Subject: اصول طراحی پایگاه داده ها
Year. 86 Month. 6 Date. 31 (1)

مهندس عبدالرضا پیرشایان

«دنیا آینده است، آن را جز با سیر عشق آشتی نگیرد»

جلسه اول 86.6.31 ← فصل (1) ←

DBMS : Data Base Management System

وظایف DBMS :

1. حجم عظیمی از اطلاعات را در فایل نگه داری می کند.
2. به وسیله یک سری از برنامه ها به این اطلاعات (داده های ذخیره شده) دسترسی پیدا می کند.

این دسترسی به دو صورت انجام می گیرد:

- دسترسی DBMS به اطلاعات ذخیره شده ← Convenient (راحت) / efficient (بهره وری)

معایب عدم وجود DBMS :

1. Data redundancy (افزونگی داده ها) ← حافظه اضافی اشغال می شود و باعث حجم بالایی شود
2. inconsistency (عدم همخوانی) ← نسخه های مختلف سیستم با هم هماهنگی ندارد
3. مشکل در دسترسی داده ها

E.F. Codd

Jim Gray قوانین جامعیت E.F.Codd را به 4 حالت زیر تبدیل کرد:

1) Atomicity (تجزیه ناپذیر) ؛ یک عمل یا کامل انجام می شود و یا اصلاً انجام نمی شود.
2) Consistency (هم خوانی) ؛ نسخه های مختلف با هم هماهنگ هستند.
3) Isolation (جداپذیری) ؛ همزمانی کاربرها تاثیری روی هم نداشته باشند و هر کاربر به تنهایی کارش را انجام دهد.
4) Durability (پایایی) ؛ برای داده های ذخیره شده حفاظت ایمنی و قابل قبول صورت گیرد تا داده ها از بین نروند. (داده ها باید پایدار باشند)

PAPCO

Subject:
Year. Month. Date. ( )

Rbstraction: نگرش برداشت کلی از یک واقعیت را گویند یعنی به جزئیات توجهی نداریم

Data Rbstraction: (سطوح داده ها (انتزاعی

1, Physical level ⇝ نحوه ذخیره شدن داده ها (داده چه جوری ذخیره می شود)
2, logical level ⇝ چه داده ای ذخیره می شود) نوع داده های ذخیره شونده
3, View level ⇝ براساس نیاز کاربر اطلاعات مورد نیاز را در اختیار می گذارد
↑ level بالاترین

Schema: ساختار data base بدون در نظر گرفتن داده ها را schema گویند
و کثیف ترین می آید که schema تغییر کند (یعنی رکوردها را حذف کرده باشیم)

Schema instance: نمونه های از schema در هر لحظه (یعنی داده ها در یک لحظه یک schema هستند)

Data Models:

Data Models مجموعه ای از ابزارهای مفهومی برای توصیف داده ها، ارتباط داده ها، مفاهیم و نقش داده ها (Semantic)، همخوانی و محدودیت (Constraint) داده ها برای حفظ یکپارچگی داده هاست؛

1, Relational Model
2, Entity - Relationship Model
3, Object - Based Model
4, Semi - Structured Model

P4PCO

Subject: اصول طراحی پایگاه داده ها
Year 86 Month 7 Date 7 (2)

جلسه دوم 86.7.7 ← فصل (6)

