236-9987
Tom	Jackson	Production	TJ@somewhere.com	444-9980
Eleanore	Caldera	Legal	EC@somewhere.com	767-0900
Richard	Bandalone	Legal	RB@somewhere.com	767-0900

شکل (8) جدول EMPLOYEE

شکل (9-الف) و شکل (9-ب) دو جدول را نشان میدهند که در حالت بودن Relation یا جدول نمیباشند، یعنی شرایط ذکر شده در مورد Relation را کاملا صدق نمیکنند. جدول (9-الف) بخاطر تعدد دیتا در ستون Phone یک Relation نیست. 'Tom' دارای سه نمبر تلفون و 'Richard' دارای دو نمبر تلفون است. پس شرط موجودیت یک جزء دیتا در یک Cell تطبیق نشده و Relation نمیباشد.

بهمین ترتیب جدول (9-ب) به سبب صدق نه نمودن دو شرط، Relation نیست. اول اینکه اگر موقعیت سطرها تبدیل شود، امکان اشتباه در قسمت 'Tom' و 'Eleanore' وجود دارد، بخاطریکه 'Fax Number' و 'Home Phone' آنها نیز درج جدول میباشد. همچنان دیتا در ستون EmailAddress از یک نوع نمیباشد.

TABLE 1					
EmployeeNumber	**FirstName**	**LastName**	**Department**	**Email**	**Phone**
100	Jerry	Johnson	Accounting	JJ@somewhere.com	236-9987
200	Mary	Abernathy	Finance	MA@somewhere.com	444-8898
300	Liz	Smathers	Finance	LS@somewhere.com	777-0098
400	Tom	Caruthers	Accounting	TC@somewhere.com	236-0000, 236-0991, 236-0992
500	Tom	Jackson	Production	TJ@somewhere.com	444-9980
600	Eleanore	Caldera	Legal	EC@somewhere.com	767-0900
700	Richard	Bandalone	Legal	RB@somewhere.com	767-0900, 767-0011

شکل (9- الف) جدول با شرایط NonRelation

TABLE 2					
EmployeeNumber	**FirstName**	**LastName**	**Department**	**Email**	**Phone**
100	Jerry	Johnson	Accounting	JJ@somewhere.com	236-9987
200	Mary	Abernathy	Finance	MA@somewhere.com	444-8898
300	Liz	Smathers	Finance	LS@somewhere.com	777-0098
400	Tom	Caruthers	Accounting	TC@somewhere.com	236-9987
				Fax:	236-9987
				Home:	555-7151
500	Tom	Jackson	Production	TJ@somewhere.com	444-9980
600	Eleanore	Caldera	Legal	EC@somewhere.com	767-0900
				Fax:	236-9987
				Home:	555-7171
700	Richard	Bandalone	Legal	RB@somewhere.com	767-0900

شکل (9-ب) جدول با شرایط NonRelation

ناگفته نباید گذاشت اینکه طول دیتا داخل شده در یک Cell مشخص نمیباشد، یعنی یک جزء دیتا امکان دارد با طول زیاد در یک Cell جابجا شده باشد و شرایط بودن Relation نیز درست باشد. طور مثال شکل (10) جدول شکل (8)

را با اضافه شدن یک ستون دیگر نشان میدهد که این جدول نیز یک Relation است.

TABLE 3						
Employee Number	**FirstName**	**LastName**	**Department**	**Email**	**Phone**	**Comment**
100	Jerry	Johnson	Accounting	JJ@somewhere.com	236-9987	Joined the Accounting Department in March after completing his MBA.
200	Mary	Abernathy	Finance	MA@somewhere.com	444-8898	
300	Liz	Smathers	Finance	LS@somewhere.com	777-0098	
400	Tom	Caruthers	Accounting	TC@somewhere.com	236-0000	
500	Tom	Jackson	Production	TJ@somewhere.com	444-9980	
600	Eleanore	Caldera	Legal	EC@somewhere.com	767-0900	
700	Richard	Bandalone	Legal	RB@somewhere.com	767-0900	Is a full time consultant to legal on a retainer basis.

شکل (10) Relation با داشتن دیتا طویل در یک Cell

یادداشت: در ساحه دیتابیس اصطلاحات Table و Relation معمولا به عین منظور استفاده میشوند و نشان دهنده جدولی با شرایط ذکر شده درس قبلی در صفحه "39" میباشد.

بعضی اوقات اصطلاح فایل به عوض Table، خصوصا در ساحه تجارتی دیتابیس استفاده میشود، در چنین حالتی اصطلاح Record به عوض سطر(Row) و اصطلاح Field به عوض ستون(Column) استفاده میشود. بیجا نخواهد بود تا یادآوری شود اینکه بعضا بعوض Table، اصطلاح Relation بکار برده شده، بعوض سطر، اصطلاح Tuple و بعوض ستون اصطلاح Attribute استفاده میشود.

این سه Set که در شکل(11) نیز نشان داده شده اند، استفاده کننده را متردد یا Confuse مینماید. مهمتر اینکه بعضا این ست ها نیز اشتباها با هم یاد میشوند، مانند Rowها و Fieldها...

Table	Row	Column
File	Record	Field
Relation	Tuple	Attribute

شکل(11) ست های معادل در دیتابیس

بعضا در جدول های بزرگ چنین اتفاق می افتد که تکرار سطرها صورت گیرد. بآنهم جدول متذکره یک Relation به شمار میآید. یعنی پیدا نمودن سطرهای مشابه در بعضی حالات مهم نبوده و باعث نبودن Relation نمیشود.

انواع کلیدها (Types of Keys)

یک کلید (Key) عبارت از یک یا اضافه از یک ستون در یک جدول استکه معمولا برای شناسایی یک سطر یا ریکورد استفاده میشود]4، ص. 28[. این نوع ستون میتواند دیتا تکراری و یا هم غیرتکراری داشته باشد. طور مثال در جدول شکل(8) ستون EmployeeNumber یک کلید یا Key است که از تکرار دیتا در آن جلوگیری شده است. اگر توسط کیوري دیتا طوري خواسته شود که در این ستون عدد "200" باشد، تنها و تنها یک سطر نشان داده خواهد شد، در حالیکه ستون Department، عبارت از کلیدی میباشد که اجازه ذخیره دیتا تکراری را نیز دارد. این بخاطری ستون Key میباشد که توسط آن یک سطر معین میشود، اما شرط یکتا بودن در آن مطرح نیست و چندین شخصیکه در عین دیپارتمنت وظیفه دارند، دارای عین دیتا در این ستون بوده میتوانند. با براه انداختن یک کیوري، که در آن تمام اشخاص مربوط دیپارتمنت Accounting خواسته شده باشند، اضافه از یک سطر به نمایش در می آیند. (در اینجا دو سطر)

اگر به جدول شکل(8) دقت شود، دیده مي شود که ستون های EmployeeNumber، Email و LastName هرکدام به تنهایي تفکیک دهنده اشخاص موجود در جدول شده میتوانند. پس برای ثبوت این که این ستون ها، ستون های کلیدی اند و یا خیر، دیزاینرهای دیتابیس باید به مشوره استفاده کنندگان دیتابیس عمل نمایند. طور مثال اگر ستون

LastName در یک دیتابیس دارای دیتای تکراری باشد و دیتابیس طوری دیزاین شده باشد که این ستون دیتا تکراری را قبول نه نماید، دیتابیس ناقص بوده، مشکلاتی را در زمان داخل کردن دیتا ببار خواهد آورد.

کلیدهای ترکیبی (Composite-Keys)

فرض شود استفاده کنندگان پیشنهاد مینمایند این که، دیتا در ستون LastName امکان دارد تکرار شود، بنا دیزاینر دیتابیس باید از کلید ترکیبي یا (Composite-Keys) استفاده نماید [4، ص. 28] . کلید ترکیبي از ترکیب دو ستون (در اینجا LastName و Department) ایجاد میشود. این شرط در حالتی صدق مینماید که هیچ دو شخصی با 'LastName' یکسان در یک دیپارتمنت کار نه نمایند. اگر دیتابیس وسیعتر شود و دوباره امکان تکرار دیتا بوجود آید، دیزاینر میتواند سه ستون را بطور مشترک بشکل کلید ترکیبي تعیین نماید. در مثال بالا امکان دارد ستون FirstName نیز به کلید ترکیبی اضافه شود. در اینصورت جدول متذکره دارای یک کلید ترکیبی متشکل از سه ستون (FirstName, LastName, Department) خواهد گردید تا از تکرار دیتا در آن جلوگیری بعمل آید.

کلیدهای اولیه و کاندیدی (Primary and Candidate-Keys)

طور مثال در جدول شکل(8) ستون EmployeeNumber و ستون Email هرکدام بصورت مستقل میتوانند دیتا غیرتکراری داشته باشند. بعین ترتیب، ترکیب ستون های LastName، FirstName و Department نیز میتواند دیتا غیرتکراری داشته باشد. بنا دیزاینر دیتابیس میتواند یکی از این کلیدها را انتخاب نموده و Primary-Key تنظیم نماید. کلیدهای غیرتکراری دیگر که کلید اولیه تنظیم نشده باشند بنام Candidate-Keys یاد میشوند [3، ص. 269] [4، ص. 28].

Primary-Keyها تنها بخاطر داشتن دیتا غیرتکراری مهم نبوده بلکه بخاطر نمایش یک جدول در Relationship نیز داراي اهمیت اند. وقتی Relationship بین دو جدول ایجاد میشود، ستون یا ستون های Primary-Key یکی از جداول در دیگر آن جابجا میشود و یا Primary-Keyهای هردو جدول در یک جدول جدیدا ایجاد شده سوم (نظر به نوع Relationship) جابجا شده و باعث میشود تا سطرهای جداول جهت تمثیل Relationship نشان داده شوند.

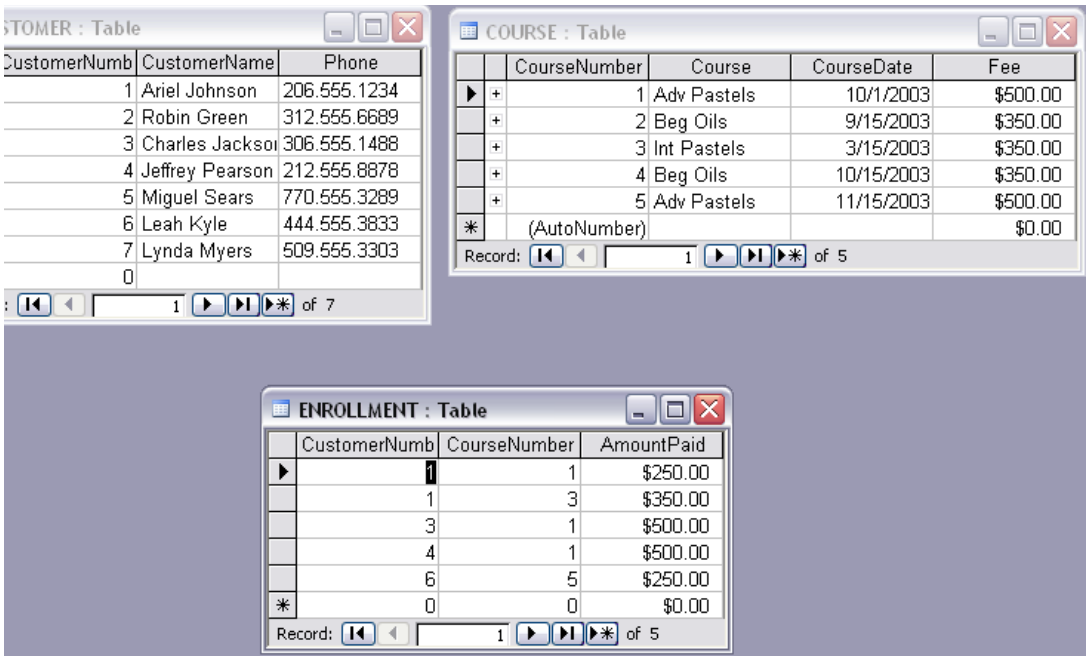
برعلاوه تولید کننده گان DBMSها محتواء Primary-Key را به اهداف مختلف دیگر نیز استفاده مینمایند. مثلا بخاطر تنظیم یا Sort نمودن محتواء Relation در داخل

دیتابیس، تهیه Indexها و بعضی ساختمان های دیگر بخاطر دستیابی فوری به دیتا و غیره.

کلیدهای خارجي و قاعده RIC

(Freign-Keys and Referential Integrity Constraint)

طوریکه قبلا نیز گفته شد، برای نشان دادن یک Relationship، کاپي Primary-Key جدول اول در جدول دوم جابجا میشود. همین کلید جابجا شده در جدول دوم بنام Freign-Key یاد میشود]3، ص. 272[ ]4، ص. 29[. به عباره دیگر جهت نشان دادن یك Relationship، کاپي Primary-Key جدول بالایي یا (جدول Parent) در جدول پایاني یا (جدول Child) جابجا مي شود و بنام کلید خارجي یا (Foreign-Key) یاد مي شود. طور مثال در شکل(12) ستون CustomerNumb از جدول CUSTOMER، که Primary-Key است، در جدول ENROLLMENT جابجا شده و یک Foreign-Key میباشد. بهمین ترتیب ستون CourseNumber از جدول COURSE، که Primary-Key است، نیز در جدول ENTOLLEMENT جابجا شده و یک Freign-Key را تشکیل داده است.



شکل (12) سه جدول با داشتن Relationship ها بین هم

دو جدول پائین فرض شوند:

EMPLOYEE (EmployeeNumber, FirstName, LastName, Department, Email, Phone)

و

DEPARTMENT (DeptName, BudgetCode, OfficeNumber)

در این یادداشت نام جدول در ابتدا سطر نوشته شده است. در داخل قوسها نام تمام ستون های موجود در جدول لست

گردیده اند. (مودل رابطه اي)*

فرض شود EmployeeNumber و DeptName عبارت از Primary-Keyهای جداول EMPLOYEE و DEPARTMENT باشند. پس جداول متذکره به این شکل نوشته شده میتوانند:

EMPLOYEE (EmployeeNumber, FirstName, LastName, Department, Email, Phone)

DEPARTMENT (DeptName, BudgetCode, OfficeNumber)

حال فرض شود اینکه، اگر عناصر ستون Department در جدول EMPLOYEE با عناصر ستون DeptName در جدول DEPARTMENT یکی باشند. بنا ستون Department در جدول EMPLOYEE یک Freign-Key به جدول DEPARTMENT میباشد. چون Freign-Key به شکل Italic نشان داده میشود، پس

EMPLOYEE (EmployeeNumber, FirstName, LastName, *Department*, Email, Phone)

DEPARTMENT (DeptName, BudgetCode, OfficeNumber)

در این نوت:
نام جدول تماما به حروف بزرگ نوشته میشود.
حرف اول نام ستون بزرگ میباشد (حرف اول تمام کلمات مشتمل در نام ستون، بزرگ میباشد، مانند FirstName ...
Primary-Key بصورت Underline نشان داده میشود.
Freign-Key به شکل Italic نشان داده میشود.
درصورت بودن هردو شرط PK و FK بالاي يك كليد، هردو بصورت همزمان استفاده شده اند.

استفاده از عین نام برای Primary-Key و Freign-Key (Primary-Key) جدول اول که به شکل Freign-Key در جدول دوم استفاده میشود) امکان دارد ولي شرط نمیباشد. شرط مهم داشتن عین گروپ عناصریا(دیتا) و عین Domain یا نوع دیتا (Data-Type) میباشد. عناصر ستون Department حتما باید در ستون DeptName شامل باشند.

همیشه باید عناصر ستون (هاي) Primary-Key در برگیرنده عناصر ستون (هاي) Freign-Key باشند. Freign-Key ست فرعی Primary-Key است. بعباره دیگر هر عنصر ستون Freign-Key در جدول (Child) باید در ستون Primary-Key جدول بالایي (Parent) موجود باشد. در مثال قبلی هر عنصر ستون Department جدول EMPLOYEE باید در ستون DeptName در جدول DEPARTMENT موجود باشد. پس چنین گفته میشود.

عناصر Department در EMPLOYEE باید شامل DeptName در DEPARTMENT باشند.

این قاعده بنام 'Referential Integrity Constraint (RIC)' یاد میشود. با دیدن Freign-Key همیشه باید قاعده بالا در نظر گرفته شود.

ساختمان جدول EQUIPMENT در شکل(13-الف) عبارت است از:

EQUIPMENT (SerialNumber, Type, AcquisitionCost)

فرضا این وسیله (Equipment) به اشخاص لست شده در شکل(8) منسوب شود، اگر Primary-Key جدول EMPLOYEE عبارت از EmployeeNumber باشد، پس کاپی این ستون به شکل Freign-Key به جدول EQUIPMENT اضافه شده و Relationship ایجاد میشود. شکل(13-ب) جدول EQUIPMENT را با Freign-Key اضافه شده به آن نشان میدهد. این جدول نشان میدهد که چطور سه وسیله اول لست به یک شخص با نمبر "100" ارتباط گرفته است، دو وسیله بعدی به شخصی با نمبر "300" ارتباط دارد، ...

EQUIPMENT

SerialNumber	Type	AcquisitionCost
1000	Computer	$2,600.00
1200	Printer	$275.00
1300	Monitor	$350.00
1400	Computer	$1,700.00
1500	Monitor	$400.00
1600	Scanner	$350.00
1700	Computer	$3,100.00
1800	Monitor	$350.00

شکل(13-الف) جدول EQUIPMENT

EQUIPMENT			
SerialNumber	**Type**	**AcquisitionCost**	**EmployeeNumber**
1000	Computer	$2,600.00	100
1200	Printer	$275.00	100
1300	Monitor	$350.00	100
1400	Computer	$1,700.00	300
1500	Monitor	$400.00	300
1600	Scanner	$350.00	100
1700	Computer	$3,100.00	200
1800	Monitor	$350.00	200

شکل (13-ب) جدول EQUIPMENT با ستون EmployeeNumber به حیث Freign-Key

ساختمان متني اين جدول طور ذيل نشان داده ميشود:

EQUIPMENT (SerialNumber, Type, AcquisitionCost, *EmployeeNumber*)

قاعده RIC برای این جداول عبارت است از:

- عناصر ستون EmployeeNumber جدول EQUIPMENT باید در ستون EmployeeNumber جدول EMPLOYEE موجود باشند.

کلیدهای جانشین (Surrogate-Keys)

جهت دانستن این نوع کلیدها به مثال ذیل توجه شود:

یک شرکت وظیفه فروش بته های نباتات مختلف و نصب آنها برای اشخاص مختلف را دارد. در Relation یا جدول باید آدرس خریدار، مشخصات بته، تاریخ و مجموع ساعات کار توضیح داده شود. سه جدول ذیل وجود دارند:

PROPERTY (Street, City, State, Zip)

PLANT (ItemNumber, VarietyName, Price)

SERVICE (InvoiceNumber, Date, TotalHours)

تفصیل این جداول در شکل(14) نشان داده شده است.

PROPERTY

Street	City	State	Zip
123 East Elm	Riverside	CA	90800-4987
82334 - 188th Street	Oakland	CA	91200-9810
477 North Greenbrier Circle	Riverside	CA	90822-3328
One South Highrent Estates Parkway	Rolling Hills	CA	93455-0080

SERVICE

InvoiceNumber	Date	TotalHours
500	5/5/2002	17
550	5/6/2002	6
600	5/9/2002	12
650	5/11/2002	8

PLANT

ItemNumber	VarietyName	Price
1000	Carpet Rose, White	$9.95
1100	Carpet Rose, Yellow	$11.50
1200	Lilac, 5 gal, purple	$57.55
1300	Lilac, 2 gal, purple	$27.50
1400	Hybrid Tea, Rose	$22.45

شکل (14) نمونه دیتا برای جداول PLANT، PROPERTY و SERVICE

هر نبات به شخصی از جدول PROPERTY فروخته میشود. پس یک Relationship از جدول PROPERTY به جدول PLANT ایجاد میشود، یعنی کاپی Primary-Key جدول PROPERTY در جدول PLANT اضافه میشود. بهمین ترتیب هر نبات به یک تاریخ معین به فروش رسیده و در زمان معین نصب میشود. پس ضرورت به ایجاد یک Relationship بین جداول PROPERTY و SERVICE نیز میباشد، بنا کاپی Primary-Key جدول PROPERTY در جدول SERVICE هم جابجا میشود. نتیجه این Relationshipها طور ذیل میشود.

PROPERTY (Street, City, State, Zip)

PLANT (ItemNumber, VarietyName, Price, *Street, City, State, Zip*)

SERVICE (InvoiceNumber, Date, TotalHours, *Street, City, State, Zip*)

قاعده RIC برای این جداول عبارت است از:

- عناصر ستون های (Street, City, State, Zip) در جدول PLANT باید در ستون های (Street, City, State, Zip) در جدول PROPERTY موجود باشند.

- عناصر ستون های (Street, City, State, Zip) در جدول SERVICE باید در ستونهای (Street, City, State, Zip) در جدول PROPERTY موجود باشند.