Entity Relationship Model: ← این مدل بهترین ابزار برای طراحی Data Base است

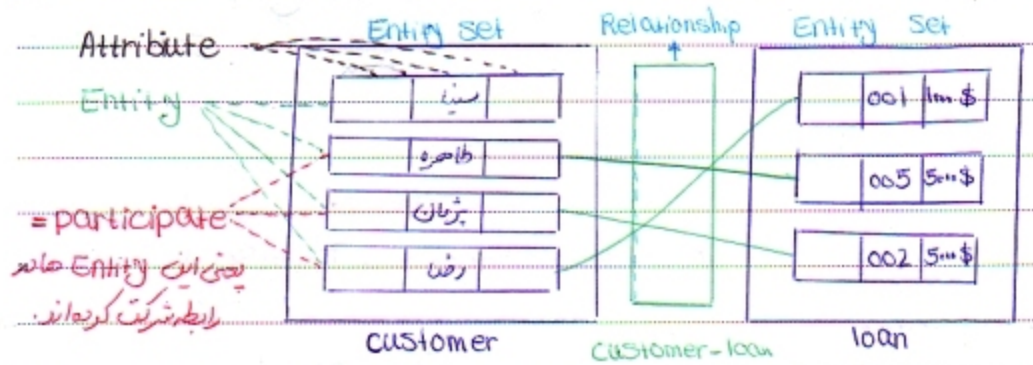
موجودیت ارتباط

- Entity: یک چیز، شی (object) است. مثلاً دانشجو، درس، استاد و... در سیستم دانشگاه Entity هستند.

- Set (مجموعه): گروهی از موجودیت ها (چیزها) که ویژگی مشترکی دارند را یک Set گوییم.

- Entity Sets: به مجموعه‌ای از Entity ها گفته می‌شود مثلاً مجموعه درس، مجموعه اساتید و مجموعهٔ دانشجوها در سیستم دانشگاه Entity sets هستند.

مثال: در زیر دو Entity Sets بنام‌های Customer (مشتری) و loan (وام) وجود دارد که هرکدام از آن‌ها به ترتیب شامل 4 و 3 Entity هستند که بین این دو Entity Sets روابط تعریف شده است این Relationship را Customer-loan می‌نامیم.



توجه: همانطور که می‌بینیم هر وام حتماً باید به کسی تعلق گیرد یعنی تمام وام‌ها participate هستند اما در مجموعهٔ مشتری‌ها می‌تواند شخصی باشد که وامی را دریافت نکرده (یعنی در رابطهٔ شرکت نکرده باشد) مثل سینا در این مثال.

P4PCO

Subject:
Year. Month. Date. ( )

Attributes: اطلاعاتی راجع به Entity به ما می‌دهد و به طور کلی به دسته‌های زیر تقسیم می‌شود:

- Simple (ساده): صفت ساده مثل اسم، شماره دانشجویی و...
- Compo Site (مرکب): صفتی که بتوان آن را به قسمت‌های کوچکتر تقسیم کرد مثل آدرس
- Single valued (تک مقداری): صفتی که هر Entity تنها یک مقدار را بتواند قبول کند مثلاً شماره دانشجویی برای هر دانشجو یک صفت تک مقداری است.
- multi valued (چندمقداری): صفتی که هر Entity بتواند چند مقدار آن را قبول کند مثل شماره تلفن که هر Entity می‌تواند بیش از یک شماره تلفن داشته باشد.
- Derived (مشتق شده): یعنی مقدار آن از At. دیگری گرفته شده باشد مثلاً At. سن که از روی At. تاریخ تولد محاسبه می‌شود (سن از تاریخ تولد مشتق شده) و یا معدل که از روی نمرات درس مشتق شده است.

Relationship Set: در زیر به تعریف ریاضی مجموعهٔ روابط می‌پردازیم، باید توجه داشته باشیم که Relationship set مجموعه‌ای از چندتایی‌هاست.

Relationship Set:

$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

(مجموعه‌ای — از چندتایی‌ها)

entity: ✗ / Entity Sets ✓

- دوتایی قبل دو Entity Sets داریم ← درجه 2 است.

Relationship Set درجه = تعداد Entity Set ⇐

(مجموعه 2تایی)

Customer_loan = {(طاهره, 005), (رضا, 001), (پرمان, 002)}

Customer = {پرمان، سینا، رضا، طاهره}

loan = {001, 002, 005}

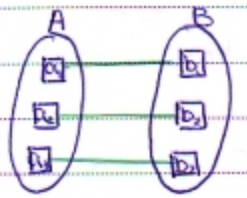
* محدودیت‌ها (Constraints) چه هستند:

1* mapping Cardinalities → صحبت روی تعداد map ها را گویند
نگاشت تعدادپذیری

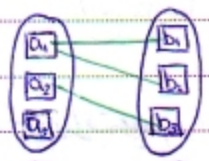
mapping: نگاشت بین دو چیز و نسبت دادن را mapping نامیم.