در صورت ایجاد Relationship به این شکل، دو مشکل عمده بروز خواهد کرد. اول اینکه یک مقدار زیاد دیتا به شکل تکراری نوشته خواهد شد. طول کلید ترکیبی (Street, City, State, Zip) شاید اضافه از 100 کرکتر گردد، که همین 100 کرکتر به هر سطر جدول های PLANT و SERVICE، بخاطر علاوه شدن Freign-Key اضافه میشود. ثانیا یک استفاده کننده بخاطر داخل نمودن دیتا معمولی، مجبور خواهد بود تا تمام دیتا در مورد آدرس شخص، اعم از نام سرک، نام شهر، نام ولایت و کود را داشته باشد.

راه حل ایجاد یک کلید جانشین یا (Surrogate-Key) برای جدول PROPERTY میباشد. این نوع کلید به شکل Primary-Key جدول متذکره استفاده شده [4، ص. 32] ، معمولا دارای محتواء رقمي میباشد و نماینده گی از تمام ستون های که برای کلید ترکیبی (Composite-Key) استفاده شده اند، را مینماید. کلید های جانشین (Surrogate-Keys) برای استفاده کننده گان کدام مفهومی ندارند و در حالت

نورمال در دیتابیس به شکل مخفی میباشند، یعنی استفاده کننده در ایجاد و استفاده فورم ها، کیوري ها و راپورها (Reports) کلیدهای Surrogate را نمیبیند و ضرورتی برای دیدن آنها نیز نمیباشد.

در مثال بالا از PropertyID بحیث 'Surrogate-Key' استفاده شده است. با استفاده از این نوع کلید شکل جداول یاد شده طور ذیل تغییر می نماید.

PROPERTY (PropertyID, Street, City, State, Zip)

PLANT (ItemNumber, VarietyName, Price, *PropertyID*)

SERVICE (InvoiceNumber, Date, TotalHours, *PropertyID*)

قاعده RIC برای این جداول عبارت است از:

- عناصر ستون PropertyID در جدول PLANT باید در ستون PropertyID در جدول PROPERTY موجود باشند.

- عناصر ستون PropertyID در جدول SERVICE باید در ستون PropertyID در جدول PROPERTY موجود باشند.

این دیزاین از تکرار و اضافه نویسی دیتا تا اندازه اي زیادی جلوگیری مینماید و بعوض نوشتن چهار عنصر دیتا، صرف یک عنصر، آنهم رقمي، نوشته میشود.

بسیاری از تولید کننده گان DBMSها چنین مشکلاتی را حل ساخته و بصورت اتوماتیک کلید جانشین (Surrogate-Key) را برای جداول دیزاین و استفاده مینمایند. طور مثال، بعد از ایجاد یک جدول جدید، پروگرام اکسس یک ستون را در فارمت AutoNumber به صورت اتومات به جدول اضافه مینماید.

شکل(15) جداول شکل(14) را معه اي استفاده از Surrogate-Key و ایجاد Relationship بین آنها نشان میدهد.

PROPERTY

PropertyID	Street	City	State	Zip
100	123 East Elm	Riverside	CA	90800-4987
110	82334 - 188th Street	Oakland	CA	91200-9810
120	477 North Greenbriar Circle	Riverside	CA	90822-3328
130	One South Highrent Estates Parkway	Rolling Hills	CA	93455-0080

PLANT

ItemNumber	VarietyName	Price	PropertyID
1000	Carpet Rose, White	$9.95	120
1100	Carpet Rose, Yellow	$11.50	130
1200	Lilac, 5 gal, purple	$57.55	120
1300	Lilac, 2 gal, purple	$27.50	130
1400	Hybrid Tea, Rose	$22.45	100

SERVICE

InvoiceNumber	Date	TotalHours	PropertyID
500	5/5/2002	17	100
550	5/6/2002	6	130
600	5/9/2002	12	120
650	5/11/2002	8	100

شکل (15) جداول PLANT، PROPERTY

و SERVICE معه استفاده از Surrogate-Key

Functional-Dependency and Normalization

این قسمت بعضی مفاهیم استفاده شده برای دیزاین Relational-Databaseها را بصورت ابتدایی معرفی نموده، قابلیت استفاده آنها را نشان میدهد.

Functional-Dependency

برای دانستن مفهوم 'Functional-Dependencies' ، مثال های ساده الجبری زیر مرور شوند]4، ص. 35[:

> -فرضا در یک فروشگاه، قطی های کلچه فی دانه مبلغ 10 افغانی قیمت دارند، قیمت اضافه از یک قطی بطور ساده چنین معلوم میشود.

CookieCost = NumberOfBoxes X 10

اگر به سطر بالا توجه شود، Relationship بین عناصر ستون CookieCost و عناصر ستون NumberOfBoxes عبارت از مربوط بودن عنصر CookieCost به عنصر NumberOfBoxes میباشد. یعنی عنصر اول تابع عنصر دوم است (بصورت Functionally). در اساسات دیتابیس این جمله بشکل ذیل ارائیه میشود:

NumberOfBoxes → CookieCost

جمله بالا طور ذیل نیز گفته شده میتواند:

عنصر NumberOfBoxes مشخص کننده (Determinant) عنصر CookieCost است. متحولیکه در طرف چپ واقع شده است، یعنی NumberOfBoxes ، بنام Determinant عنصر طرف راست (CoodieCost) یاد میشود.

-در مثال دیگر فکر شود که یک خریطه با اشیاء مختلف با رنگ های سرخ، آبی و زرد با شخص اول موجود است. همچنان فرض شود که اشیای با رنگ سرخ دارای وزن 5 کیلوگرام، اشیای با رنگ آبی دارای وزن 3 کیلوگرام و اشیای با رنگ زرد دارای وزن 7 کیلوگرام اند. شخص دوم یکی از این اشیا را بدون اینکه شخص اول ببیند، برداشته و صرف رنگ آن را میگوید، با دانستن نوعیت رنگ شخص اول بصورت فوری وزن شی متذکره را تشخیص میدهد. بنا رنگ شی تعیین کننده یا (Determinant) وزن شی است. بعباره دیگر، وزن شی Functionally مربوط به رنگ شی میباشد. همین جمله طور ذیل واضح شده میتواند.

ObjectColor → Weight

بهمین ترتیب اگر گفته شود که اشیاء سرخ عبارت از توپ ها و اشیاء آبی و زرد عبارت از شش ضلعی ها اند، جمله ذیل نیز درست میباشد:

ObjectColor → Shape

چون رنگ شی Determinant وزن و شکل یا Shape هر دو میباشد، پس

ObjectColor → (Weight, Shape)

همین Relation طور ذیل نیز نشان داده شده میتواند:

ObjectColor	Weight	Shape
Red	5	Ball
Blue	3	Cube
Yellow	7	Cube

جدول بالا تمام شرایط ذکر شده بودن یک Relation را تکمیل نموده و یک Relation میباشد. ستون ObjectColor عبارت از Primary-Key است. پس چنین نوشته شده میتواند:

OBJECT (ObjectColor, Weight, Shape)

بعین شکل، Relation مثال قبلی در ذیل در نظر گرفته شود:

PLANT (ItemNumber, VarietyName, Price, *PropertyID*)

'Functional-Dependency' جدول بالا عبارت است از:

ItemNumber → (VarietyName, Price, PropertyID)

بنا بر جملات بالا، گفته میشود اینکه Primary-Keyها و Candidate-Keyها هر کدام Determinant جداول مربوطه میباشند.

-به مثال ذکر شده در شکل(8) توجه شود:

EMPLOYEE (EmployeeNumber, FirstName, LastName,

Department, Email, Phone)

طوریکه قبلا نیز گفته شده است، این Relation دارای سه Candidate-Key است، اول EmployeeNumber، دوم Email و سوم کلید ترکیبی (FirstName, LastName, Department). بنا به توضیحات در مورد Functional-Dependency چنین نوشته شده میتواند:

EmployeeNumber → (FirstName, LastName,

Department, Email, Phone)

و یا

Email → (EmployeeNumber, FirstName, LastName, Department, Phone)

و یا هم

(FirstName, LastName, Department) → (EmployeeNumber, Email, Phone)

Transitive-Dependency

Transitive-Dependency وقتی بمیان می آید که یک یا چندین مشخصه در یک جدول توسط یکی از مشخصه های غیر کلیدی در همان جدول مشخص یا Determine شده باشند] 2 ص. 289[.

به مثال ذیل توجه شود:

STUDENT (Stu_ID, Stu_Name, Dorm, Dorm_Location)

در مثال بالا فیلد Stu_ID عبارت از Primary-Key بوده و دیترمنانت متباقی فیلدها نیز میباشد. فیلد Dorm ، که یک مشخصه غیر کلیدی است و توسط Stu_ID دیترماین شده است، خود فیلد Dorm_Location را دیترماین یا مشخص میکند. بنا چنین نوشته شده میتواند:

Stu_ID → Dorm → Dorm_Location

جمله بالا یک Transitive-Dependency را در جدول STUDENT نشان میدهد.

مثال دیگری در نظر گرفته شود:

VEHICLE (VIN, Make, Model, Year, NID, Owner)

در مثال بالا فیلد VIN عبارت از Primary-Key بوده و متباقی فیلدهای جدول را نیز دیترماین مینماید. فیلد NID ، که یک مشخصه غیر کلیدی است و توسط VIN دیترماین شده است، خود دیترمنانت یا مشخص کننده فیلد Owner میباشد. بنا چنین نوشته شده میتواند:

VIN → NID → Owner

این جمله نیز یک Transitive-Dependency را در جدول VEHICLE نشان میدهد.

در یک جدول احتمال دارد چندین Transitive-Dependency وجود داشته باشد و علاوه از دو Determinant نیز وجود داشته میتواند.

نارمل سازي (Normalization)

نارمل سازي در ديتابيس يک عنوان بسيار وسيع و در عين حال مهم است.

تعريف: نارمل سازي عبارت از عمليه اي است که توسط آن جدولها تحليل و تجزيه شده و مشکلاتيرا که در زمان تغير آوردن و Update شدن ديتا داخل جدول، پاک کردن ديتا از جدول و داخل کردن ديتا به جدول بميان می آيند رفع مينمايد. همچنان نارمل سازي از تکرار ديتا در موقعيت های مختلف نيز جلوگيری مينمايد. توسط عمليه نارمل سازي مشخصه ها به Entityها تنظيم ميشوند]2، ص. 282[.

نارمل سازي دارای اشکال مختلف بوده و در مراحل جداگانه عمليه های آن بالای يک جدول تطبيق شده و جدول متذکره به کامل ترين و واضح ترين شکل خود تبديل ميشود.

- طوريکه گفته شد، بعضا يک جدول با وجود تکميل نمودن تمام شرايط بودن يک Relation*، مشکلاتی در قسمت Update شدن ديتا ميداشته باشد. برای وضاحت بيشتر موضوع، به جدول ADVISER-LIST توجه شود.

DVISER-LIST (AdviserID, AdviserName, Department, Phone, Office, StudentNum, StudentName)

* شرايط بودن Relation يک جدول در صفحهء "39" اين نوت درج است.

اگر خواسته شود تا Primary-Key این جدول مشخص شود، نظر به تعریف Primary-Key، گفته میشود: Primary-Key عبارت از ستونی است که دیترمنانت (Determinant) ستون های دیگر این جدول باشد. چنین ستونی عبارت از StudentNum است. بدین مفهوم که StudentNum دیترمنانت تمام مشخصه های این جدول است. یعنی

StudentNum → (AdviserID, AdviserName, Department, Phone, Office, StudentName)

پس Relation بالا چنین نوشته شده میتواند:

ADVISER-LIST (AdviserID, AdviserName, Department, Phone, Office, StudentNum, StudentName)

جدول بالا با وجود پوره نمودن شرایط بودن Relation، مشکلاتی در قسمت Update شدن دیتا دارد. طور مثال معلومات در مورد Adviser (مشاور) بطور تکراری برای هر Advisee (مشوره گیرنده) در جدول داخل میگردد. بدین مفهوم، اگر کدام تغیری در دیتا یک Adviser می آید، این تغیر برای هر Advisee مربوط همان Adviser در جدول باید وارد گردد. بعباره اي دیگر، جهت وارد نمودن یک تغیر، چندین سطر باید تغیر داده شوند. اگر یک Adviser بیست Advisee داشته باشد، و شماره تلفون Adviser تغیر نماید، در اینصورت شماره تلفون Adviser باید در بیست سطر یا ریکورد تغیر داده شود.

- مشکل دیگری که در این جدول موجود بوده میتواند، در قسمت پاک شدن یا Delete شدن دیتا است. طور مثال، اگر یک Adviser، تنها یک محصل را بحیث Advisee داشته باشد و ضرورت افتد تا Record محصل متذکره Delete گردد، با Delete شدن این Record برعلاوه اي پاک شدن دیتا در مورد محصل، دیتا در مورد Adviser نیز از Relation پاک میشود. یعنی اکثرا با از بین بردن دیتا در مورد یک عنوان، دیتا در مورد عناوین دیگر نیز از بین میرود.

- به عین ترتیب در زمان داخل کردن دیتا به جدول نیز مشکلاتی خواهد بود. و برای Recordهای مختلف عین گروپ دیتا ضرورت خواهد بود تا داخل گردد.

نوت: به حیث قاعده عمومی هر جدول در داخل دیتابیس باید دیتا در مورد یک عنوان یا یک Topic داشته باشد!

اگر به Relation بالا دقت شود، بر علاوه Functional-Dependency اول، یک Functional-Dependency دیگر نیز وجود دارد و آن عبارت است از:

AdviserID → (AdviserName, Department, Phone, Office)

با توجه به این حقیقت، مشکلات ذکر شده بالا رفع خواهد گردید. پس گفته میشود اینکه، جدول ADVISER-LIST بصورت درست دیزاین نشده است. بخاطریکه این جدول برعلاوه StudentNum دارای یک Determinant دیگر نیز است و

این فیلد یک Functional-Dependency را نشان میدهد در حالیکه فیلد دیترمنانتنه Primary-Key ونه هم Candidate-Key در جدول است.

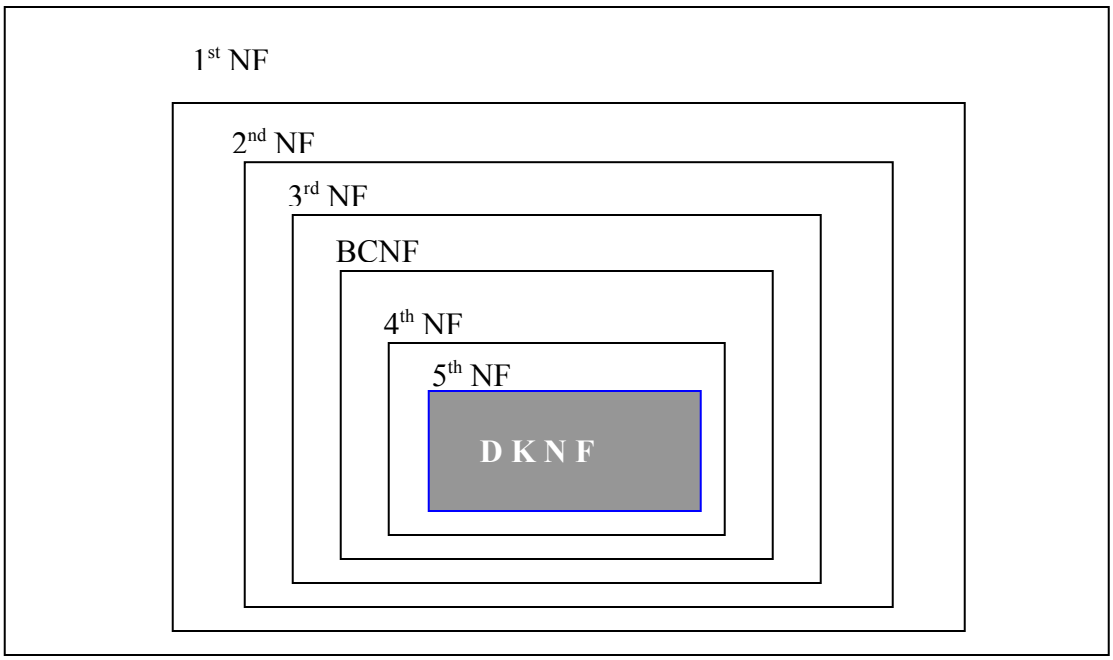
بنا به گفته های بالا، یک Relational-Design طور ذیل خلاصه میگردد]4، ص. 37[.

- برای دیزاین بهتر یک جدول، هر Determinant باید یک Candidate-Key باشد.

- هر جدولیکه شرط بالا در آن صدق نه نماید، باید به دو یا اضافه از دو جدول تقسیم شود تا اینکه هر Determinant یک Candidate-Key را تمثیل نماید.

نارمل سازي، بطور خلاصه، عبارت از دو اصل ذکر شده بالا میباشد.

شکل(16) Levelهای مختلف Normalization را نشان میدهد.



شکل (16) Levelهای مختلف Normalization

طوریکه در شکل دیده میشود Levelهای مختلف نارمل سازي وجود دارند که بنام مراحل نارملایزیشن نیز یاد مي شوند و عبارت اند از 1stNormalForm(1NF) ، 2nd Normal Form (2NF)، 3rd Normal Form (3NF)، Boyce-Codd Normal Form (BCNF)، 4th Normal Form(4NF)، 5thNormalForm(5NF) و Domain Key Normal Form (DKNF)، ص. 351، 352 [3].

1NF: هر جدول یا Relation که شرایط ذکر شده اي صفحه اي "39" را صدق نماید، عبارت از Relation در شکل 1st NF (Normal Form) میباشد [3، ص. 358].

شکل "1NF" آسانترین شکل در نارمل سازي میباشد. کمک کمی در قسمت رفع مشکلات در دیتابیس مینماید. و یک شکل ضروری بخاطر تطبیق مراحل بعدی نارمل سازي میباشد.

2NF: جدولی در شکل 2NF استکه در شکل 1NF بوده، برعلاوه هر مشخصه غیرکلیدی (NonKey) آن، تابع تمام ستونهاي شامل د ر Primary-Key همان جدول باشد (در صورتیکه Primary-Key به شکل ترکیبی باشد)]2، ص. 290[.

نوت: در شکل یا مرحله اي "2NF" تاثیر کلیدهای ترکیبی قابل توجه است.

3NF: جدولی در شکل 3NF استکه در شکل 2NF بوده، علاوتا هیچ Transitive-Dependency در آن موجود نباشد]2، ص. 291[.

BCNF (Boyce/Codd Normal Form) : جدولی در شکل BCNF استکه تمام Determinantهای Functional-Dependency موجود در آن یک Candidate-Key باشد. بعباره دیگر، جدولی در شکل BCNF گفته میشود که هر Candidate-Key آن Determinant حد اقل یک مشخصه در همان جدول باشد]2، ص. 294[.

با در نظر گرفتن اصول دیزاین ذکر شده اي بالا، تقریبا تمام مشکلات جدول های غیرمرتب (Denormalized) رفع خواهد شد. یعنی جدول ها بصورت بسیار ساده و آسان Normalize میشوند و از آنها در ایجاد و پروسس دیتابیس استفاده شده میتواند.

پروسه های نارمل سازي

اگر اصول دیزاین ذکر شده، بالای یک Relation تطبیق شوند، پروسه های ذیل جهت اجراء نارمل سازي ضرورت اند]4، ص. 38[:

1. تشخیص و پیدا نمودن تمام کلیدهای Candidate در جدول.
2. تشخیص و پیدا نمودن تمام Functional-Dependencyها در جدول.
3. تفکیک Determinantهای Functional-Dependency که آیا هر Determinant یک Candidate-Key است و یا خیر؟ اگر کدام یک از دیترمنانت ها Candidate-Key نباشد، پس جدول دارای مشکل Normalization میباشد. برای رفع مشکل

Normalization و دیزاین بهتر جدول باید کارهای ذیل اجراء شوند:

a.ستون های مربوطه این Functional-Dependency باید در یک جدول جداگانه گذاشته شوند.

b.همین ستون دیترمنانت Functional-Dependency باید Primary-Key جدول جدید تنظیم شود.

c.یک کاپی همین ستون دیترمنانت باید در جدول اول به شکل Freign-Key باقی بماند.

d.اصل (Referential Integrity Constraint (RIC باید بالای جدول اول یا (Parent Relation) و جدول تشکیل شده جدید یا (Child Relation) تطبیق شود.

4.پروسه سوم تا وقتی تکرار شود که هر ستون Determinant در هر جدول یک Candidate-Key شود.

جهت واضح شدن بهتر موضوع به مثال های ذیل توجه شود:

PRESCRIPTION (PrescriptionNumber, Drug, Dosage, CustomerName, CustomerPhone, CustomerEmail)

جدول مربوطه در شکل (17) نشان داده شده است.