Cardinalites: یعنی تعدادپذیری به این معنی که اجازه داشته باشیم یک چیز را چندبار map کنیم. مثلاً در مثال قبل به رضا هم وام 001 و هم وام 002 تعلق گیرد.

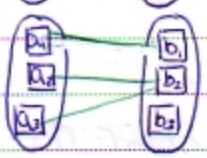
* در زیر به بررسی 4 mapping Cardinalities می پردازیم: (با توجه به مثال قبل)



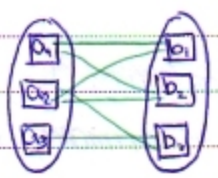
One_to_One ← هر مشتری تنها می تواند یک وام بگیرد و هر وام تنها می تواند به یک مشتری تعلق گیرد.



One_to_many ← هر مشتری می تواند صفر یا چند وام بگیرد ولی هر وام تنها باید یک مشتری تعلق گیرد.



Many_to_One ← هر مشتری می تواند حداکثر یک وام بگیرد و چند مشتری با هم می توانند یک وام بگیرند



Many_to_many ← هر مشتری می تواند به هر تعداد وام بگیرد و هر وام هم می تواند به هر تعداد مشتری تعلق گیرد (آزادترین حالت و بدون محدودیت)

2* participation Constraints: → (محدودیت های روابط)
شرکت کردن محدودیت

مثلاً تعیین حداقل و حداکثر شرکت کردن هر مشتری در وام گرفتن (بدون درنظرگرفتن mapping)

Subject:
Year. Month. Date. ( )

توجه: باید دقت کنیم که participation Constraints و mapping Cardinalities کاملاً متفاوت است و نباید با هم اشتباه گرفته شوند.

*3 keys (کلیدها): کلید یک Entity Sets می‌تواند یک یا چند Attribute باشد که آن Attribute باید منحصر به فرد باشد.

- Super keys (ابرکلید): به کلیدی گفته می‌شود که بتواند به کلیدهای زیرمجموعه خود داشته باشد به طوری که به کلیدی Super keys گفته می‌شود که در دل خود یک یا چند کلید داشته باشد. مثلاً اگر فرض کنیم نام و نام خانوادگی و شماره دانشجویی کلید باشد حتماً Super keys خواهد بود.

- Candidate keys: به کلیدی که قابلیت انتخاب شدن دارد را Candidate keys گوییم، درواقع Super keys که شکسته نشود را Candidate keys نامند. مثلاً کلید (اسم، تهر، شماره شناسنامه) یک Candidate keys است زیرا چنانچه این کلید را بشکنیم و هر قسمتش را برداریم دیگر کلیدی نخواهیم داشت.

توجه: بهتر است Attribute کمتری را به عنوان Candidate keys در نظر بگیریم، مثلاً بین (نام و نام خانوادگی و شماره شناسنامه) و (شماره دانشجویی) بهتر است شماره دانشجویی را انتخاب کنیم.

- primary key: انتخاب یک key از بین چند Candidate keys یک primary key است؛ در مثال بالا شماره دانشجویی یک primary key است.

جلسه سیزدهم 86.7.14

نکته: primary key ⊆ candidate key ⊆ super key

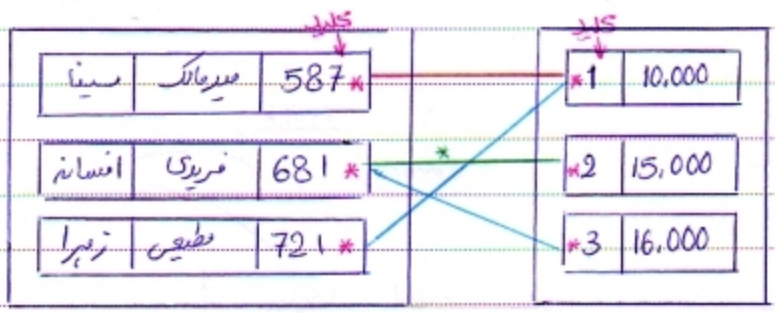
- هر super key ممکن است به کلیدهای دیگر تجزیه شود و باید توجه داشته باشیم که هر primary key یک super key هم هست.