PRESCRIPTION						
PrescriptionNumber	**Date**	**Drug**	**Dosage**	**CustomerName**	**CustomerPhone**	**CustomerEmail**
P10001	5/17/2003	Lipitor	10mg	A.B. Smith	555.123.2233	SmithAB@somewhere.com
P10003	5/17/2003	Amoxicillan	35mg	Jeff Rhodes	555.125.5588	RhodesJ@somewhere.com
P10004	5/17/2003	Lipitor	20mg	Sarah Smith	555.123.2233	SmithS@somewhere.com
P10007	5/19/2003	Nexium	20mg	Michael Frye	555.145.6666	MF@somewhere.com
P10010	5/19/2003	Nexium	20mg	Jeff Rhodes	555.125.5588	RhodesJ@somewhere.com

شکل (17) دیتا جدول PRESCRIPTION

تطبیق مرحله اول نارمل سازي

در ابتدا کلیدهای Candidate این جدول تشخیص داده میشوند. طوریکه دیده میشود، ستون PrescriptionNumber، دیترمنانت ستونهای Drug، Date و Dosage است. چون یک نسخه یا (Prescription)، بصورت قانونی باید برای یک مریض باشد، پس همین ستون PrescriptionNumber دیترمنانت ستون

ها ی CustomerPhone، CustomerName و CustomerEmail نیز میباشد. در این صورت گفته میشود که ستون PrescriptionNumber عبارت از یک Candidate-Key در جدول PRESCRIPTION است.

آیا در این جدول Candidate-Keyهای دیگر وجود دارد؟

- ستون های Drug، Date و Dosage نمیتوانند Candidate-Key باشند، زیرا امکان نوشتن چندین نسخه به عین تاریخ، امکان نوشتن چندین نسخه برای عین دوا و امکان نوشتن چندین نسخه با عین Dosage میباشد.

در مورد سه ستون دیگر، که دیتا در مورد مریض (در اینجا Customer) دارند، گفته میشود که اگر مریض صرف یک نسخه دوا بگیرد، ستون CustomerEmail، دیترمنانت ستون های مربوط Customer و ستون های مربوط Prescription یعنی تمام ستون ها، میباشد. اما، چون یک مریض امکان دارد اضافه از یک نسخه دوا بگیرد، این ستون (CustomerEmail) نمیتواند یک Candidate-Key باشد.

پس یک کلید Candidate جدول PRESCRIPTION عبارت از ستون PrescriptionNumber است.

تطبیق مرحله دوم نارمل سازي

مطابق مرحله دوم پروسه نارمل سازي، تمام Functional-Dependencyها باید مشخص گردند. طوریکه قبلا نیز گفته شد، PrescriptionNumber دیترمنانت تمام ستون های جدول است. اگر برای یک دوا صرف یک Dosage تعیین باشد، گفته میشود که Dosage → Drug

اما این گفته درست بوده نمیتواند، چرا که یک دوا امکان دارد برای مریضان مختلف به Doseهای مختلف تجویز گردد. بنا ستون Drug، دیترمنانت نمیباشد. بهمین ترتیب Dosage دیترمنانت نیست، بخاطریکه عین Dosage امکان دارد برای دواهای مختلف تطبیق شود.

در زمره اي ستون های مربوط Customer (سه ستون اخیر شکل(16))، بصورت واضح یک Functional-Dependency دیده میشود و آن عبارت است از:

CustomerEmail → (CustomerName, CustomerPhone)

نوت: این در حالتی درست است که هر Customer دارای آدرس Email باشد.

تطبیق مرحله سوم نارمل سازي

مطابق این مرحله در پروسه نارمل سازي، دیده میشود که دیترمنانتی وجود دارد و این دیترمنانت Candidate-Key نمیباشد (ستون CustomerEmail). پس جدول PRESCRIPTION دارای مشکل Normalization است. برای رفع این مشکل، مطابق مرحله سوم قانون نارمل سازي، این Functional-Dependency باید به جدول دومی انتقال داده شود و جدول یاد شده تقسیم یا Split شود. یعنی

CUSTOMER (CustomerName, CustomerPhone, CustomerEmail)

ستون دیترمنانت Functional-Dependency یعنی ستون CustomerEmail، کلید اولیه برای جدول جدید (یعنی جدول CUSTOMER) تنظیم شد.

بهمین ترتیب، یک کاپی از ستون CustomerEmail در جدول اساسی به شکل Freign-Key باقی مانده است و ساختمان فعلی جدول PRESCRIPTION به شکل ذیل تغیر نموده است:

PRESCRIPTION (PrescriptionNumber, Date, Drug, Dosage, *CustomerEmail*)

قاعده RIC برای این دو جدول عبارت است از:

-عناصر ستون CustomerEmail در جدول CUSTOMER باید شامل ستون CustomerEmail در جدول PRESCRIPTION باشند.

طوریکه دیده میشود، هیچکدام از جداول بالا دارای Determinant نمیباشند که Candidate-Key نباشد. یعنی تمام ستون های دیترمنانت ، Candidate-Key همان جدول نیز میباشند. بنا گفته میشود اینکه جداول بالا فعلا Normalize اند.

یک مثال دیگر در نظر گرفته شود: جدول شکل(18) دیتا محصلین داخل لیلیه را نشان میدهد.

STUDENT			
StudentNum	**StudentName**	**DormName**	**DormCost**
100	Smith	Stephens	$3,500.00
200	Johnson	Alexander	$3,800.00
300	Abernathy	Horan	$4,000.00
400	Smith	Alexander	$3,800.00
500	Wilcox	Stephens	$3,500.00
600	Webber	Horan	$4,000.00
700	Simon	Stephens	$3,500.00

شکل(18) دیتا محصلین داخل لیلیه در یک جدول

اولا کلیدهای Candidate باید تشخیص داده شوند. چون ستون StudentNum دیترمنانت تمام ستون های جدول است، پس این ستون یک Candidate-Key میباشد. ستون StudentName، بخاطر داشتن نام 'Smith' برای دو محصل، Candidate-Key بوده نمیتواند. هیچ ستون دیگر مشخص کننده ستون های داخل جدول بوده نمیتوانند. پس ستون StudentNum یگانه کلید Candidate این جدول میباشد.

در قدم دوم Functional-Dependencyها در جدول جستجو میشوند. علاوه از ستون StudentNum، یک Functional-Dependency دیگر نیز وجود دارد و آن عبارت است از

DormName → CormCost

در مرحله سوم دیترمنانتی جستجو میگردد که Candidate-Key نباشد و آن عبارت است از ستون DormName. پس این جدول دارای مشکل Normalization است.

برای حل این مشکل، ستون های مربوطه در جدول جدید جابجا میشوند. نام جدول جدید DORM گذاشته شده، ستون DormName بحیث Primary-Key تنظیم میشود. همچنان یک کاپی از ستون DormName به شکل Freign-Key در جدول اساسی (STU-DORM) باقی میماند.

بعد از طی این مراحل، جدول های ذیل دیزاین خواهند شد.

STU-DORM (StudentNum, StudentName, *DormName*)

و

DORM (DormName, DormCost)

قاعده RIC برای این جداول عبارت است از:

-عناصر ستون DormName جدول DORM باید شامل ستون DormName جدول STU-DORM باشند.

شکل(18) جداول STU-DORM و DORM را بعد از طی مراحل نارمل سازي نشان میدهد.

STU-DORM

StudentNum	StudentName	DormName
100	Smith	Stephens
200	Johnson	Alexander
300	Abernathy	Horan
400	Smith	Alexander
500	Wilcox	Stephens
600	Webber	Horan
700	Simon	Stephens

DORM

DormName	DormCost
Stephens	$3,500.00
Alexander	$3,800.00

DORM	
DormName	**DormCost**
Horan	$4,000.00

شکل(18) دیتا محصلین داخل لیلیه در دو جدول

شکل نارمل یک جدول

نظریه بودن یک جدول در شکل Normal در اوایل طوری بود، که گفته میشد یک جدول در سه شکل یا مرحله به حالت 3rd Normal Form (شکل کامل بودن نارمل) میرسد. وقتی یک جدول صرف شرایط بودن یک Relation ذکر شده در صفحه "39" را تکمیل نماید، در حالت 1NF میباشد. در این حالت جدول دارای نواقصی بوده، بعد از رفع بعضی از این نواقص، جدول به حالت 2NF و بعد هم به حالت 3NF دیزاین میشد. این نظریه که توسط E. F. Codd طرح شده بود، برای مدتی، حد کامل Normalization برای جداول قبول شده بود.

بعد از چندی، به اثبات رسید که یک جدول با وجود بودن در شکل 3NF، دارای نواقصی میباشد. پس یک شکل دیگر نارمل سازي بنام 'Boyce-Code Normal Form' یا (BCNF)

توسط خود E. F. Codd و یک شخص دیگر بنام R. Boyce طرح گردید تا مشکلات و نواقص شکل 3NF را مرفوع سازد.

با توجه به تعریفات بالا، گفته میشود که یک Relation در شکل BCNF بصورت اتوماتیک در شکل 3NF است و یک Relation در شکل 3NF نیز در شکل 2NF و بعین ترتیب در شکل 1NF میباشد.

بعدها نظر به ضرورت، شکل 4NF جهت رفع مشکلات یک نوع دیگر Dependency در جدول بنام Multi Valued Dependency دیزاین شد. و بعد هم مشکلی در قسمت جدا کردن جدول ها ظهور کرد و آن طوری بود که وقتی یک جدول به جداول دیگر تقسیم میشد، دوباره بصورت درست با هم یکجا شده نمی توانستند، برای رفع این مشکل، شکل 5NF طرح گردیده و عملی میباشد.

یک طریقه دیگر نارمل سازي در سال 1981 توسط R. Fagin طرح گردید که بنام Domain Key Normal Form یا (DKNF) یاد میشود. چون طریقه DKNF زیاد معمول نشده است، به این خاطر توضیحات بیشتر در ضمینه داده نمیشود.

به هر صورت، موضوع مهم در مورد نارمل سازي (Normalization) که قبلا نیز به آن اشاره شده است، اینست که هر دیترمنانت یک Functional-Dependency باید یک Candidate-Key در آن Relation باشد.

دیتا مودل E-R

Data-Modeling and the Entity Relationship (E-R) Model

در دروس گذشته، معلومات مقدماتی در مورد مفهوم دیتابیس و مسایل مربوط به اداره دیتابیس توضیح گردید. در قسمت اول معلومات عمومی در مورد معرفی اجزا سیستم دیتابیس، و اینکه یک دیتابیس متشکل از جداول مرتبط است، توضیح گردید. قسمت های بعدی توضیحات در مورد مودل رابطه اي (Relational-Model) را دارا بوده، برعلاوه معلومات مقدماتی در مورد Functional-Dependency و پروسه های نارمل سازي در آن گنجانیده شده است.

با درنظر گرفتن معلومات ارائیه شده، استفاده کننده این نوت فعلا باید آشنایی با سیستم های اداره، ایجاد و استفاده از دیتابیس را داشته باشد. اما استفاده عملی و درک مزیت های دیتابیس به مطالعه بیشتر و اجراء کارهای عملی در زمینه را ایجاب مینماید. در این درس و درسهای بعدی، با در نظر داشتن امکانات، کوشش بعمل آمده است تا هرچه واضحتر و بهتر مسایل مربوطه توضیح و عملی گردند.

پروسه های دیزاین یک دیتابیس

بخاطر دیزاین یک دیتابیس، سه مرحله ذیل باید در نظر گرفته شوند:

1.تعیین هدف و تهیه مواد دیتابیس (تیوري – بدون استفاده از DBMS)

1)تحلیل UoD عنوانیکه دیتابیس به آن دیزاین مي شود

2)تهیه کردن مودل یا دیاگرام دیتا (Data Model)

3)تبدیل کردن دیتا مودل به مودل رابطه اي (Relational Model) با جداول نارمل شده

2.دیزاین دیتابیس (عملي - با استفاده از DBMS)

1) ایجاد کردن دیتابیس

2) ایجاد جدول ها در داخل دیتابیس

3)ایجاد ارتباط ها (Relationships) بین جدول ها

3.اجراء دیتابیس (عملي – با استفاده از DBMS)

1)دیزاین کردن بخش هاي محیط استفاده (Interface) مانند فورم

ها ، کیوري ها ، راپورها ،
مینوها و غیره

-برای ایجاد یک دیتابیس، دیزاینر باید بداند که به چی هدف دیتابیس ایجاد میشود. جهت تهیه مواد، نظریات استفاده کننده گانی که در آینده آنرا بکار میبرند، باید گرفته شوند. طرح های از جدول ها، ارتباط ها بین جدول ها، قوانین و مقررات به اساس درخواست استفاده کنندگان و غیره باید ترتیب شوند. یعنی یک مودل دیتا یا (Data-Model) ایجاد شود، که تمام محتویات دیتابیس، Relationshipها و قوانین قابل تطبیق در داخل دیتابیس را نشان دهد. همچنان دیتامودل تهیه شده باید با در نظرداشت قوانین نارمل سازي جداول به مودل رابطه اي (Relational Model) تبدیل شود تا در قسمت عملي ایجاد دیتابیس سهولت هاي را به میان آورد.

-در جریان مرحله دوم دیزاین دیتابیس، همین مودل رابطه اي با جداول نارمل شده به شکل فایل دیتابیس درآورده مي شود. در این مرحله دیتابیس ایجاد مي شود. اولین کاریکه بعد از ایجاد دیتابیس باید صورت گیرد اجرا مي شود، یعني جدول ها به صورت عملي ایجاد مي شوند. ارتباط ها بین جدول

ها به همان شکلي که در ديتا مودل و مودل رابطه اي طرح شده اند، به شکل عملي بالاي جدول ها تطبيق مي شوند. جدول ها با نام های معین، نام های ستون های جدول ها، ارتباط ها شامل کليدهاي اوليه و کليد هاي خارجي ، Constraintها مشخص ميشوند. همچنان مشخص نمودن قواعد RIC و Business Ruleها * نيز شامل اين دوره ميشوند. بصورت خلاصه اگر گفته شود، در مرحله دوم شکل و ساختمان عمومي ديتابيس (Database Structure) با استفاده از کود SQL و يا هم وسايل داخلي DBMS ترتيب و تطبيق مي شود.

-مرحله سوم عبارت از ديزاين محيط استفاده براي استفاده کنندگان (User Interface) مي باشد. در اين مرحله ديزاينر ديتابيس با استفاده از امکانات DBMS و يا هم مستقيماً با تطبيق کود SQL بخش هاي محيط استفاده براي استفاده کنندگان ديتابيس را ايجاد مي نمايد. اين بخش ها شامل فورم ها، مينوها، کيوري ها، راپورها، صفحه ها و غيره مي باشد. اجزا ياد شده به صورت ديناميکي با درنظرداشت اصل مودل رابطه اي در هر زماني که استفاده مي

مثال های Business Rules برای وضاحت این اصطلاح طور ذیل اند:
- در یک فروشگاه اجناسی قابل فروش اند که حد اقل، یک گرنتی شرکت تولیدی همان جنس را داشته باشند.
- محصلین فاکولته کمپیوتر ساینس قبل از شروع سمستر پنجم باید رشته بندی شوند.

شوند، Update شده و نشان داده مي شوند. برعلاوه، استفاده كردن عملي تمام بخش هاي (User Interface) به صورت عملي استفاده مي شوند. بعد از رفع نقايص و كمبودي ها كاپي اخير ديتابيس به سيستم نصب (Install) شده و در دسترس استفاده كننده گان قرار داده مي شود.

نوت: راهنماي استفاده از ديتابيس كه در اصطلاح علم كمپيوترسايندس بنام (Documenation) ياد ميگردد و مشابه به User-Manual يك جنس مي باشد، بايد توسط ديزاينر ديتابيس ترتيب شده و در دسترس استفاده كنندگان قرار داده شود. Documenation ميتواند به داخل ديتابيس به شكل Command-Button يا مينو ارتباط داده شود، و يا هم در فايل جداگانه ترتيب شده و ضميمه ديتابيس شود.

اين درس مرحله اول ديزاين ديتابيس را بطور مفصل تحت پوشش قرار داده و موضوع بحث Data-Modeling ميباشد. يک نوع مهم مودل ديتا، بنام EntityRelationshipDataModel يا E-R Data-Model در اين قسمت توضيح ميگردد. اهميت يک ديتا مودل E-R در اين است كه نشان مي دهد به چی شكل اهداف ايجاد يک ديتابيس برای يک پروژه تجارتی تعين شده و بروي دياگرام نشان داده مي شود. استفاده كنندگان ديتابيس خواه مسلكي علم ديتابيس باشند و يا خير، مي

توانند دیاگرام یا مودل تهیه شده با سمبول هاي E-R را بدانند و نظر بدهند.

The E-R Data-Model

اهداف و یا مقاصدی که یک دیتابیس به اساس آن ایجاد میگردد، باید به یک شکلی از Data-Model تبدیل گردند تا مطالب و مفاهیم اساسي را بشکل درست آن تمثیل نماید. طوریکه برای نوشتن و پروگرام کردن Applicationها، نظریه که به اساس آن پروگرام نوشته میشود، اولا باید به شکل Flowchartها و یا دیاگرام های تمثیل دیتا تعویض شوند.

برای اجراء این منظور راه های مختلفی استفاده شده میتوانند. یکی از این راه ها، که بسیار معمول نیز است، استفاده از دیتا مودل E-R یا Entity-Relationship(E-R) Model است. نمونه های مختلف دیتا مودل E-R استفاده شده اند، در این نوت دو نوع آن، که تقریبا بیشتر معمول میباشند]4، ص. 87[، تحت مطالعه قرار میگیرند. سمبول های استفاده شده برای نشان دادن اولین نمونه یا Version یک دیتا مودل بصورت ساده توسط تمام اشخاص مسلکی قابل تفکیک میباشد.

نمونه یا Version دومی مودل E-R عبارت از دیتا مودل UML یا Unified Modeling Language است. در مودل UML، سمبول ها و نوت های متفاوت استفاده میشوند یعنی این سمبول ها با سمبول های مودل معمول E-R دارای تفاوت های اند. سمبول هایکه جهت نمایش دیتا در مودل UML استفاده میشوند، در این نوت توضیح شده اند. بنا استفاده کننده

این نوت، قادر به تبدیل و ترجمه سمبول های مودل E-R به مودل UML و برعکس خواهد بود.

عناصر مهم مودل E-R عبارت اند از:

Entities-

Attributes-

Identifiers-

Relationships -

Cardinality-

Existence Dependency-

Weak and ID-Dependent Entities-

Composite Entities-

هرکدام اینها توضیح میگردند!

Entityها

یک Entity عبارت از چیزی (مثلا قسمتی از دیتا) است که استفاده کنندگان بتوانند آنرا در دیتابیس بکار برند. به اساس یک Entity اکثرا یک یا چند جدول در دیتابیس ایجاد میشوند.

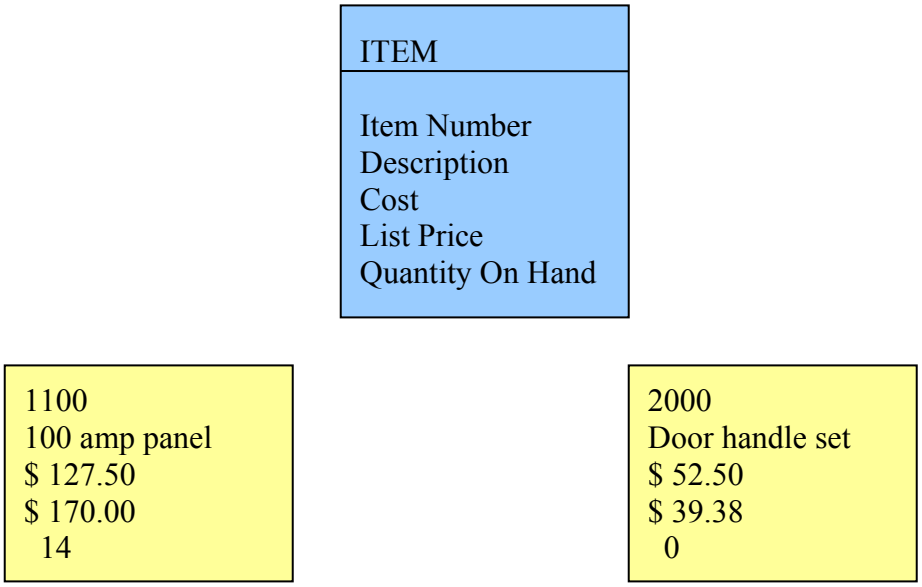
مثال های از Entity عبارت اند از: CUSTOMERS، SALES-PERSONS، PRODUCTS، PURCHASES و غیره.

طوریکه در مثال های بالا دیده میشود، یک Entity نشان دهنده یک عنوان یا Topic معین است.

تمام Entityهای که از یک نوع باشند، در یک گروپ تنظیم شده و همین گروپ بنام Entity-Class همان نوع، یاد میشود. یعنی گروپ Entity هائیکه از نوع PRODUCT باشند، مربوط Entity-Class نوع PRODUCT میشوند.