P4PCO

Subject: اصول طراحی پایگاه داده
Year 86 Month 7 Date 14 4

نکته: در Relationship set برای ما primary key محسوب است

مثال: دو Entity set زیر را در نظر بگیرید



برای نوشتن مجموعه Relationship باید primary key را در نظر بگیریم.

Relationship set: {(681, 2), (721, 1)}

باتوجه به mapping cardinalits عبارت است نوع (681, □) تکرار شود.

one-to-one: هیچ نوعی تکرار نمی‌شود و کلید ارتباط می‌تواند کلید Customer یا loan باشد

* one-to-many: نوع تکراری خواهد داشت؛ مثل (681, 2) (681, 3)

* many-to-one: نوع تکراری خواهد داشت؛ مثل (587, 1) (721, 1)

- همانطور که می‌بینیم در سمتی که one است تکرار وجود دارد بنابراین برای حل این مشکل کلید سمت many را به عنوان کلید ارتباط در نظر می‌گیریم.

و در حالت many-to-many اجتماع کلید هر دو سمت کلید ارتباط را تشکیل می‌دهند.

Entity Relationship Diagrams:

- قراردادهای رسم Diagram به صورت زیر است:
- مستطیل: Entity set را نشان می‌دهد.
- لوزی: Relationship را نشان می‌دهد.
- بیضی: Attribute را نشان می‌دهد.

Subject:
Year. Month. Date. ( )

نکته: نسبت‌های مختلف را هم diagram وصل می‌کند.

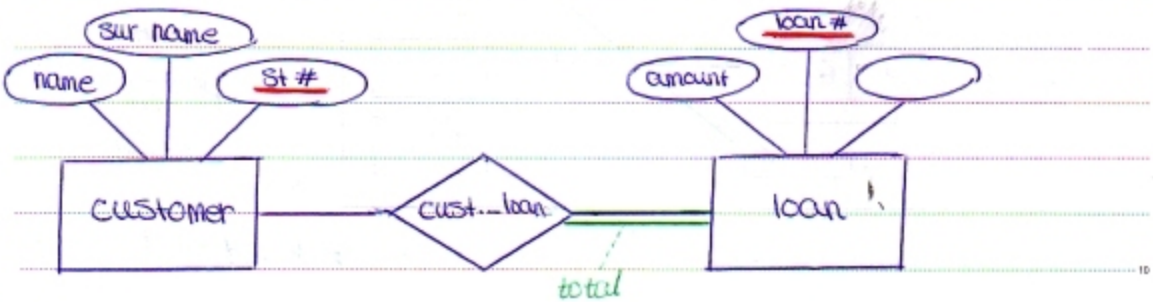
نشان دهندهٔ many است. : —

نشان دهندهٔ one است. : →

نوک فلش به سمت مستقل

one-to-many : loan به customer از

many-to-one : customer به loan از



total: در Entity set که همهٔ Entity ها باید (حداقل) حتماً یکبار در رابطه شرکت کنند آنگاه آن Entity set را total می‌نامند و خط دوبل نشان داده می‌شود.
مثلاً در اینجا هر وامی باید حداقل یکبار در ارتباط شرکت کند.

multivalued att.: Double ellipses را نشان می‌دهد.

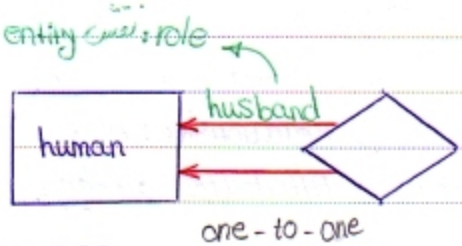
derived att.: Dashed ellipses را نشان می‌دهد.

L..h : تعیین حداقل و حداکثر شرکت کردن را نشان می‌دهد.

مثلاً 0..5 حداقل 0، حداکثر 5 بار می‌تواند شرکت کند.

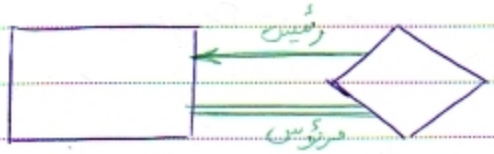
و 1..1 یعنی فقط و فقط یکبار می‌تواند در ارتباط شرکت کند.

نکته: در برخی موارد لازم است از روابط بازگشتی استفاده کنیم. مثلاً اگر مجموعهٔ انسان‌ها را در نظر بگیریم برای نمایش ارتباط زناشویی، ارتباط دوستی و... باید از خطوط بازگشتی مطابق شکل استفاده کنیم زیرا زن و شوهر هر دو در همان مجموعه انسان‌ها هستند



P4PCO

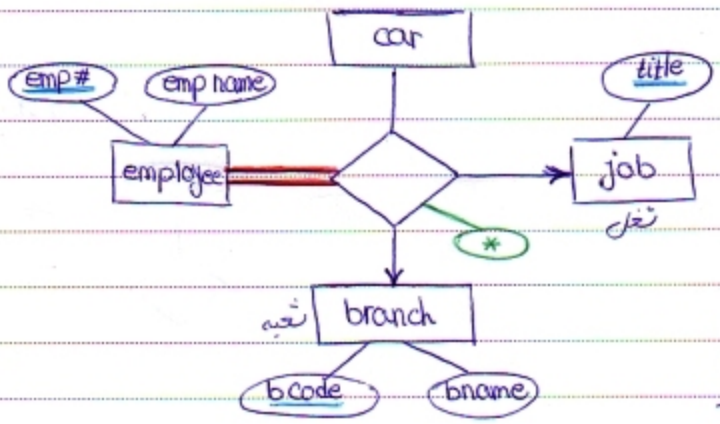
Subject: اصول طراحی پایگاه داده‌ها
Year. 86 Month. 7 Date. 14/5



مثال: رابطه رئیس و مرئوس بدون هیچ یک ارتباط بازگشتی است با توجه به اینکه کسی نمی‌تواند در ارتباط مرئوس بدون شرکت نکند بنابراین total است. و فرض می‌کنیم هر کس یک رئیس مستقیم داشته باشد.

many-to-one از مرئوس به رئیس
one-to-many از رئیس به مرئوس

مثال:
در اینجا یک Relationship چهارتایی داریم



* همانطوری که می‌بینیم می‌توانیم برای Relationship ها هم attribute در نظر بگیریم و این پدیده وقتی رخ می‌دهد که آن Relationship به هیچ entity set مربوط نباشد.

* total است زیرا هر کارمند باید job داشته باشد یعنی در همه ارتباطات باید شرکت کند زیرا در غیر این صورت دیگر شخص کارمند نیست.

توجه: این مثال دارای ابهام است زیرا دارای دو فلش است (one) مثلاً می‌توان ارتباط سه‌تایی ترکیب از employee و car ــــ با یک ترکیب از job و branch تعریف کرد. یا یک job و یک branch تعریف کرد.

برای رفع ابهام در ارتباطات بیش از دوتایی باید تنها یک ارتباط را با فلش تعریف کنیم. در غیر این صورت مبهم خواهد بود.

PAPCO

Subject :
Year : Month : Date : ( )

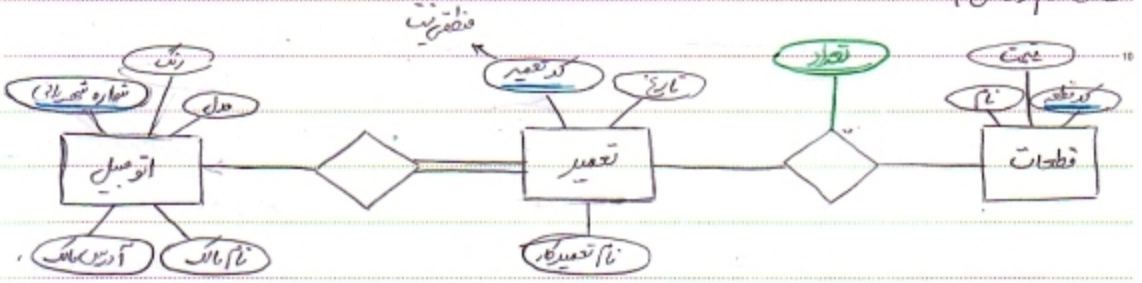
مثال : Diagram تعمیرگاهی را رسم کنید که اطلاعات مشتریها، اتومبیل ها و قطعات تعویضی در آن نشان داده شود.