Entity-Class با Entity-Instance تفاوت دارد. Entity-Instance یک جزء مشخص یا ریکوردي از Entity-Class را نشان میدهد. بدین مفهوم که گروپی از Entity-Instanceها با هم یکجا گردیده و یک Entity-Class تشکیل داده میشود. Entityها اکثرا توسط فورمها و یا مستقیما توسط جدولها در داخل دیتابیس تعریف و استفاده میشوند. یعنی Entityها قسمت اساسی اکثر جدول ها و فورم ها را در دیتابیس تشکیل

میدهند. شکل (19) نشاندهنده یک Entity-Class و دو Entity-Instance مربوط همین Entity-Class میباشد.



شکل(19) یک Entity-Class و دو Entity-Instance

در دیتابیس انواع دیگر Entityها نیز وجود دارند. بعضی از آنها تحت عناوین مربوطه در صفحات بعدی توضیح میشوند.

مشخصه ها (Attributes)

Entityها دارای مشخصه ها یا مشخصه ها اند، که این مشخصه ها نشاندهنده نوعیت Entityها میباشند. مثال های مشخصه ها در شکل(19) نشان داده شده اند و عبارت اند از: Cost، Description، ItemNumber و غیره. طوریکه دیده میشود، برای نوشتن نام مشخصه ها از حروف کوچک و بزرگ

استفاده میشود. در مودل E-R و در دیتابیس، عناصر یک مشخصه باید از یک نوع باشند، یعنی دارای یک Domain باشند]2، ص. 198[.

مشخصه ها دارای نوع دیتا (Data-Type) و Propertyهای مشخص میباشد. DataType و Property یک مشخصه در مرحله اول دیزاین دیتابیس تعین میشود. مثال های Data-Type برای مشخصه ها عبارت از Currency، Date، Numeric، Character، ... میباشند. Property شرایطی برای داخل کردن دیتا به مشخصه ها را وضع مینماید. همچنان تعین مینماید که یک مشخصه بصورت Default کدام دیتا را داشته باشد و قابلیت گرفتن عناصر خالی (Null) را داشته باشد و یا خیر؟ و غیره.

مشخصه ها دارای یک Domain اند. Domain عبارت از ست عناصر ممکنه برای یک مشخصه میباشد. مثلا برای یک مشخصه بنام Gender، که جنس را تعیین مینماید، صرف دو حرف M و F و یا هم دو کلمه Male و Female ممکن بوده و عبارت از Domain همان مشخصه میباشد. و یا هم برای مشخصه بنام Title، کلماتی از قبیل Mrs، Mr.. و Mss عبارت از Domain میباشد. یک Domain توسط چندین مشخصه مشترک یا Share شده میتواند. آدرس های Student و Professor، طور مثال، میتوانند از عین Domain یا گروپ آدرسهای ممکنه، استفاده نمایند.

Attributeها نظر به ساختمان شان به دو نوع اند:

2)مشخصه ترکیبی (Composite-Attribute)

3)مشخصه ساده (Simple-Attribute)

-یک مشخصه ترکیبی (Composite-Attribute) عبارت از مشخصه است که بتواند به چندین مشخصه دیگر تقسیم و یا تجزیه شود. مثلا Address میتواند به مشخصه های State، City، Street و Zip تجزیه شود.

-یک مشخصه ساده (Simple-Attribute) نمیتواند به اجزاء کوچکتر تجزیه شود. مثال های این نوع مشخصه ها عبارت اند از: Marital-Status، Gender، Age، و غیره.

مشخصه ها نظر به داشتن محتواء به دو نوع اند:

Single-Value-1

Multi-Value-2

-مشخصه های Single-Value، دارای دیتا معین برای هر جزء میباشند. یعنی برای هر Entity، صرف یک جزء دیتا موجود میباشد.

مانند Social-Security-Number اشخاص، و یا Serial-Number یک جنس و غیره.

-مشخصه های Multi-Value میتوانند دارای دیتا مختلف باشند. مثلا یک شخص میتواند چندین نمبر تلفون با فارمت های مختلف (موبایل، دفتر، خانه، ...) داشته باشد و یا تاریخ های مختلف از قبیل شمسی، قمری و یا میلادی در یک مشخصه استفاده شود.

نوت: طوریکه در بخش نارمل سازي (Normalization) خوانده شد، در حالتیکه یک مشخصه ترکیبي وجود داشته باشد، مشخصه متذکره به اجزاء کوچکتر تقسیم شده و هرکدام آن یک Entity جدید را تشکیل داده و به Entity قبلی یا اساسی Relationship میداشته باشند.

Identifierها

Entity-Classها دارای Identifier اند [4، ص. 88].
Identifierها عبارت از مشخصه های اند که یک Entity-Instance را شناسایي یا Identify میکنند. طور مثال Entity-Instance بنام EMPLOYEE میتواند توسط Social-Security-Number، و یا توسط EmployeeNumber و یا هم توسط EmployeeName شناسایي شود.

Identifier یک Entity-Instance، امکان دارد از یک مشخصه تشکیل شود و یا هم از تعداد بیشتر مشخصه ها ترکیب شود. Identifier که از دو و یا اضافه از دو مشخصه بصورت ترکیبي تشکیل شده باشد، بنام Composite-Identifier یاد میشود. مثال های Identifier ترکیبی عبارت اند از (AreaCode, LocalNumber)، (ProjectName, TaskName)، (FirstName, LastName, Phone) و غیره.

یک Identifier میتواند دیتا تکراری نگرفته و یا هم دیتا تکراری را قبول نماید. اگر امکان تکرار دیتا در یک Identifier نباشد (دارای دیتا Unique باشد)، محتواء آن تنها یک Entity-Instance را شناسایي مینماید. در صورت قبول دیتا تکراری توسط Identifier، گروپی از Entity-Instanceها توسط آن شناسایي میشوند. مثلا EmployeeNumber، معمولا یک Identifier از نوع اول (با دیتا غیر تکراری) بوده و EmployeeName با وجود بودن Identifier، میتواند دیتا تکراری را هم قبول نماید.

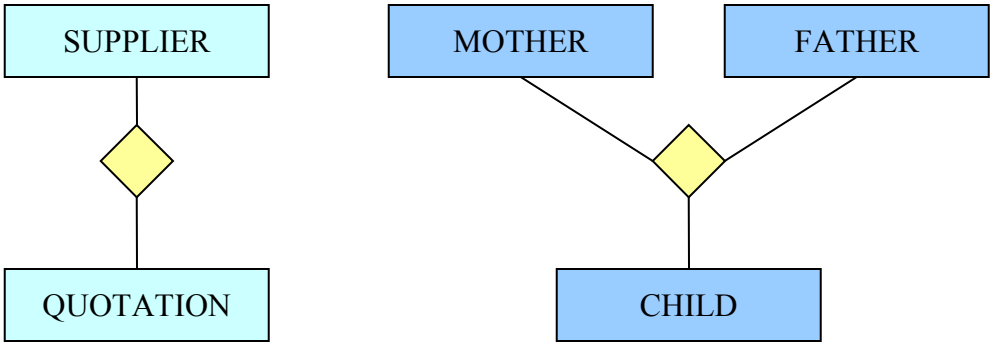
با در نظر گرفتن توضیحات بالا، یک Identifier در دیتا مودل E-R مشابه به Primary-Key در Relational-Model است. البته بعضی تفاوت های بین آنها وجود دارد، از جمله، تفاوت اول اینکه Identifier به مفهوم منطقی یا Logic آن در مودل E-R استفاده میشود. یعنی نام مشخصه یا مشخصه هائیکه بحیث Identifier استفاده میشوند، مفهوم مکمل یک Entity را برای استفاده کنندگان افاده مینماید. دوم اینکه، Primary-Keyها و Candidate-Keyها حتما باید دارای

دیتا غیرتکراری (Unique) باشند، در حالیکه Identifierها هر دو امکان را دارند، یعنی قابلیت گرفتن دیتا تکراری را نیز میداشته باشند.

Relationshipها

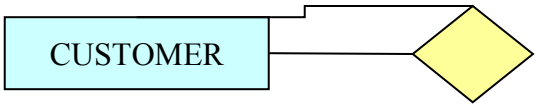
Entityها را میتوان با ایجاد Relationshipها با یکدیگر ارتباط داد. در مودل E-R دو نوع Relationship وجود دارد. یکی Relationship-Class و دیگر Relationship-Instance. اولی بخاطر ارتباط دادن Entity-Classها و دومی بخاطر ارتباط دادن Entity-Instanceها استفاده میشود. Relationship-Instance به نام Recursive-Relationship نیز یاد میگردد که بعد تر توضیح میشود.

در Entity-Class امکان ارتباط دادن چندین Entity-Class وجود دارد (اضافه از دو). تعداد Entity-Classها در Relationship عبارت از درجه Relationship میباشد]4، ص. 89[،]2، ص. 204[. شکل(20) Relationshipهای از درجه "2"، درجه "3"، و Recursive-Relationship را نشان میدهد.



Binary Relationship

Ternary Relationship



Recursive-Relationship

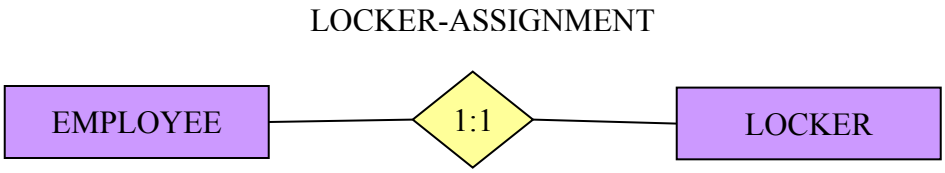
شکل(20) مثال های از Relationshipها

نوت: Relationshipهای با درجه 2 یعنی (Binary-Relationships) بیشتر معمول اند.

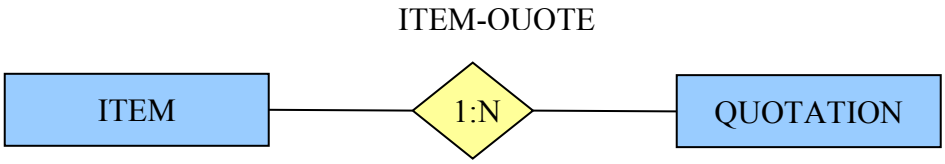
در این قسمت Binary-Relationshipها و Recursive-Relationshipها مورد بحث قرار میگیرند.

Binary-Relationships

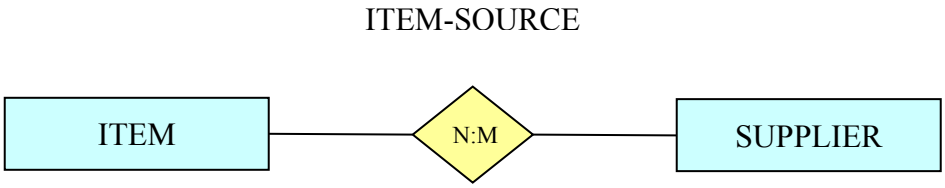
Relationshipهای با درجه 2 یا Binary به سه شکل ایجاد شده میتوانند]4، ص. 89[. مثال های آن در شکل(21) نشان داده شده اند.



)الف) یک به یک (باینری



)ب) یک به چندین (باینری



)ج) چندین به چندین (باینری

شکل(21) سه مثال از Binary-Relationshipها

در جزء (الف) شکل بالا، یعنی Relationship یک به یک (1:1) ، Entity-Instance از یک نوع به Entity-Instance از نوع دیگر ارتباط (Relationship) دارد. در Relationship شکل(الف) (LOCKER-ASSIGNMENT) یک EMPLOYEE به یک LOCKER و برعکس، ارتباط برقرار مینماید. پس هیچ EMPLOYEE نمیتواند اضافه از یک LOCKER داشته باشد و هیچ LOCKER نمیتواند به اضافه از یک EMPLOYEE ارتباط بگیرد.

جزء (ب) شکل بالا، نوع دوم Binary-Relationship را نشان میدهد. این نوع Relationship عبارت از یک به چندین (N:1) است. در این Relationship یعنی (ITEM-QUOTE)، یک عنصر از ITEM به چندین QOUTATION ارتباط داشته، یعنی دارای چندین QUOTATION بوده، اما یک QUOTATION تنها میتواند به یک عنصر از ITEM ارتباط بگیرد.

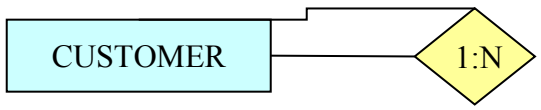
جزء (ج) شکل(21)، نوع سوم Binary-Relationship را نشان میدهد. در این حالت (چندین به چندین یا N:M)، چندین عنصر از هر دو جانب در Relationship سهم گرفته میتوانند. در مثال ذکر شده (ITEM-SOURCE) شکل(21-ج)، یک ITEM میتواند توسط چندین شخص Supply شود و یک Supplier میتواند چندین ITEM را تهیه نماید.

وقتی به شکل(21) دقت شود، دیده میشود که سمبول 1، N و M که به جانب Entity قرار دارند، نشان دهنده سهم گرفتن Entity در Relationship میباشد. مثلا در جزء (ب) شکل(

21)، سمبول عدد "1" به سمت ITEM نوشته شده است و نشان میدهد که یک عنصر از ITEM در Relationship سهم میگیرد و سمبول "N" نشاندهنده سهم گرفتن چندین QUOTATION در Relationship است. اگر موقعیت سمبول های "1" با "N" عوض شود، در نتیجه چندین ITEM دارای یک QOUTATION و یک QOUTATION دارای چندین ITEM خواهد بود. پس موقعیت سمبول ها در ارائیه Relationshipهای مودل E-R مهم میباشد.

Recursive-Relationships

یک Entity میتواند Relationship را بین عناصر خود Entity به شکل داخلی ایجاد نماید]4، ص. 90[. چنین Relationshipها بنام Recursive-Relationships یاد میشوند. شکل(22) یک Recursive-Relationship را نشان میدهد.



شکل(22) مثال یک Recursive-Relationship

Recursive-Relationshipها مانند Relationshipهای باینری میتوانند (1:1)، (N:1) و (N:M) باشند]4، ص. 90[.

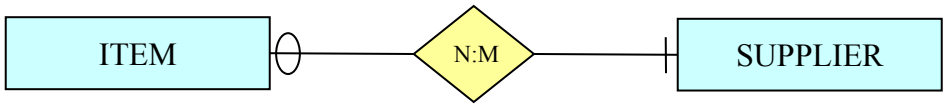
کاردینالتي (Cardinality)

کاردینالتي حد اصغری و حد اعظمی را برای یک Entity بخاطر سهم گرفتن در Relationship تعین مینماید. بعباره دیگر، به هر تناسبیکه یک Entity بتواند در Relationship سهیم شود، عبارت از کاردینالتي همان عنصر در Relationship میباشد]2، ص. 207[.

Minimum-Cardinality به اشکال مختلف نشان داده میشود. از جمله برای نمایش حد اقل یک Entity، یک خط عمودی در کنار Entity بروی خط Relationship ترسیم میشود و برای نمایش حد اقل صفر Entity، یعنی موجودیت Entity در Relationship اختیاری باشد، یک بیضوی کوچک بشکل عمودی در کنار Entity بروی خط Relationship دیاگرام آن ترسیم میشود.

به شکل(23) دقت شود. یک ITEM باید با حد اقل یک SUPPLIER ارتباط یا Relationship داشته باشد، اما یک SUPPLIER مجبور نیست که حتما با یک ITEM ارتباط برقرار نماید. بعباره دیگر یک ITEM کمترین کاردینالتي یک و بیشترین کاردینالتي چندین را با SUPPLIER دارا است. یک SUPPLIER، کمترین کاردینالتي صفر و بیشترین کاردینالتي چندین را با ITEM دارد.

ITEM-SOURCE



شکل(23) Relationshipها با کمترین کاردینالتي

در مثال دیگر فرض شود، یک محصل بتواند 1 الی 6 صنف را در جریان یک سمستر تعقیب نماید و تعداد محصلین در یک صنف از صفر الی 35 نفر بوده میتوانند. شکل (24) دیاگرام مربوطه را نشان میدهد. کاردینالتي اصغری "1" و اعظمی "6" برای محصل و بعین ترتیب کاردینالتي اصغری "صفر" و اعظمی "35" برای صنف تعیین میباشد.

STUDENT-CLASS

شکل(24) مثال نمایش کاردینالتي در یک Relationship

Existence-Dependency

Existence-Dependency در حالتی مهم است که برای ایجاد یک Entity-Instance ضرورت باشد تا یک یا اضافه از یک Entity-Instance در جدول یا جداول دیگر قبلا موجود باشد]2، ص. 210[. در کدام زمان چنین اتفاقی روی میدهد؟ - وقتی ارتباط N:1 بین دو Entity موجود باشد، بصورت قانونی باید کاپی Identifier یکی از Entityها در دیگر آن اضافه شود، در اینصورت Entity-Instance اولی باید موجود باشد تا Entity-Instance دومی اضافه گردیده بتواند. بنا ترتیب داخل کرد ن Entity-Instanceها در ایجاد دیتابیس بسیار مهم میباشد.

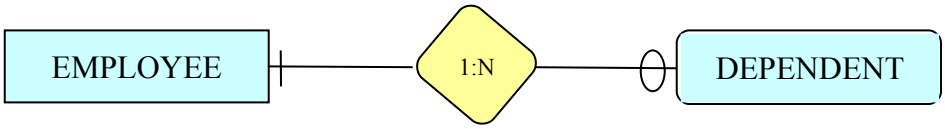
نوت: در بعضی از DBMSها الی تکمیل کار دیتابیس، کدام غلطی در مورد بالا (Existence-Dependency) رخ نمیدهد. اما باید محتاط بود.

Weak and ID-Dependent Entities

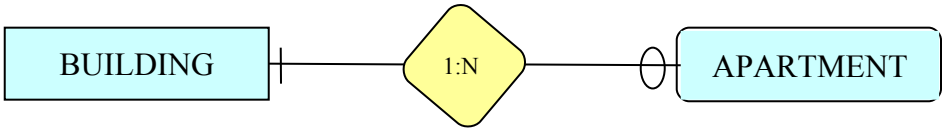
مودل E-R دارای یک نوع مخصوص Entity بنام Weak-Entity یا Entity ضعیف است]4، ص. 91[. چنین Entityها به اساس Entityهای دیگر در دیتابیس موجود بوده میتوانند. بدین مفهوم که وجود Entity دیگر ضروری بوده، اگر آن Entity (که

Strong Entity به آن گفته میشود) وجود نداشته باشد، این نوع Entity وجود داشته نمیتواند [3، ص. 414، 415].

در جزء الف شکل (25) دیتا مربوط DEPENDENT، متکی به دیتا EMPLOYEE است. اگر EMPLOYEE وجود نداشته باشد، دیتا مربوطه، یعنی دیتا DEPENDENT، وجود داشته نمیتواند.



الف: Weak-Entity



ب: ID-Dependent Entity

شکل(25) مثال های Weak-Entities

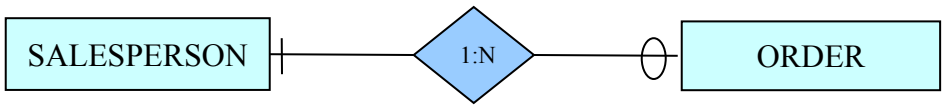
طوریکه در شکل بالا دیده میشود، Weak-Entityها در چوکات با کناره های گول نشان داده میشوند، همچنان چوکات مربوطه چنین Relationship نیز به عین شکل (با کناره های گول) نشان داده میشوند. باید یادآوری شود اینکه در بعضی کتاب ها، برای نشان دادن Weak-Entityها

در مودل E-R چوکات Entity و هم چوکات Relationship مربوطه بصورت دو خطه (Double-Line) نشان داده میشوند.

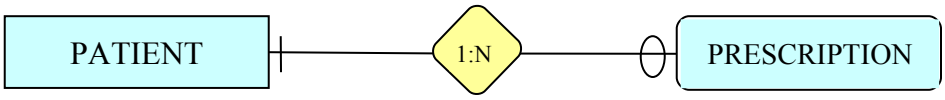
یک نوع دیگر Entity ضعیف بنام ID-Dependent Entity یاد میشود]4، ص. 91[. در چنین حالتی، Identifier یک Entity-Instance برای Entity-Instance دیگر بحیث Identifier و یا جزء از Identifier استفاده میشود، شکل (23-ب).

فرض شود که Identifier جدول BUILDING عبارت از BuildingName بوده و APARTMENT یک Composite-Identifier را استفاده می نماید (BuildingName, ApartmentNumber). چون در ترکیب Identifier اپارتمان، از Identifier تعمیر (Building) استفاده شده است، پس چنین Entity بنام ID-Dependent Entity یاد شده و Relationship آن نیز ID-Dependent Relationship گفته میشود. اگر بصورت منطقی هم فکر شود، این گفته تدقیق میشود که در صورت موجود نبودن تعمیر، اپارتمان هیچگاه وجود داشته نمیتواند.

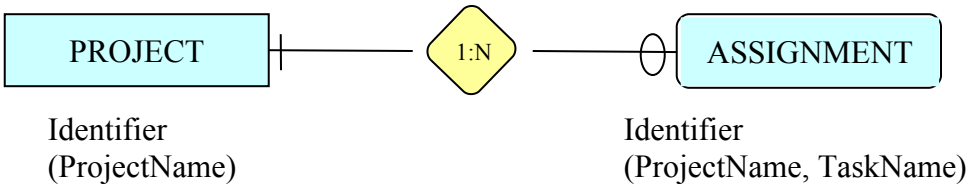
برای وضاحت بیشتر موضوع، مثال های بعدي در نظر گرفته میشود.



الف: ORDER یک Strong Entity است.



ب:PRESCRIPTION یک Weak Entity است.



ج:ASSIGNMENT یک ID-Dependent Entity است.

شکل(26) مثال های Entityها

شکل (26- الف) یک Strong-Entity و Strong-Relationship را نشان میدهد. در این مثال یک شخص فروشنده میتواند چندین ORDER بگیرد و یا هم هیچ ORDER نگیرد و هر ORDER امکان دارد به یک شخص فروشنده (SALESPERSON) ارتباط بگیرد و یا هم بصورت نقده پول جنس به Cashier پرداخته شود. پس ORDER دارای دیتا مستقل بوده و یک Strong-Entity میباشد.

در شکل (26-ب)، یک Weak-Entity به نظر میرسد. یعنی PRESCRIPTION نمیتواند بدون موجودیت PATIENT وجود داشته باشد. وقتی مریضی موجود نباشد، نسخه اي ضرورت نیست. در شکل، نظر به کاردینالتي، یک مریض میتواند هیچ و یا چندین نسخه دوا بگیرد و یک نسخه باید حد اقل به یک مریض مربوط باشد.

بعین ترتیب شکل (26-ج) یک ID-Dependent Entity را نشان میدهد. ASSIGNMENT، که دارای Identifierترکیبی (ProjectName, TaskName) است، یعنی PROJECT، Identifier جزء Identifier آن میباشد. پس وجود ASSIGNMENT متکی به PROJECT است. هر ASSIGNMENT باید حد اقل به یک PROJECT مربوط باشد و یک PROJECT میتواند هیچ و یا چندین ASSIGNMENT داشته باشد.

از تعریف و مثال های بالا نتیجه گیری میشود که:

- Weak-Entityها دارای کمترین کاردینالتي، "یک" در Relationshipها اند.

- تمام Entityهای ضعیف، بصورت منطقی یا Logical مربوط به Entityهای دیگر اند که بنام Entity قوي یاد میشوند.

- تمام Entityهائیکه به اساس 'ID' از Entity دیگر استوار اند، عبارت از Weak-Entity مي باشند.

Composite-Entities

بخاطر به اجرا گذاشتن Relationship چندین به چندین (N:M) بین دو Entity، باید کاپی Identifierهای هر دو Entity گرفته شده و در یک Entity جدید، جابجا شوند]2، ص. 218[. برای به اجراء گذاشتن این عمل، دو Relationship جدید از نوع یک به چندین (N:1) یا یك به یك (1:1) ایجاد میشوند. به این مفهوم که یک Relationship چندین به چندین باید به دو Relationship یک به چندین و یا یك به یك تقسیم شود. چون در Relationship نوع N:1، کاپی Identifier، جناح یک یا Entity اول در جناح چندین یا Entity دوم جابجا میشود و همین کار بترتیب بین Entity سوم و دوم نیز باید اجرا گردد. پس Entity جدیداً تشکیل شده بنام Entity ترکیبی یا Composite-Entity یاد میشود.

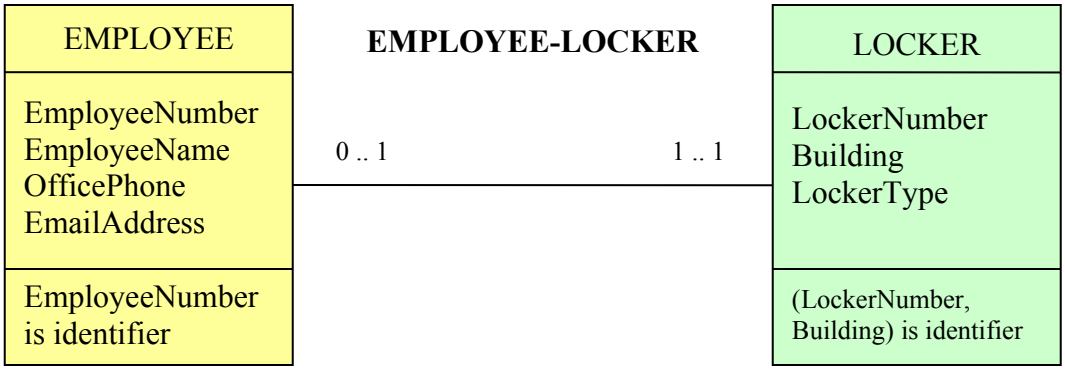
نمایش مودل دیتابیس در UML

UML یا (Unified-Modeling-Language)، عبارت از ستی از دیاگرام ها و ساختمان هایی است که برای نمایش دادن و دیزاین پروگرام های OOP (Object-Oriented) و Applicationها بصورت ستندرد استفاده میشود. برعلاوه UML یک ستی از اشکال و سمبول ها بوده که برای دیزاین و انکشاف پروگرام ها در ساحه سافتویر انجنیرنگ (Software-Engineering) از آن استفاده مي شود. UML بعد از دهه 1980 معمول شده است]4، ص. 93[.

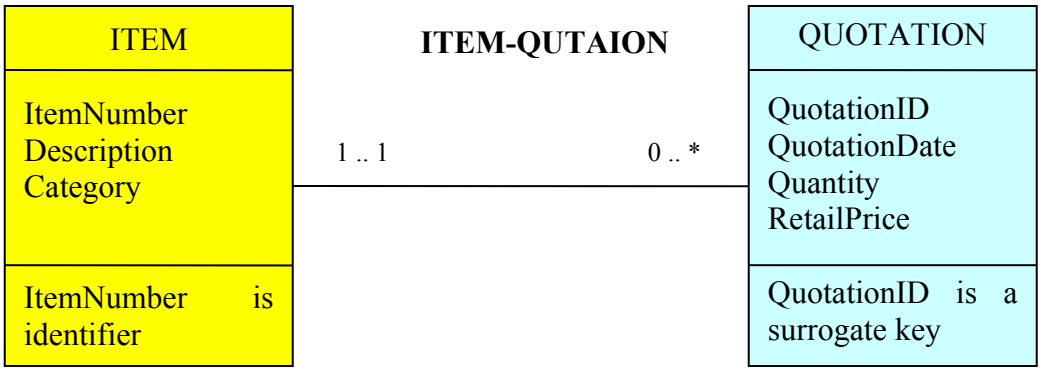
چون UML، بیشتر با پروگرام ها و Applicationها ارتباط دارد، پس لسان یاد شده اساسا موضوع صنف 'System-Development' بوده و به پیمانه کمی در ساحه دیزاین دیتابیس از آن استفاده بعمل میآید. به این جهت، استفاده کنندگان این نوت، باید با دیاگرام های دیتابیس در UML آشنایي داشته باشند. بخاطر باید داشت که در ساحه دیتابیس، دیاگرام های UML مانند دیاگرام های عادی E-R در نظر گرفته شده و استفاده میشوند.

Entityها و Relationshipهای UML

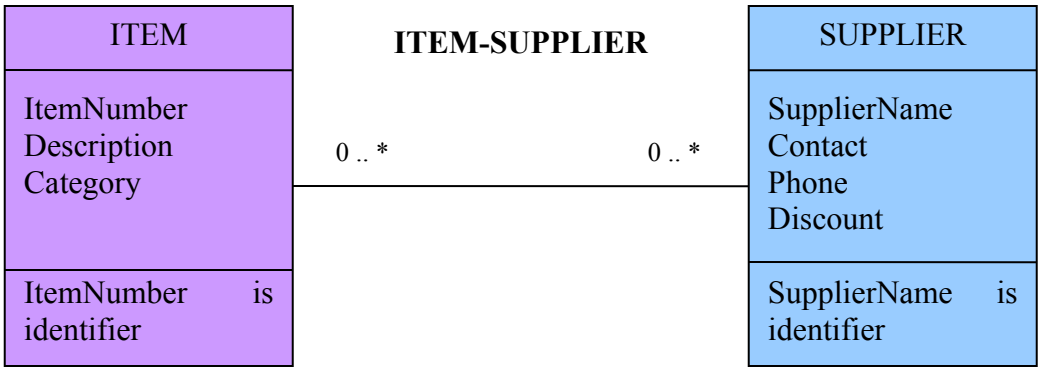
شکل(27) دیاگرام دیتابیس شکل(21) را در دیزاین UML نشان میدهد. هر Entity توسط یک Entity-Class در داخل چوکاتیکه از سه قسمت تشکیل شده، نشان داده شده است. قسمت بالایی چوکات، نام Entity را نشان میدهد. قسمت دوم نام مشخصه های موجود در Entity-Class را نشان میدهد و قسمت سوم نشان دهنده بعضی قواعد و میتودهائیکه پروگرام باید به آن عمل نماید و مربوط Entity اند، میباشد.



الف: Relationship از نوع 1:1



ب: Relationship از نوع N:1



ج: Relationship از نوع N:M

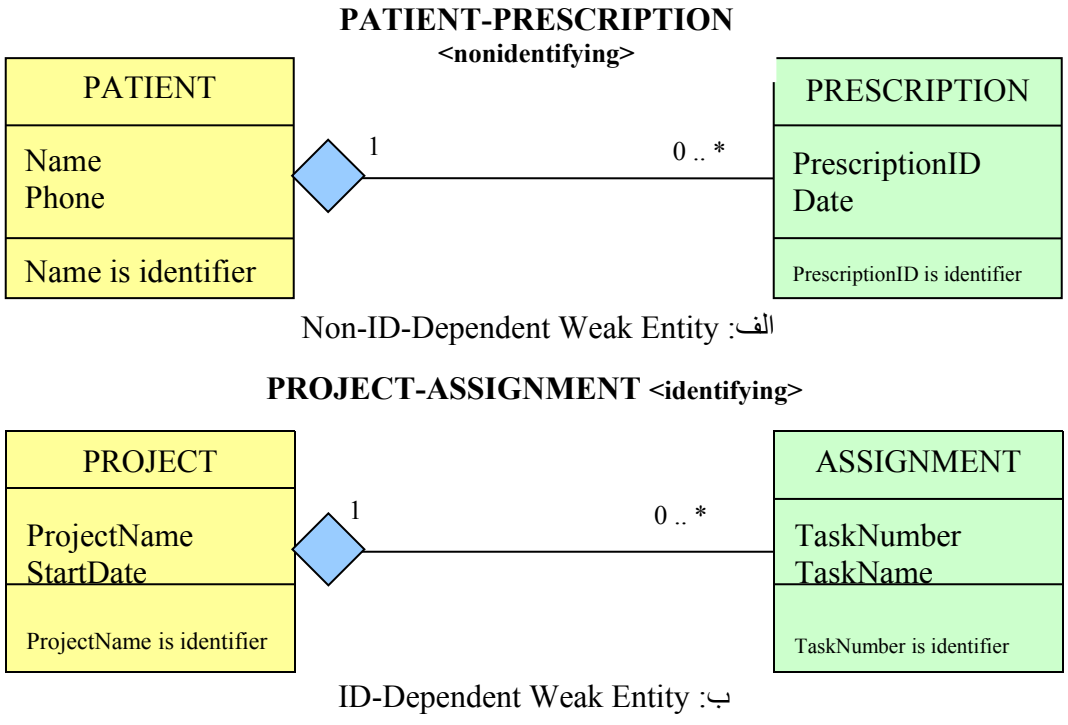
شکل(27) دیاگرام های مودل UML در دیتابیس

Relationshipها توسط یک خط اتصالی بین Entityها نشان داده میشوند. کاردینالتي به شکل 'x .. y' نشان داده میشود. طوریکه 'x' کمترین و 'y' بیشترین کاردینالتي یک Entity را در Relationship نشان میدهد. طور مثال Entity با داشتن سمبول '0 .. 1' میتواند حد اقل "صفر" و حد اکثر "یک" سهم در Relationship داشته باشد. بهمین ترتیب سمبول '1 .. *' یک Entity را با حد اقل "یک" و حد اکثر چندین نشان میدهد. اجزاء "الف"، "ب" و "ج" شکل(27) Relationshipهای '1:1'، 'N:1' و 'N:M' را در دیزاین UML نشان میدهد.

نوت: در دیاگرام های دیتابیس نوع UML و یا هم دیاگرام های عادی E-R، کلیدهای خارجی (Freign-Key) نشان داده نمیشوند. فقط کلیدهاي خارجي بعد از تبدیل نمودن مودل E-R به Relational-Model تعیین میشوند.

نمایش Weak-Entityها در UML

شکل (28) Entityهای ضعیف را در مودل UML نشان میدهد.



شکل(28) دیاگرام های مودل UML

یک شکل مربعی رنگ شده بالای خط Relationship در کنار Strong-Entity ترسیم شده است. در جزء "الف" شکل بالا PRESCRIPTION یک Entity ضعیف بوده و PATIENT عبارت از Entity قوی میباشد. طوریکه هر Entity ضعیف مربوط به Entity قوي است، همین Entity قوي بنام والد (Parent) Entity ضعیف یاد میشود. چون همیشه کاردینالتي سمت والد یا عبارت

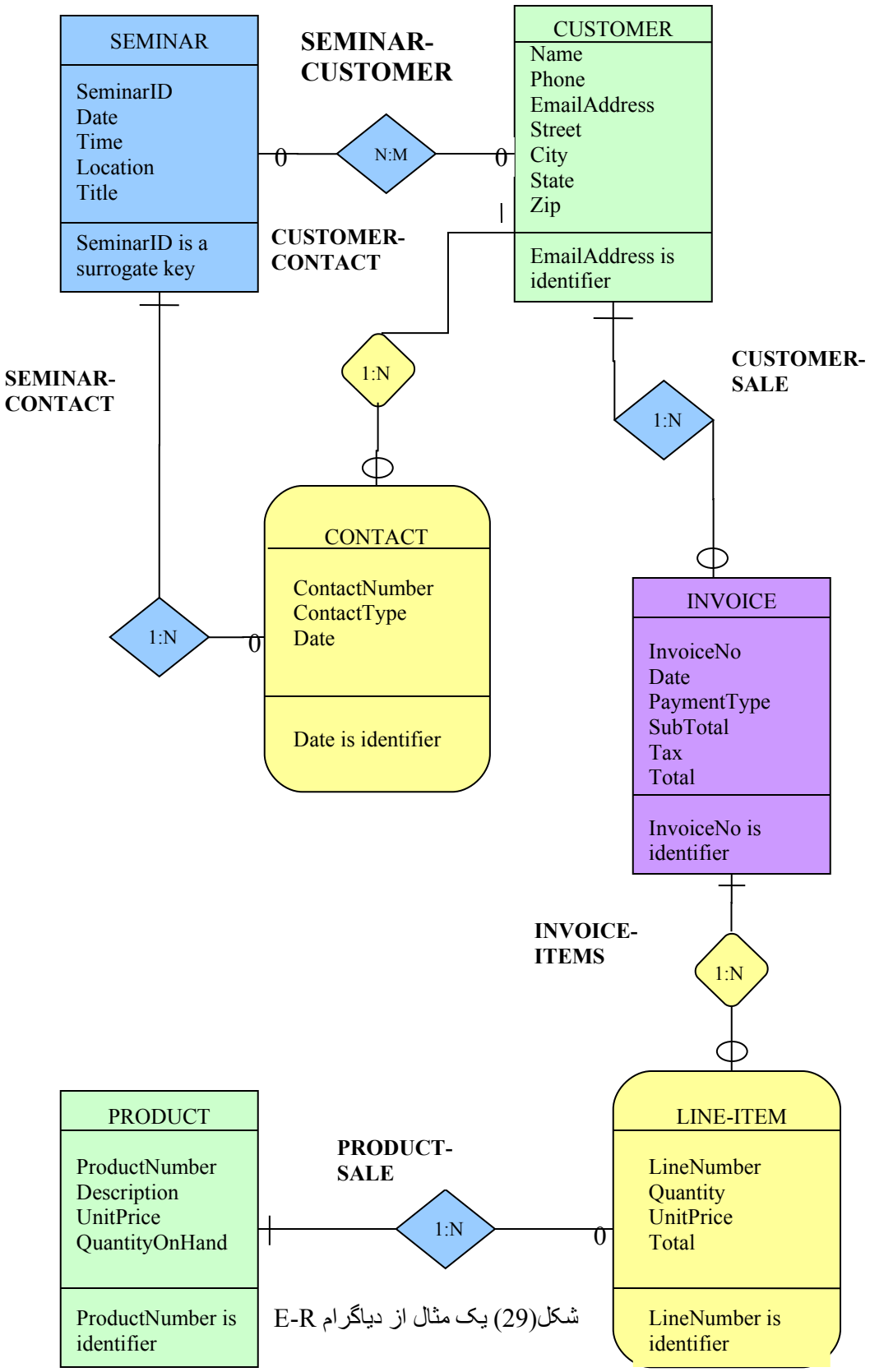
از '1 . . 1' است، پس بصورت ساده این کاردینالتي به شکل "1" نشان داده میشود.

بهمین ترتیب، چون Entity شکل(28-الف) ، ID-Dependent نیست، پس در کنار نام Relationship این موضوع به شکل < nonidentifying> نشان داده شده است. برعکس، شکل(28-ب) یک ID-Dependent Entity را نشان میدهد و در کنار نام Relationship یاد شده کلمه <identifying> نشان داده شده است.

بحث در مورد دیاگرام های دیتابیس در UML در همین قسمت خاتمه داده شده، به امید اینکه استفاده کنندگان نوت، در آینده قادر به تفکیک و استفاده از این نوع دیاگرام ها و مودل هاي دیتابیس شده بتوانند.

در قسمت باقیمانده نوت از دیاگرام های ستندرد E-R استفاده میشود.

برای آشنایی با نمونه اي از یک دیتا مودل E-R، شکل(29) ، که برای یک کمپنی دیزاین شده است و در آن Entity-Classها ، مشخصه ها ، Entityهاي قوي ، Entityهاي ضعیف، کاردینالتي ها و Relationshipها نشان داده شده اند، مرور شود.



شکل(29) یک مثال از دیاگرام E-R

خصوصیات مشخصه های ، Entityهای مختلف شکل بالا در جدول شکل (30) نشان داده شده است.

Entity Name	**Attribute Name**	**Data Format (Length)**	**Required?**	**Default Value**	**Input Mask**	**Remarkable Constraints**
SEMINAR	SeminarID	Autonumber	Yes	DBMS supplied		Surrogate
SEMINAR	Date	Date	Yes	None	mm/dd/yyyy	09/01/2003
SEMINAR	Time	Text(4)	No	None	hh(AM/PM)	
SEMINAR	Location	Text(7)	Yes	None	None	
SEMINAR	Title	Text(35)	Yes	'Intro'	None	
CUSTOMER	Name	Text(35)	Yes	None	None	
CUSTOMER	Phone	Text(10)	No	None	(nnn)nnn-nnn	
CUSTOMER	EmailAddress	Text(50)	No	None		
CUSTOMER	…					

شکل (30) خصوصیات (Property) مشخصه های شکل (29)

Validating the Data-Model

بعد از تکمیل دیزاین دیتا مودل، باید عملیه Validation یا ارزیابي نهایي مودل متذکره صورت گیرد]4، ص. 104[. در نتیجه اي نظریه های مختلف شاید ضرورت افتد تا بعضی Entityها به اجزاء کوچکتر تجزیه شوند و Entityهای جدید ایجاد شوند. برای اجراء این کار باید با استفاده کننده گان دیتابیسی که ایجاد میشود مشوره صورت گیرد. ممکن تمام استفاده کننده گان به دیتا مودل E-R بلدیت نداشته باشند، پس برای سهولت کار، همین دیتا مودل اگر به متن تبدیل شود و بعدا با استفاده کننده گان در میان گذاشته شود، نتیجه بهتر خواهد داشت. طور مثال یک اطاق برای یک Customer است، وقتی این موضوع، که نشان دهنده کمترین کاردینالتي "یک" است، با استفاده کننده دیتابیس در میان گذاشته شود و استفاده کننده بدون اینکه مفهوم Minimum-Cardinality را بداند، میپرسد "پس Customer دومی ام بکجا رود؟"، بصورت واضح فهمیده میشود که در اینجا به Minimum-Cardinality اضافه از یک ضرورت است.