- می توانیم مشتری را یک entity set درنظر بگیریم و یا مشتری را به عنوان یک entity set ، at. اتومبیل درنظر بگیریم که حالت دوم بهتر است.

حالت اول (نامناسب)

هر ماشین متعلق به یک نفر است ولی هر مشتری می تواند چند ماشین داشته باشد.

حالت دوم (کامل)



تعلق قطعات هم به تعمیر و هم به قطعات بستگی دارد بنابراین روی Relationship قرار می گیرد.

- توجه : اگر ارتباط اولی را به این صورت درنظر می گرفتیم و قطعات را هم به آن متصل می کردیم Diagram اشتباه بود زیرا در آن صورت هر ماشینی که به تعمیرگاه می رفت باید قطعه ای را هم تعویض می کرد در صورتی که امکان دارد اتومبیل بدون تعویض قطعه تعمیر شود.
- تعلق را نمی توانیم روی قطعات تعریف کنیم زیرا در آن صورت مشخص نمی شود که کدام قطعه مربوط به کدام تعمیر (ماشین) است.

نکته : می توان Entity Relationship Diagram را به table متعلق کرد (به راحتی) به همین دلیل ER بسیار مورد توجه و استفاده قرار گرفته است.

P4PCO

مثال: می خواهیم مثال قبل را به صورت جدول تبدیل کنیم.

در تبدیل Diagram به table:

هر Entity set به یک جدول و هر Relationship set نیز یک جدول انتقال می یابد. ستونهای جدول Entity set ها attribute ها را تشکیل می دهند و ستونهای جدول Relationship set کلید ارتباط و همچنین at. های روی ارتباط تشکیل می دهند و سطرهای این جداول شرکت کننده ها هستند.

جدول تعمیر

نام تعمیرکار	تاریخ	کد تعمیر
رضا	۸۹/۷/۱۴	1
بهمن	۸۹/۹/۹	2
حسن	۸۹/۳/۵	3
محسن	۸۹/۷/۹	4

جدول اتومبیل

آدرس مالک	نام مالک	مدل	رنگ	شماره شهربانی
تهران	علی	۱۳۷۹	سفید	۲۵ م ۹۴۵
تهران	سارا	۱۳۸۹	نقره‌ای	۱۵ ق ۱۴۱
تهران	مینا	۱۳۸۳	سبز	۴۵ م ۸۹۹

جدول قطعات

کد قطعه	نام	قیمت
1	سگ‌صبور	
2	شمع	
3		

- چون رابطه اول many-to-many کلید هردو Entity set را باید در جدول بنویسیم + (دو ستون) دار ارتباط دوم (سه ستون) داریم زیرا علاوه بر دو کلید یک ستون هم به at. تعلق می گیرد.

جدول ارتباط (R) اول

کد اتومبیل ←	کد تعمیر ←
شماره شهربانی	کد تعمیر
۲۵م ۶۴۵	2
۲۵ م ۶۴۵	3
۴۵ م ۸۹۹	1
۱۵ ق ۱۴۱	4

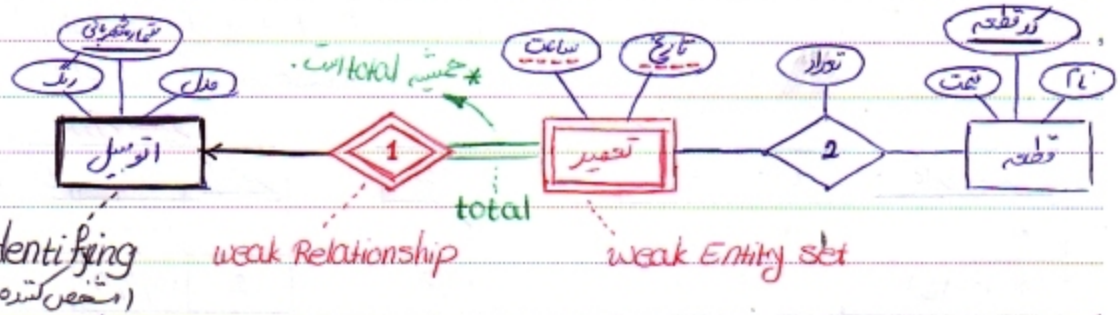
جدول ارتباط (R) دوم

کد قطعه ←	کد تعمیر ←	at ←
کد قطعه	کد تعمیر	تعداد
1	2	1
2	2	4

Subject:
Year. Month. Date. ( )