پروسه ارزیابي نهایي (Validation) دیتا مودل، یک پروسه بسیار مهم و پر ارزش است]4، ص. 104[. در این مرحله، تبدیل ویا تغییر آوردن در ساختمان دیتابیس بدون کدام مصرف، بصورت ساده و آسان صورت گرفته میتواند، در حالیکه بعدتر شاید برای آوردن یک تغیر عادی در دیتابیس، ساختمان دیتابیس ضرورت به کار بسیار

در از قبیل ایجاد جداول جدید، ایجاد Relationshipهای جدید، طرح قواعد تازه و غیره، را ایجاب نماید. این کار باعث ضیاع وقت و مصرف بیشتر میشود. حتي در بعضي موارد ضرورت به افراد با تجربه بیشتر در ساحه پروگرام سازي خواهد بود.

دیزاین دیتابیس

در این قسمت، پروسه های تبدیل دیتا مودل E-R به دیزاین Relational، یا یک مرحله دیگر در دیزاین دیتابیس، شرح میشود]4، ص. 109[.

در جریان این درس، نمایش سه نوع Relationshipها (1:N ، 1:1, N:M) و استفاده کلیدهاي خارجي (Freign-Key) در ایجاد این نوع Relationshipها نشان داده شده است. در اخیر این درس، اکثر مسایلي که بخاطر دیزاین دیتابیس ها استفاده مي شوند، توضیح شده و مثال های از آنها کار خواهد شد.

نمایش Entityها در Relational-Model

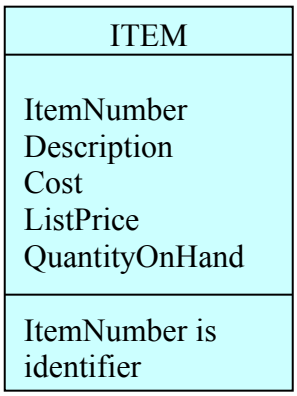
نشان دادن یک Entity-Class به شکل مودل Relational یک پروسه ساده مي باشد. در این پروسه نام هر Entity به حیث

نام Relation (جدول) استفاده میشود و هر مشخصه یک ستون را در Relation نشان میدهد]4، ص. 110[.

برای تعیین نمودن کلید اولیه (Primary-Key) یک جدول، مشخصه یا مشخصه هائیکه به حیث Identifier در Entity تعیین شده اند، استفاده میشوند. بالاخره، پروسه های نارمل سازي بالای جدول تطبیق شده، و جدول آماده دیزاین و استفاده میشود.

مثال های ذیل در نظر گرفته شوند:

-یک Entity بنام ITEM در شکل(31) نشان داده شده است.



شکل(31) The ITEM Entity

طوریکه دیده میشود مشخصه های این Entity-Class عبارت اند از ItemNumber، Description، Cost، ListPrice و QuantityOnHand.

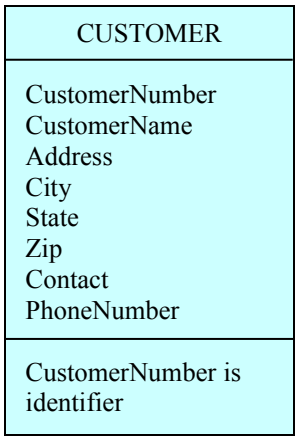
Entity-Class بالا به یک مودل Relational (جدول) تبدیل شده میتواند، طوریکه مشخصه ها هرکدام ستون های جدول ITEM را نشان دهند. ItemNumber که عبارت از Identifier است، بحیث کلید اولیه در جدول جدید تنظیم شود. در نتیجه:

ITEM (ItemNumber, Description, Cost, ListPrice, QuantityOnHand)

طوریکه از درس های گذشته میدانیم، ستون Primary-Key جدول بالا بشکل Underline نشان داده شده است.

جدولیکه به این شکل ترتیب داده شود، امکان دارد دارای مشکل Normalization باشد، بنا پروسه های نارمل سازي باید بالای آن تطبیق شود. فرض شود ستون Primary-Key یعنی (Identifier) جدول بالا دارای دیتای غیر تکراری بوده دیترمنانت متباقی ستون های جدول است و کدام Functional-Dependency دیگر هم در آن وجود ندارد. پس گفته میتوانیم که جدول ITEM، نارملایز است.

-در مثال دیگر، Entity بنام CUSTOMER در شکل (32) نشان داده شده است و در آن احتمال وجود مشکل Normalization نیز هست.



شکل (32) The CUSTOMER Entity

اگر این Entity به مودل Relational تبدیل شود، چنین خواهد بود:

CUSTOMER (CustomerNumber, CustomerName, Address, City, State, Zip, Contact, PhoneNumber)

ستون CustomerNumber عبارت از Primary-Key است. جهت تطبیق پروسه های نارمل سازي باید Functional-Dependencyها در این Relation تشخیص داده شوند.

یک Functional-Dependency در این جدول عبارت است از:

Zip → (City, State)

ستون Zip در جدول بالا Candidate-Key نیست. پس جدول نارملایز نبوده دارای مشکل میباشد.

همچنان یک Functional-Dependency احتمالی دیگر در قسمت PhoneNumber نیز وجود دارد. احتمالی به این خاطر، که نمبر تلفون یا مربوط کمپنی که شخص در آن کار میکند است و یا مربوط خود شخص میباشد. بخاطر معلوم کردن این موضوع، باز هم باید استفاده کنندگان پرسیده شوند. درحالت اول Functional-Dependency وجود ندارد، اما در حالت دوم Functional-Dependency وجود دارد و آن عبارت است از:

Contact → PhoneNumber

فرضا حالت دوم درست باشد، یعنی نمبر تلفون مربوط کمپنی شخص باشد.

مطابق مرحله سوم نارمل سازي، ستون های مربوط هر Functional-Dependency به یک جدول جدید انتقال میشوند. دیترمنانت هر Functional-Dependency بحیث Primary-Key جدول جدید تعیین شده و یک کاپی آن در جدول اول بحیث Freign-Key باقی میماند. یعنی

CUSTOMER (CustomerNumber, CustomerName, Address, *Zip*, *Contact*)

ZIP (Zip, City, State)

CONTACT (Contact, PhoneNumber)

با قواعد RIC ذیل:

-عناصر ستون Zip در جدول CUSTOMER باید شامل ستون Zip در جدول ZIP باشند.

-عناصر ستون Contact در جدول CUSTOMER باید شامل ستون Contact در جدول CONTACT باشند.

این سه جدول فعلا نارملایز بوده و دارای کدام مشکلی نمیباشند و دیتابیس نیز ایجاد شده میتواند.

نوت: بعضا ضرورت می افتد تا یک جدول یا جداولی بحالت نارملایز نه، بلکه به حالت Denormalize باقی بمانند]4، ص. 111[. **Denormalization** طریقه است که توسط آن محتواء یک جدول بصورت واضح تر و بهتر نشان داده میشود. در مثال بالا اگر Functional-Dependency

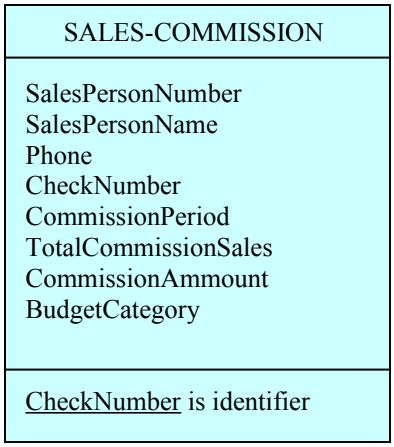
Zip → (City, State)

در نظر گرفته نشده و ستون های آن بصورت عادی در جدول باقی بمانند، شاید مطلب واضحتر ارائیه شود و امکان غلطی یا تکرار Data آنقدر محسوس نباشد.

در این حال جدول بالا بشکل ذیل باقی خواهد ماند:

CUSTOMER (CustomerNumber, CustomerName, Address, City, State, Zip, *Contact*)

-مثال بعدی یک Entity را بنام SALES-COMMISSION در شکل (33) نشان میدهد.



شکل (33) The SALES-COMMISSION Entity-Class

طوریکه قبلا توضیح داده شده است، برای تبدیل نمودن یک Entity-Class به مودل Relational، نام Entity-Class را نام Relation قرار داده، هر مشخصه ، یک ستون یا Field در جدول شده و Identifier آن بحیث Primary-Key جدول تنظیم میشود. بعدا دیده میشود که آیا این جدول نارملایز است و یا خیر؟ در صورت ضرورت، مراحل نارمل سازي بالای آن تطبیق شده و قواعد Referential Integrity Constraint (RIC) برای جدول های جدید تعین میگردند. پس

SALES-COMMISSION (SalesPersonNumber, SalesPersonName, Phone, CheckNumber, CheckDate, CommissionPeriod, TotalCommissionSales, CommissionAmmount, BudgetCategory)

در جدول بالا Functional-Dependencyهای ذیل وجود دارند:

SalesPersonNumber → (SalesPersonName, Phone, BudgetCategory)

و

(SalesPersonNumber, CommissionPeriod) → (TotalCommissionSales, CommissionAmmount)

حال برای هر Functional-Dependency یک Relation جدید ایجاد مینماییم. یعنی

SALESPERSON (SalesPersonNumber, SalesPersonName, Phone, BudgetCategory)

SALES (SalesPersonNumber, CommissionPeriod, TotalCommissionSales, CommissionAmmount)

جدول اولی شکل ذیل را بخود میگیرد:

SALES-COMMISSION (*SalesPersonNumber*, CheckNumber, CheckDate, *CommissionPeriod*)

قاعده RIC برای جداول بالا عبارت است از:

-عناصر ستون *SalesPersonNumber* در جدول SALES-COMMISSION باید شامل ستون SalesPersonNumber در جدول SALESPERON باشند.

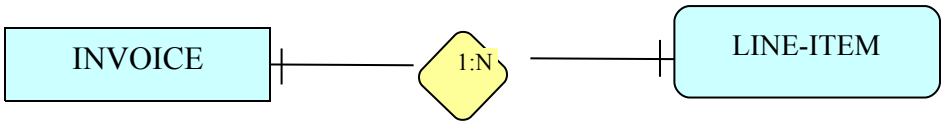
-عناصر ستون های (*CommissionPeriod, SalesPersonNumber*) در جدول SALES-COMMISSION باید شامل ستون های (CommissionPeriod, SalesPersonNumber) در جدول SALES باشند.

فعلا جداول بالا نارملایز میباشند.

نمایش Entityهای ضعیف در Relational-Model

طریقه که قبلا توضیح گردید، برای تمام Entityها، به شمول Entityهای ضعیف، قابل تطبیق است، اما برای ID-Dependent Weak Entityها یک نکته مهم باید در نظر گرفته شود و آن اینکه Primary-Key مربوط Strong Entity باید جزء Weak Entity، Primary-Key باشد [4، ص. 114].

شکل (34) این نوع Relationship را نشان میدهد.



(الف) مثال یک ID-Dependent Weak Entity

(LINE-ITEM (LineNumber, Qty, ItemNumber, Description, Price, ExtPrice

(ب) نمایش جدول LINE-ITEM با Primary-Key ناقص

LINE-ITEM (InvoiceNumber, LineNumber, Qty, ItemNumber, Description, (Price, ExtPrice

(ج) نمایش جدول LINE-ITEM با Primary-Key کامل

شکل (34) نمایش مودل Relational برای ID-Dependent Weak Entityها

در شکل با لا LINE-ITEM یک Entity ضعیف است، بخاطریکه موجود بودن آن مربوط به INVOICE میباشد. یعنی در صورت موجود نبودن INVOICE، محتوا جدول LINE-ITEM وجود داشته نمیتواند. برعلاوه LINE-ITEM یک ID-Dependent Weak Entity است، بخاطریکه Identifier جدول INVOICE همچنان مکمل Identifier جدول LINE-ITEM است.

با درنظرداشت اصل یاد شده در مورد ID-Dependent Weak Entity، اگر خواسته شود تا Relationship بین جداول INVOICE و جدول LINE-ITEM بصورت Relational-Model نشان داده شود، Primary-Key جدول INVOICE که عبارت از

InvoiceNumber است، جزء Primary-Key جدول LINE-ITEM قرار داده میشود (شکل 34-ج). در این قسمت سوالي پیدا مي شود: اگر احیانا این کار اجراء نشود در (شکل 34-ب) چی مشکلی ایجاد خواهد شد؟ - جواب واضح است و آن اینکه خواهی نه خواهی در جدول LINE-ITEM سطرهای تکراری بمیان خواهند آمد (طور مثال اگر دو Invoice برای عین جنس یا Item دارای عین مقدار یا Quantity و در عین قطار باشد، تکرار سطر یا Record صورت میگیرد.)

نمایش Relationshipها در مودل Relational

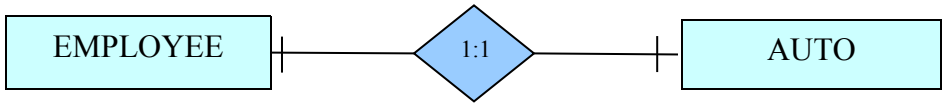
طوریکه تبدیل نمودن Entity-Classها از مودل E-R به مودل Relational توضیح گردید، تبدیل کردن Relationshipها نیز از مودل E-R به Relational-Model ضرورت به توضیح دارند.

شکل هائیکه در مود ل E-R برای نمایش دادن Relationshipهای باینری (درجه 2) استفاده میشدند عبارت بودند از: 1:1 ، N:1 و N:M. تعین نمودن این حالات مستقیما ارتباط داشتند به کاردینالتي اصغری و اعظمی Relationshipها.

موارد یاد شده هر کدام به ترتیب در مودل Relational نشان داده و توضیح میشوند.

نمایش Relationshipهای یک به یک (1:1)

ساده ترین شکل یک Relationship باینری عبارت از (1:1) است که در آن Entity-Instance از یک نوع به حد اعظمی "یک" با Entity-Instance از نوع دیگر ارتباط برقرار نموده میتواند]4، ص. 114[. مثال آن در شکل(35) نشان داده شده است. به اساس این دیاگرام، یک Employee میتواند یک موتر داشته باشد و یک موتر تنها به یک Employee ارتباط داشته میتواند.



شکل(35) مثال یک Relationship از نوع یک به یک(1:1)

وقتی خواسته شود تا یک Relationship یک به یک در مودل Relational نشان داده شود، بصورت ساده Entityها تبدیل شده و کاپي Identifier یکی از Entityها در Entity دیگر به شکل Freign-Key تنظیم میشود. در شکل(36-الف) کلید جدول AUTO در جدول EMPLOYEE به شکل Freign-Key اضافه شده است و در شکل(36-ب)، کلید جدول EMPLOYEE در جدول AUTO به شکل Freign-Key تنظیم و ارتباط برقرار شده است. هر دو حالت ممکن بوده و درست اند. به یک نکته باید دقت شود که همیشه کاپی Primary-Key جدول والد یا Parent در جدول Child به شکل Freign-Key جابجا شده و مطابق قاعده دیتابیس عناصر جدول Child ست فرعی عناصر جدول Parent میباشند.

EMPLOYEE (EmployeeNumber, EmployeeName, Phone, …, *LicenseNumber*)
AUTO (LicenseNumber, SerialNumber, Color, Make, Model, …)

قاعدهء RIC عبارت است از:

عناصر ستون *LicenseNumber* در جدول EMPLOYEE باید شامل ستون LicenseNumber در AUTO باشند

(الف) جابجاسازی کاپي کليد جدول AUTO در جدول EMPLOYEE

EMPLOYEE (EmployeeNumber, EmployeeName, Phone, …)
AUTO (LicenseNumber, SerialNumber, Color, Make, Model, …, *EmployeeNumber*)

قاعدهء RIC عبارت است از:

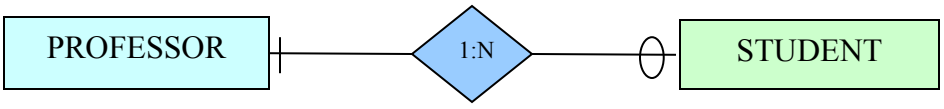
- عناصر ستون EmployeeNumber در جدول AUTO باید شامل ستون EmployeeNumber در EMPLOYEE باشند.

(ب) جابجاسازی کاپي کليد جدول EMPLOYEE در جدول AUTO

شکل (36) نشان دادن طریقه های مختلف و ممکنه برای ایجاد Relationshipهای یک به یک (1:1)

نمایش Relationshipهای یک به چندین (N:1)

نوع دوم Relationship باینری عبارت از 'N:1' یا یک به چندین میباشد. بدین مفهوم که یک Entity-Instance از یک نوع به چندین Entity-Instance از نوع دیگر ارتباط داشته مي تواند. یا یک عنصر از جدول اول به چندین عنصر از جدول دوم مربوط شده میتواند. شکل(37) یک Relationship از نوع یک به چندین (N:1) را نشان میدهد. در این شکل دیتا در مورد پروفیسور و محصل نشان داده شده است.



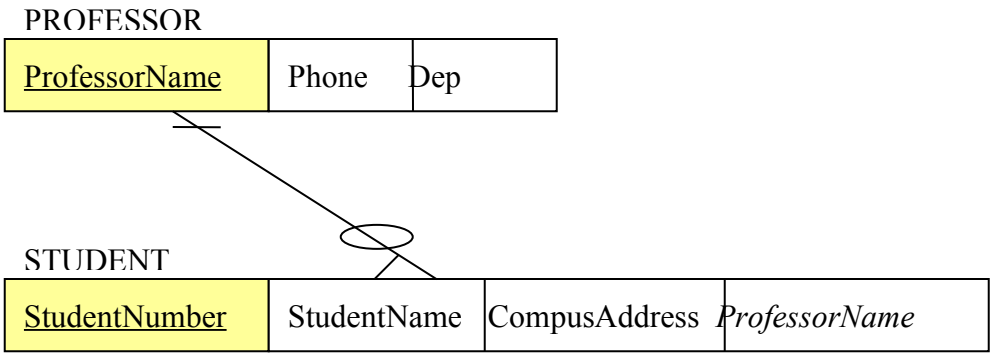
شکل(37) مثالی از یک Relationship یک به چندین (N:1)

طوریکه دیده میشود، یک پروفیسور امکان دارد هیچ شاگرد نداشته باشد، و یا هم میتواند چندین شاگرد داشته باشد. برعکس یک شاگرد تنها و تنها میتواند یک پروفیسور را بحیث راهنما داشته باشد. شرایط ذکر شده توسط سمبول های "بیضوی" و "خط عمودی"، که بالای خط Relationship ترسیم شده اند نشان داده میشود.

اصطلاحات Parent و Child در این نوع Relationshipها نیز معمول بوده و استفاده میشوند]4، ص. 116[. جناحیکه "یک" را در Relationship افاده مینماید، بنام Parent و جناح مقابل آن (چندین)، بنام Child یاد میشود. در مثال

بالا PROFESSOR عبارت از Parent بوده و STUDENT عبارت از Child میباشد.

نمایش دادن Relationshipهای یک به چندین به شکل Relational-Model کار نسبتا ساده است. اولا Entityها معه مشخصه های آنها مکملا توضیح شده و بعدا کاپی Primary-Key جدول Parent در جدول Child بحیث Freign-Key علاوه میشود. شکل(37) دیده شود. کاپي کلید جدول Parent یا جدول PROFESSOR در جدول Child یا جدول STUDENT اضافه شده و به شکل Freign-Key نشان داده میشود. همین کار در شکل(38) در دیاگرام ساختمان دیتا نشان داده شده است.



قاعدهء RIC عبارت است از:

عناصر ستون *ProfessorName* در جدول STUDENT باید شامل ستون ProfessorName در جدول PROFESSOR باشند.

شکل(38) نمایش مودل Relational جدول PROFESSOR و STUDENT از شکل(37)

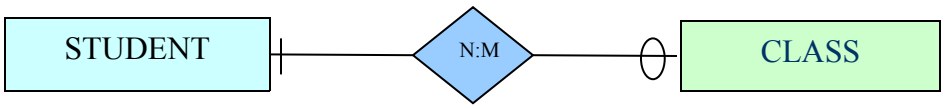
نکته اساسی که در Relationshipهای N:1 و 1:1 باید مراعات شود اینست که در هر صورت باید کاپی کلید جدول Parent در جدول Child بحیث Freign-Key اضافه شود. در Relationshipهاي يك به چندين مسله زياد پيچيده نبوده فقط كاپي كليد جناح يك به جناح چندين اضافه شده و پروسس مي شود. اما در Relationshipهای 1:1 تعين جدول هاي Parent و Child يك اندازه مشكل بوده به دقت و معلومات بيشتر ضرورت دارد. زمانيكه Relationship يك به يك در ديتابيس طرح ميشود در اين قسمت بايد معلومات اضافه و كامل از استفاده كنندگان ديتابيس دريافت شده و بعداً تصميم گرفته شود. هر Entity كه در Relationship يك به يك قويتر است و اولاً ديتا به آن داخل ميگردد، همان Entity بايد Parent باشد و كاپي كليد آن به Entity مقابل در Relationship اضافه شده و به جدول بالايي (Parent) ريفرنس داده شود.

احياناً اگر اين اصل مراعات نشود و كاپی كليد جدول Child در جدول Parent گذاشته شود، چی اتفاقی خواهد افتاد؟ - (اگر كاپی StudentNumber، كه كليد اوليه جدول Child است، در جدول PROFESSOR گذاشته شود) در تعريف Relation توضيح شده است كه هر مشخصه صرف ميتواند يک جزء ديتا برای يک Record داشته باشد. يعنی در آنحالت برای يک پروفيسور ميشود ID تنها يک شاگرد تعين شود و اين خود تضاد با اصل يک به چندين است كه در آن يک پروفيسور ميتواند چندين شاگرد داشته باشد. پس يگانه راه، همانا جابجا سازی كاپی كليد جدول جناح "يک" در جدول جناح "چندين" ميباشد.

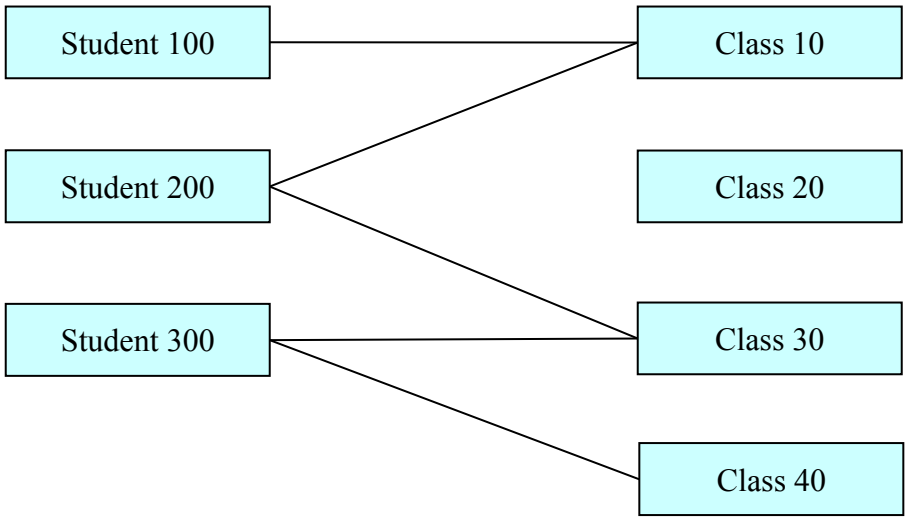
نمایش Relationshipهای چندین به چندین (N:M)

نوع دیگر Relationship باینری عبارت از N:M است و آن طوریست که یک Entity-Instance از نوع اول به چندین Entity-Instance از نوع دوم ارتباط گرفته مي تواند و بعین ترتیب یک Entity-Instance از نوع دوم به چندین Entity-Instance از نوع اول ارتباط گرفته میتواند.

شکل(39-الف) دیاگرام E-R یک Relationship چندین به چندین را بین STUDENT و CLASS نشان میدهد. یک شاگرد میتواند چندین صنف بگیرد و یک صنف میتواند چندین شاگرد داشته باشد. اشتراک هر کدام از Entity-Instanceها در Relationship اختیاری است. یعنی کاردینالتي اصغری صفر است. یک صنف میتواند هیچ شاگرد نداشته باشد و یا یک صنف میتواند چندین شاگرد داشته باشد و یک شاگرد میتواند هیچ صنف نگیرد و یا یک شاگرد میتواند چندین صنف بگیرد. شکل(39-ب) وضاحت بیشتر را در مورد افاده مینماید.



(الف) دیاگرام E-R نشاندهندهء Relationship بین جداول STUDENT و CLASS



(ب) نمونه دیتا بین سطرهای جدول هاي STUDENT و CLASS

شکل(39) مثالی از یک (N:M Relationship)

Relationshipهای چندین به چندین، مانند Relationshipهای یک به یک و یک به چندین، بصورت مستقیم نمایش داده نمی شوند. بخاطر توضیح بهتر موضوع، اگر کوشش شود تا از میتود جابجا سازی مستقیم کلید از یک جدول به جدول دیگر، مانند Relationshipهای 1:1 و N:1، استفاده شود، چی نتیجه اي ببار خواهد آورد؟ - اولاً هر دو Relation معه مشخصه های آن تعین گردند. بعد از آن کاپی کلید STUDENT که عبارت از StudentNumber است، در جدول CLASS بحیث Freign-Key جابجا شود. چون بودن چندین جزء دیتا در

یک Cell با شرایط بودن Relation مغایرت دارد. بنا شرط بودن چندین شاگرد در یک صنف نمیتواند عملی گردد و تنها یک شاگرد میتواند برای یک صنف تعین گردد.

بعین ترتیب اگر کوشش شود تا کاپی Primary-Key جدول CLASS در جدول STUDENT بحیث Freign-Key جابجا شود، همین مشکل ایجاد خواهد شد و یک شاگرد صرف در یک صنف شامل خواهد شد.

جهت رفع این مشکل، باید یک جدول یا Relation سومی ایجاد شود که نمایانگر همین Relationship باشد. شکل (40-الف) جدول STU-CLASS را نشان میدهد. نمونه دیتا جداول سه گانه وارتباط بین آنها در شکل (40-ب) نشان داده شده است. جدول تقاطع یا Intersection دارای یک سطر برای هر خط بین جداول STUDENT و CLASS در شکل (39-ب) میباشد.

STUDENT (StudentNumber, StudentName)

CLASS (ClassNumber, ClassName)

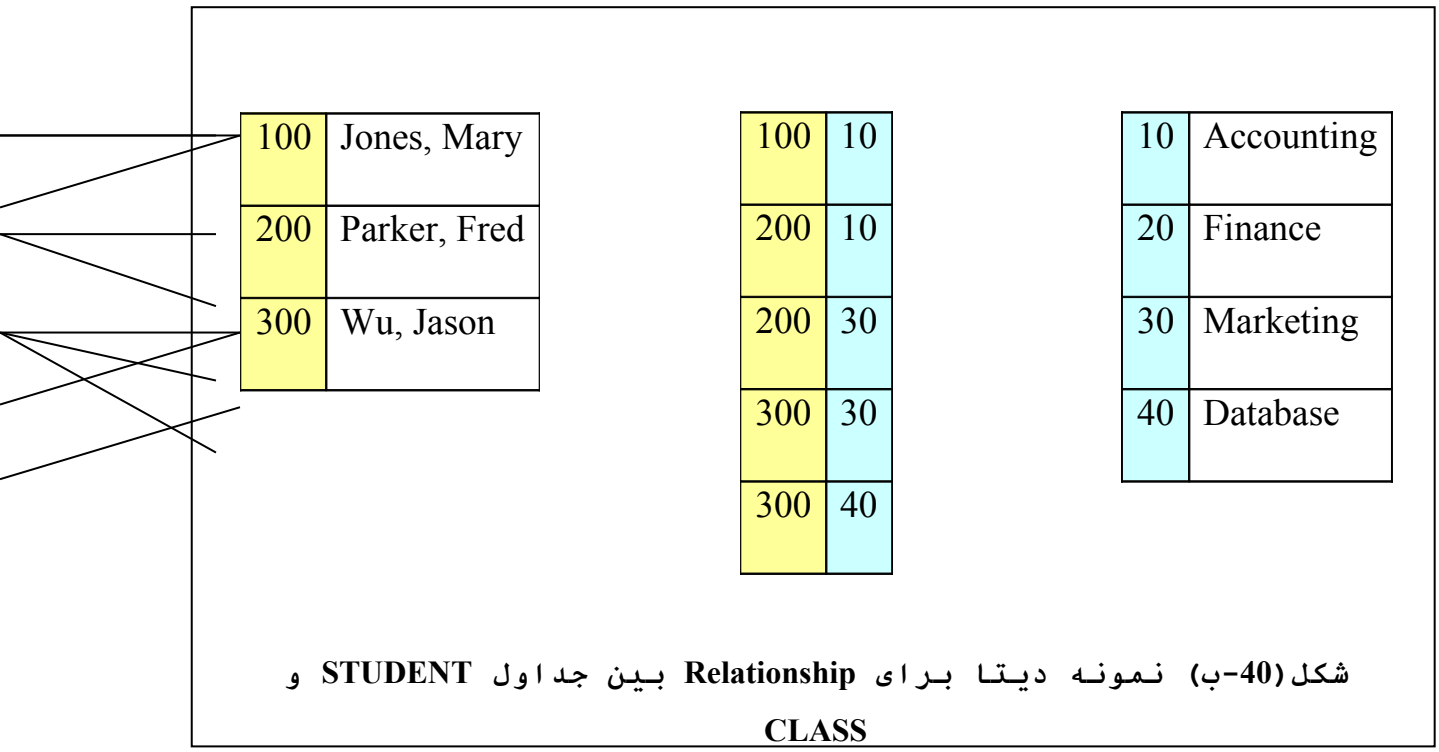
STU-CLASS (*StudentNumber*, *ClassNumber*)

قاعده RIC عبارت است از:

-عناصر ستون ClassNumber در جدول STU-CLASS باید شامل ستون ClassNumber در جدول CLASS باشند.

-عناصر ستون StudentNumber در جدول STU-CLASS باید شامل ستون StudentNumber در جدول STUDENT باشند.

شکل(40-الف) جدول هاییکه برای نمایش Relationship بین STUDENT و CLASS ضروری اند.



100	Jones, Mary
200	Parker, Fred
300	Wu, Jason

100	10
200	10
200	30
300	30
300	40

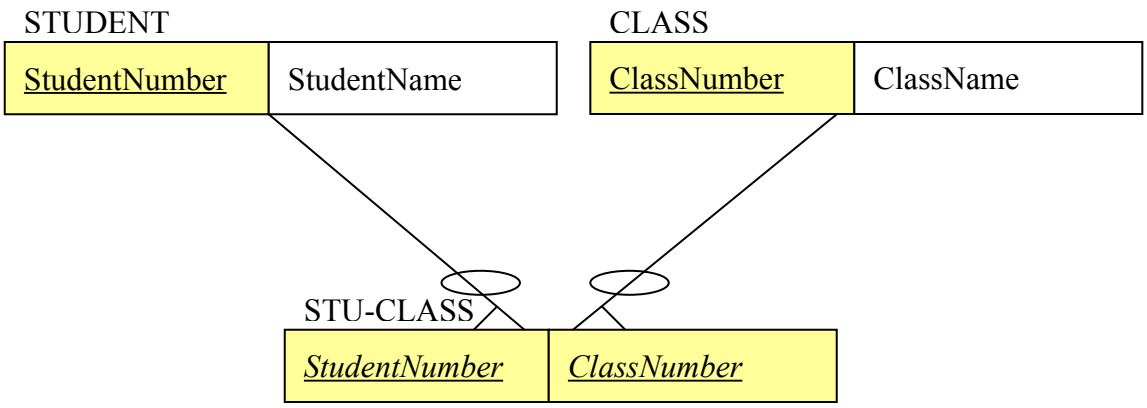
10	Accounting
20	Finance
30	Marketing
40	Database

شکل(40-ب) نمونه دیتا برای Relationship بین جداول STUDENT و CLASS

شکل(40) نمایش یک Relationship چندین به چندین (N:M)

دیاگرام ساختمان دیتا (Data Structure) برای Relationship بالا در شکل(41) نشان داده شده است[4، ص. 119، 120].

Relationship از جدول CLASS به جدول تازه ایجاد شده ای STU-CLASS عبارت از 'N:1' است. بهمین ترتیب Relationship از جدول STUDENT به جدول STU-CLASS نیز '1:N' میباشد. یعنی



شکل(41) **دیاگرام دیتا سترکچر برای** Relationship **بین جدول هاي** STUDENT **و** CLASS

طوریکه دیده میشود، کار اجراء شده عبارت از تقسیم کردن یک Relationship چندین به چندین (N:M) به دو Relationship یک به چندین (N:1) میباشد. Primary-Key جدول جدید به شکل ترکیبی متشکل از Primary-Keyهای جداول قبلی است. یعنی (StudentNumber, ClassNumber).

Primary-Key یک Intersection-Relationship همیشه از Primary-Keyهای جداول بالایي (Parent) تشکیل میشود.

مفاهيم پيشرفته در مودل معلومات اوليه E-R

Advanced Concepts of the E-R Data Model

مفاهيم و اشكال ابتدايي مودل E-R جهت ديزاين ديتابيس ها در اكثر موارد استفاده مي شوند. در بعضي موارد موضوع ديزاين ديتابيس ها اضافه تر پيچيده بوده و ضرورت مي افتد تا ميتودهاي پيشرفته ديزاين به كار برده شوند. در اين قسمت بحث بالاي ميتودهاي عمومي سازي (Generalization) و ارتباط هاي با درجه بالاتر از دو، يعني نمايش عمومي سازي و نمايش ارتباط هاي با درجه هاي بالا در مودل معلومات اوليه E-R تمركز مي يابد.

عمومي سازي (Generalization)

تحت عنوان عمومي سازي شكل هاي Superclassها و Subclassها در ديا گرام اضافه مي شوند. اين شكل شامل سلسله به ميراث بردن مشخصه ها و ارتباط ها مي باشد. بدين مفهوم كه Subclassها از Superclassها، مشخصه ها و ارتباط ها (Relationships) را به ميراث مي برند و مشخصه ها بصورت عمومي در Superclassها نشان داده مي شوند.

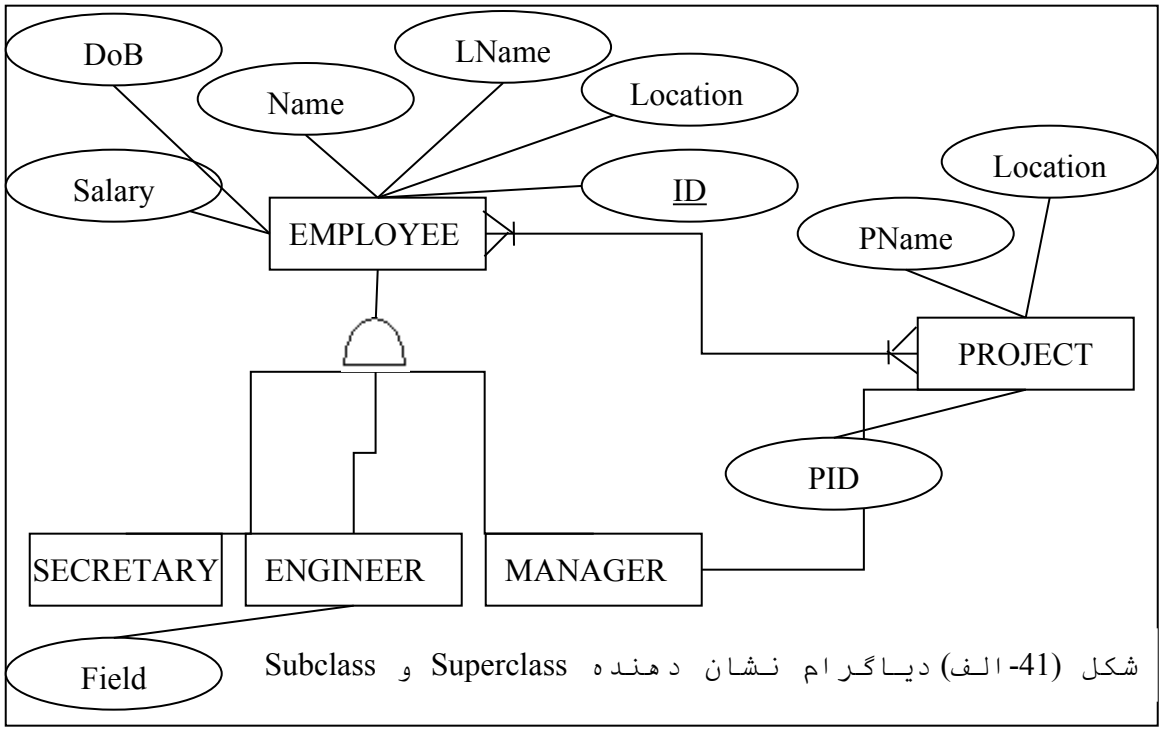
طور مثال کارمندان یک کمپني شامل شکترها، انجنیران و مدیران است. تمام کارمندان داراي شش مشخصه مشابه اند که عبارت اند از: اسم، تخلص، موقعیت، تاریخ تولد، معاش و نمبر شناسایي (ID). تمام کارمندان در پروژه هاي همین کمپني کار مي کنند. تنها انجنیران داراي یک مشخصه اضافه اند که نشان دهنده نوع سند فراغت و تخصص شان مي باشد. به عین ترتیب مدیران نیز یک مشخصه اضافه دارند و آن عبارت از اداره یک پروژه مي باشد. شکل (41-الف) نتیجه دیاگرامي است که شکل بالا را توضیح مي کند.

در این مثال "کارمند" عبارت از Superclass است و Subclassها عبارت از "سکرتر"، "انجنیر" و "مدیر" مي باشد. طوریکه قبلاً نیز یادآوري شد، Subclassها مشخصه ها را از Superclassها به ارث مي برند، در این مثال مشخصه هاي "کارمند" (اسم، تخلص و غیره) به "سکرتر" نیز صدق مي نماید. بعین ترتیب ارتباط هاي "کارمند" به "سکرتر"، "انجنیر" و "مدیر" نیز اطلاق مي گردد.

Subclassها اشکال خصوصي یا (Specialization) از Superclass مي باشند. در مثال بالا، "انجنیر" تمام مشخصه ها و ارتباط هاي "کارمند" را به ارث برده، برعلاوه داراي مشخصه اي بنام "نوع سند" نیز مي باشد. همچنان یک مدیر داراي مشخصه اضافه بنام "اداره پروژه" است.

اشکال مختلف در مودل E-R جهت نمایش دادن Superclassها و Subclassها استفاده مي شوند. در شکل (41-

الف) سمبول نيم دايره "⌓" استفاده شده است. Entityهاي Superclass و Subclass توسط خط ها يا تيرها به هم وصل مي گردند.



شكل (41-الف) ديا گرام نشان دهنده Superclass و Subclass

اگر ضرورت باشد كه يك نمونه اي Entity از Superclass تنها يك مثال Entity براي Sublcass شده بتواند، در اين صورت از سمبول "⊠" بعوض نيم دايره خالي استفاده مي شود و اگر قرار باشد كه يك Superclass بتواند در يك وقت اضافه از يك Subclass شود، در اين صورت از سمبول عادي نيم دايره خالي "⌓" استفاده مي شود.

جهت تعين كردن شكل هاي عمومي سازي (Generalization) از يك متن در مودل ER دو طريقه استفاده مي شود. **اول:**

بالا به پایین (Top-Down) ، **دوم:** پایین به بالا (Bottom-Up).

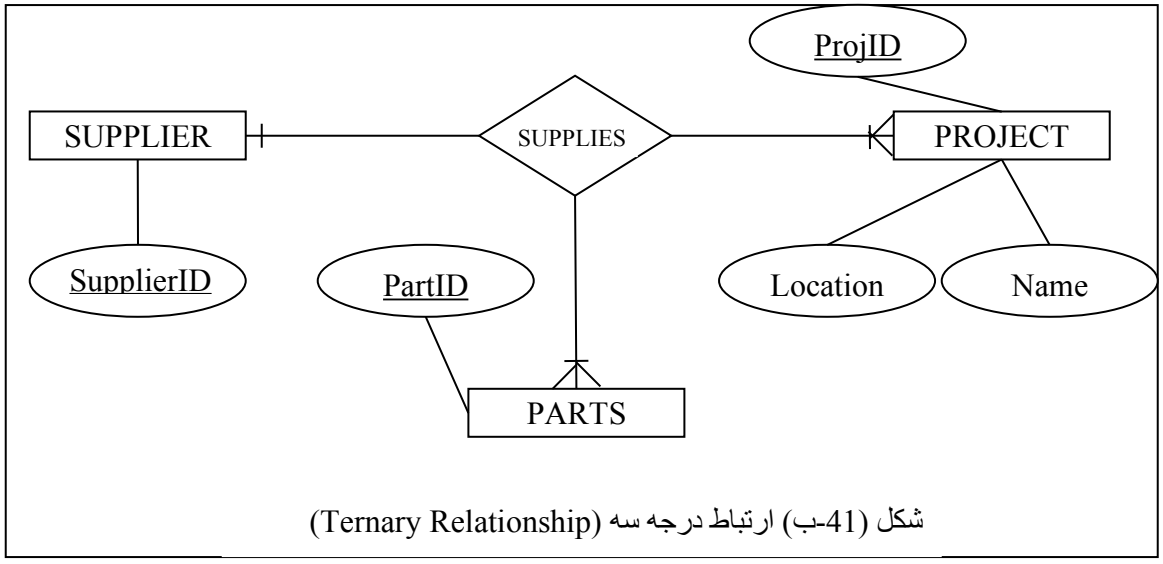
-با استفاده از طریقه اول در جریان مطالعه متن و یا پرسیدن اشخاصي که دیتابیس برایشان دیزاین مي شود، دیزاینر به عنواني برمي خورد که داراي انواع مختلف مي باشد (مانند مثال بالا).

-توسط طریقه دوم در جریان ارزیابي معلوماتیکه دیتابیس به آن دیزاین مي شود، دیزاینر به Entityهاي مشابه برخورد میکند. Entityهاي موجود مي باشند که داراي اکثریت مشخصه هاي مشابه بوده و یا تمام شان به یك یا چند Entity دیگر عین نوع ارتباط (Relationship) برقرار مي نمایند. در صورت مواجه شدن با چنین شرایط تمام Entityها بصورت Subclass قرار داده شده و Entity بحیث Superclass به ایشان دیزاین و به دیاگرام اضافه مي شود.

ارتباط هاي با درجه اضافه از دو

(Relatinships of a Degree Higher Than Two)

درجه ارتباط (Relationship) مربوط به تعداد Entityهاي شامل در آن ارتباط مي باشد. در بخش هاي قبلي ارتباط هاي درجه يك (Recursive) و درجه دو (باينري) مورد بحث قرار گرفته اند. در اين قسمت ارتباط درجه سه (Ternary) توضيح مي شود. در مثال قبلي يك Entity ديگر بنام "وسايل" در نظر گرفته شود و طوري فرض شود كه يك وسيله توسط يك تمويل كننده (Supplier) براي يك پروژه تهيه مي شود. در اين صورت يك ارتباط درجه سه در بين اين سه Entity به ميان آمده است كه در شكل (41-ب) نشان داده شده است.



شكل (41-ب) ارتباط درجه سه (Ternary Relationship)

بصورت عمومي يك ارتباط درجه سه معلومات بيشتري نظر به سه ارتباط درجه دو ارايه مي دارد. در اكثر

موارد ارايه كردن معلومات توسط ارتباط هاي درجه دو ممكن نبوده و يا مشكل مي باشد و كار زيادي ضرورت مي داشته باشد، در حاليكه با استفاده از ارتباط هاي با درجه بالاتر مشكلات حل شده و معلومات بصورت درست ارايه شده مي توانند. به هر صورت، تشخيص دادن اينكه چي وقت و در كدام قسمت ارتباط هاي بالاتر از دو ضرورت است يك اندازه باريكي داشته و در تنظيم نمودن اين نوع ارتباط ها دقت لازم به خرچ داده شود.

در مثال بالا جهت تنظيم كاردينالتي توضيح بيشتري كه ضرورت است عبارت خواهد بود: يك تمويل كننده يك يا چند وسيله را براي يك يا چند پروژه تهيه مي نمايد. در يك پروژه يك يا چند وسيله كه توسط تمويل كننده هاي مختلف تهيه شده اند استفاده مي شود.

مودل Relational ارتباط هاي درجه سه و بالاتر

(Relatinships: Ternary or Higher Degree)

طوريكه ديده شد، در ارتباط هاي درجه دو اگر ارتباط ها از نوع يك به يك (1:1)، يك به چندين (N:1) يا چندين به يك (N:1) بود، كاپي كليد اوليه (Primary-Key) جدول جناح يك در Entity جناح چندين به شكل كليد خارجي (Foreign-Key) اضافه مي شد. تنها در صورت ارتباط هاي چندين به چندين (N:M) يك Entity جديد بين Entityهاي

اولیه ایجاد شده و ارتباط N:M به دو ارتباط N:1 یا 1:1 تجزیه شده و اجرات می شد.

در ارتباط های با درجه بالاتر از دو همیشه یك Entity جدید ضرورت است و کلید اولیه Entity جدید متشکل از کاپی کلیدهای اولیه جناح چندین در ارتباط درجه سه می باشد. به گونه مثال به شکل (41-ب) توجه شود.

نتیجه مثال یاد شده در مودل رابطه ای (Relational-Model) طور ذیل خواهد بود:

SUPPLIES (*SupplierID*, *ProjectID*, *PartID*)

قاعده RIC برای مودل بالا طور ذیل می باشد:

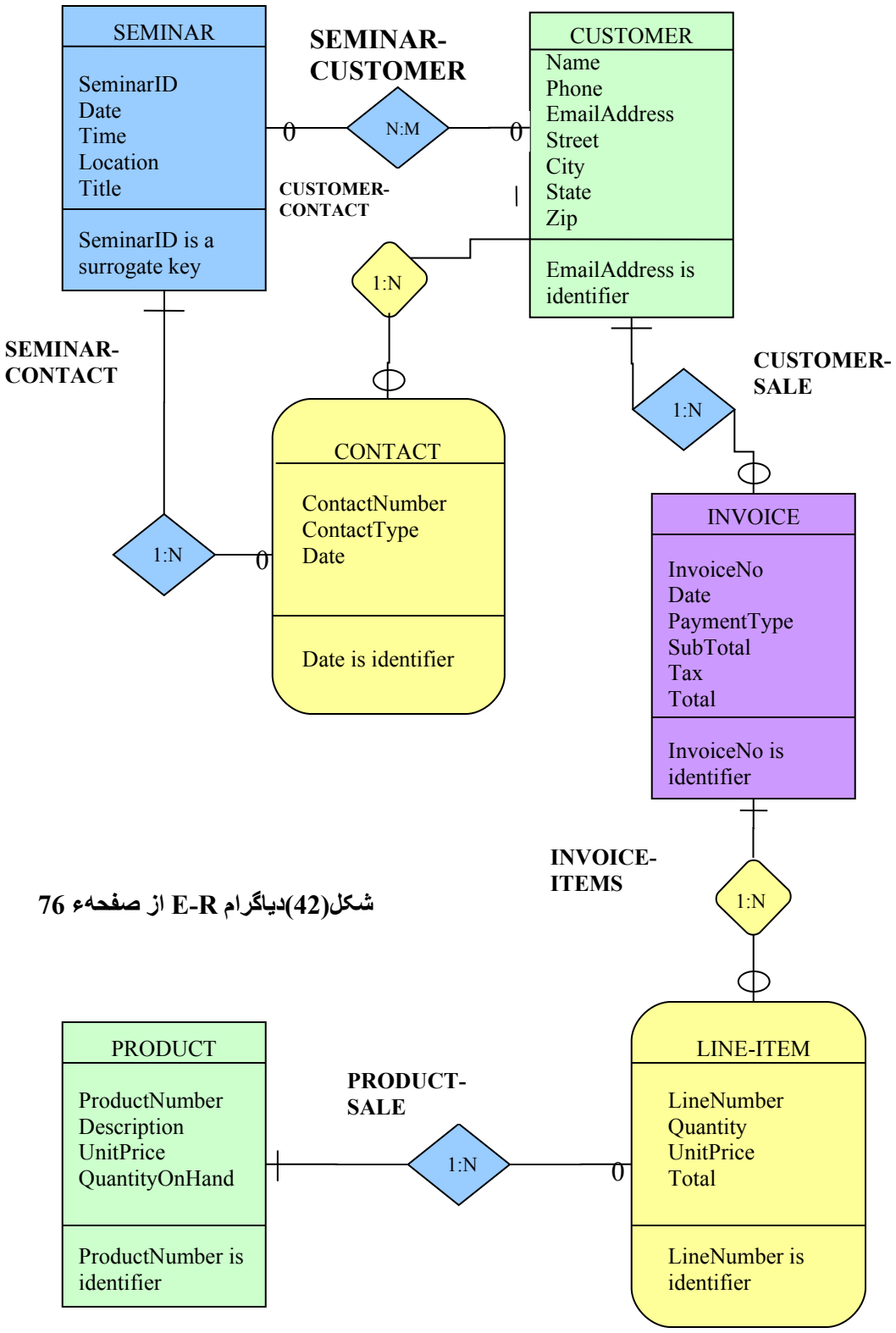
-عناصر ستون *SupplierID* در جدول SUPPLIES باید شامل عناصر ستون SupplierID در جدول SUPPLIER باشند.

- عناصر ستون *ProjectID* در جدول SUPPLIES باید شامل عناصر ستون ProjID در جدول PROJECT باشند.

-عناصر ستون *PartID* در جدول SUPPLIES باید شامل عناصر ستون PartID در جدول PART باشند.

دیزاین عملی دیتابیس (مثال صفحه 121)

در شکل(42) ، دیاگرام E-R، که برای یک کمپنی دیزاین شده، نشان داده شده است]4، ص. 124[.



شکل(42)دیاگرام E-R از صفحهء 76

در این قسمت دیاگرام بالا به شکل Relational-Model دیزاین میگردد.

برای اجراء این کار، اولا هر Entity-Class به یک Relation تبدیل شده نشان داده میشود.

SEMINAR (SeminarID, Date, Time, Location, Title)

CUSTOMER (Name, Phone, EmailAddress, Street, City, State, Zip)

PRODUCT (ProductNumber, Description, UnitPrice, QuantityOnHand)

CONTACT (ContactNumber, ContactType, Date)

INVOICE (InvoiceNo, Date, PaymentType, SubTotal, Tax, Total)

LINE-ITEM (LineNumber, Quantity, UnitPrice, Total)

در قدم بعدی مراحل نارمل سازي بالای جداول بالا تطبیق میشوند.

آیا کدام Functional-Dependency دومی در جدول ها وجود دارد؟ - بلی، یک Functional-Dependency وجود دارد.

Zip → (City, State)

بهر صورت، نظر به دلایلی که در قسمت Denormalization ارائیه شده اند، از این Functional-Dependency صرف نظر شده و در عین Table باقی میماند.

یک Functional-Dependency احتمالی دیگر در جدول SEMINAR در قسمت Location به شکل

Location → (Date, Time, Title)

میتواند وجود داشته باشد. چون این Functional-Dependency احتمالی است، پس دیزاینر باید با استفاده کنندگان دیتابیس موضوع را در میان گذاشته و تصمیم بگیرد.

فعلا از این قسمت نیز صرف نظر شده، فکر شود جداول بالا نارملایز اند و هیچ Functional-Dependency وجود ندارد.

Entityهای ضعیف

در قدم دوم Entity های ضعیف باید تفکیک شوند. در مثال ذکر شده، دو Entity ضعیف وجود دارند و هر دوی آنها ID-Dependent نیز میباشند. CONTACT یک Entity ضعیف است و Identifier آن مربوط به Identifier جدول CUSTOMER است. پس کاپی Primary-Key جدول CUSTOMER یعنی (EmailAddress)، باید در جدول CONTACT اضافه شده و بحیث قسمتی از Primary-Key در این جدول استفاده شود.

بهمین ترتیب LINE-ITEM یک Entity ضعیف است و Identifier آن مربوط یا تابع Identifier جدول INVOICE میباشد. بنا کاپی Primary-Key جدول INVOICE باید در جدول LINE-ITEM

اضافه شده و بحیث قسمتی از Primary-Key جدول یاد شده استفاده شود. یعنی

SEMINAR (SeminarID, Date, Time, Location, Title)

CUSTOMER (Name, Phone, EmailAddress, Street, City, State, Zip)

PRODUCT (ProductNumber, Description, UnitPrice, QuantityOnHand)

CONTACT (ContactNumber, ContactType, Date, *EmailAddress*)

INVOICE (InvoiceNo, Date, PaymentType, SubTotal, Tax, Total)

LINE-ITEM (LineNumber, Quantity, UnitPrice, Total, *InvoiceNo*)

نوت: فیلدها ی CONTACT.EmailAddress و LINE-ITEM.InvoiceNo هم Underline شده اند و هم بشکل *Italic* نوشته شده اند، بدین مفهوم که برعلاوه اي Primary-Key بودن، Freign-Key نیز اند.

Relationshipها

حال وقت آن رسیده است تا Relationshipها معین گردند. سه Relationship از نوع (N:1) موجود بوده و عبارت اند از:

SEMINAR & CONTACT

CUSTOMER & INVOICE

PRODUCT & LINE-ITEM

برای ایجاد هر کدام آنها، کاپی Primary-Key جداول Parent گرفته شده، در جداول Child بشکل Freign-Key اضافه میشوند. یعنی کلید د SEMINAR د ر CONTACT، کلید CUSTOMER د ر INVOICE و کلید د PRODUCT د ر LINE-ITEM گذاشته میشوند.

جداول فعلا به شکل ذیل اند.

SEMINAR (SeminarID, Date, Time, Location, Title)

CUSTOMER (Name, Phone, EmailAddress, Street, City, State, Zip)

PRODUCT (ProductNumber, Description, UnitPrice, QuantityOnHand)

CONTACT (ContactNumber, ContactType, Date, *EmailAddress*, *SeminarID*)

INVOICE (InvoiceNo, Date, PaymentType, SubTotal, Tax, Total, *EmailAddress*)

LINE-ITEM (LineNumber, Quantity, UnitPrice, Total, *InvoiceNo*, *ProductNumber*)

همچنان یک Relationship چندین به چندین (N:M) نیز در دیاگرام نشان داده شده است و آن در بین جداول SEMINAR و CUSTOMER وجود دارد. کاریکه در این قسمت ضرور است تا اجراء شود، اینست که Relationship متذکره به دو Relationship یک به چندین (N:1) تجزیه شود و یک Intersection-Table در بین آنها ایجاد شود. اگر نام جدول جدید SEMINAR-CUSTOMER گذاشته شود، فیلدهای آن متشکل از Primary-Keyهای جداول داخل Relationship خواهد بود. و همین ستونها در جدول جدید رول Primary-Key را بازی نموده و بصورت ترکیبی یا Composite استفاده میشود.

نتیجه نهایی دیزاین طور ذیل خواهد بود:

SEMINAR (SeminarID, Date, Time, Location, Title)

CUSTOMER (Name, Phone, EmailAddress, Street, City, State, Zip)

PRODUCT (ProductNumber, Description, UnitPrice, QuantityOnHand)

CONTACT (ContactNumber, ContactType, Date, *EmailAddress*, *SeminarID*)

INVOICE (InvoiceNo, Date, PaymentType, SubTotal, Tax, Total, *EmailAddress*)

LINE-ITEM (LineNumber, Quantity, UnitPrice, Total, *InvoiceNo*, *ProductNumber*)

SEMINAR-CUSTOMER (*SeminarID*, *EmailAddress*)

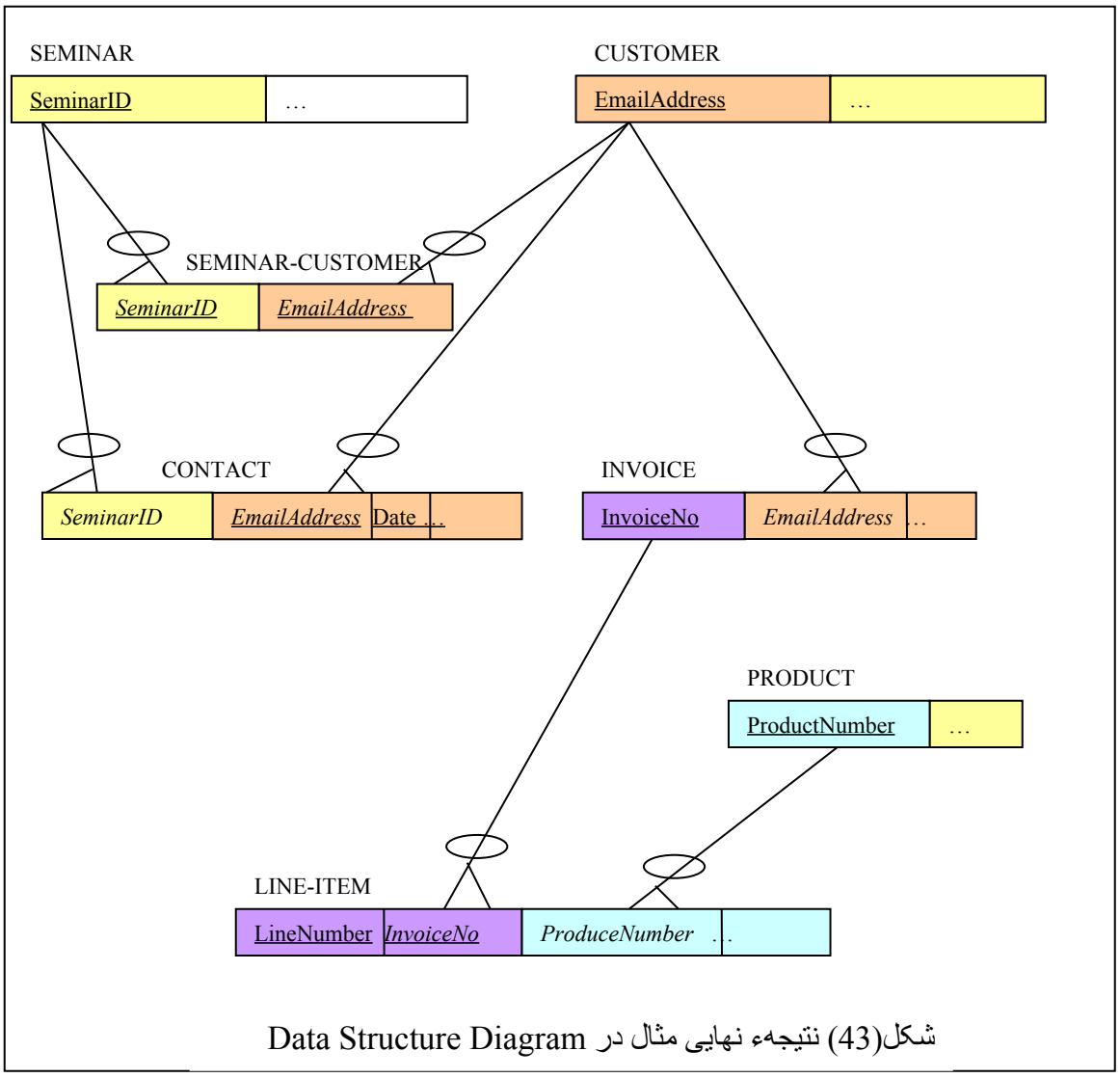
در این قسمت، بخاطر نشان دادن کاردینالتي اصغری، Childها به Parentها، ضرورت است تا تعین شود، آیا فیلدهای Freign-Key ضروری (Required)* هستند و یا خیر؟

در شکل(42) وقتی دقت شود، دیده میشود که یک INVOICE باید مربوط یک CUSTOMER باشد، بعین ترتیب یک LINE-ITEM باید مربوط یک PRODUCT باشد. پس فیلدهای INVOICE.EmailAddress و LINE-ITEM.ProductNumber باید فیلدهای 'Required' تنظیم شوند. برعکس فیلد CONTACT.SeminarID ضرورت ندارد تا 'Required' باشد، چرا که یک CONTACT ضرورت ندارد تا حتما به یک SEMINAR مربوط باشد.

* Required عبارت از صفت یک فیلد میباشد. اگر OK باشد، فیلد متذکره حالت Null را قبول نمی نماید و حتما باید دارای دیتا باشد.

نتیجه نهایی در دیاگرام Data-Structure شکل (43) نشان داده شده است.

ستون های Nonkey نشان داده نشده اند.



شکل(43) نتیجهء نهایی مثال در Data Structure Diagram

در اخیر قواعد Referential Integrity Constraint (RIC)، که در جدول شکل(44) نشان داده شده است، برای مثال قبلی قابل تطبیق اند.

Relationship	(Referential Integrity Constraint(RIC
SEMINAR به SEMINAR-CUSTOMER	عناصر ستون SeminarID در جدول SEMINAR-CUSTOMER باید شامل ستون SeminarID در جدول SEMINAR باشند.
CUSTOMER به SEMINAR-CUSTOMER	عناصر ستون EmailAddress در جدول SEMINAR- CUSTOMER باید شامل ستون EmailAddress در جدول CUSTOMER باشند.
SEMINAR به CONTACT	عناصر ستون SeminarID در جدول CONTACT باید شامل ستون SeminarID در جدول SEMINAR باشند.
CUSTOMER به CONTACT	عناصر ستون EmainAddress در جدول CONTACT باید شامل ستون EmainAddress در جدول CUSTOMER باشند.
CUSTOMER به INVOICE	عناصر ستون EmainAddress در جدول INVOICE باید شامل ستون EmainAddress در جدول CUSTOMER باشند.
INVOICE به LINE-ITEM	عناصر ستون IvoiceNo در جدول LINE-ITEM باید شامل ستون InvoiceNo در جدول INVOICE باشند.
LINE-ITEM به PRODUCT	عناصر ستون ProductNumber در جدول LINE-ITEM باید شامل ستون ProductNumber در

	جدول PRODUCT باشند.

شکل(44) جدول قواعد Referential Integrity Constraint(RIC) مثال قبلی

References

- Ramakrishnan, Ragu. 1997. Database Management Systems.
- Peter Rob, Carlos Coronel. 1997. DATABASE SYSTEMS: Design, Implementation, and Management.
- C. J. Date. 2004. An Introduction to Database Systems.
- D. M. Kroenke. 2005. DATABASE CONCEPTS.