جلسه چهارم 86.7.28

Weak Entity Sets : entity set که استقلال ندارد و به entity set های دیگر وابسته است
(primary key ندارد) . انتیتی ضعیف گفته می‌شود چرا که هویت وابسته دارد

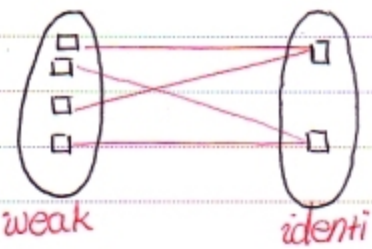


- طبق weak entity Set را partial key می‌نامند چون واقعاً کلید نیست اما در تشکیل کلید خاصیت دارد مثلاً در اینجا (ساعت، تاریخ) partial key هستند.

توجه: برای تعیین کلید weak entity Set کافی است attribute کلید + partial key وابسته را با هم در نظر بگیریم؛ یعنی در اینجا کلید انتیتی ست تعمیر (ساعت - تاریخ - شماره شناسایی) است.

نکته: اگر چند Entity Set را در یک primary key با هم یک attribute تشکیل دهند آن Entity Set نمی‌تواند weak باشد مثلاً چنانچه در مثال قبل attribute دیگری به عنوان نام تعمیرکار هم بود دیگر تعمیر یک weak ent. نبود.

* هر Entity در Entity Set تعمیر فقط و فقط به یک اتومبیل وابسته است بنابراین ارتباط از تعمیر به اتومبیل many-to-one است.



نتیجه
weak ~~many-to-one~~ identifying

P4PCO

Strong Entity Set : هر انتیتی ستی که weak نباشد Strong است یعنی انتیتی ستی که primary key دارد و مستقل است Strong گفته می‌شود.

- مثال قبل را در نظر بگیرید:

جدول تعمید | جدول ارتباط (1) | جدول ارتباط (2)

شماره شهربانی	تاریخ	ساعت

شماره شهربانی	تاریخ	ساعت

شماره شهربانی	تاریخ	ساعت	کد قطعه	تعداد

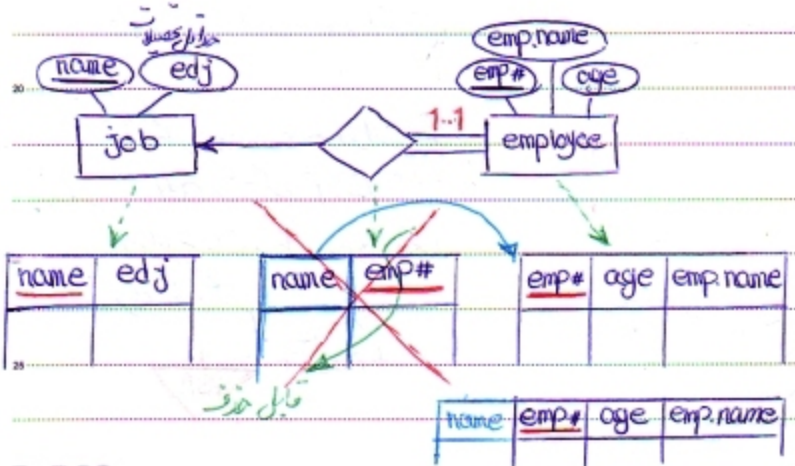
قابل حذف

- وقتی می‌توانیم چیزی را حذف کنیم که اضافی باشد پس از حذف اطلاعاتی از سیستم پاک نشده، خدشه‌ای به سیستم وارد نکند در اینجا هم همانطور که می‌بینیم Table ارتباط (1) و تعمید کاملاً مثل هم بوده یکی از آنها قابل حذف است؛

دلیل شباهت دو جدول: ۱- ارتباط مربوط به Entity Set تعمید total است و ۲- ارتباط many-to-on از تعمید به اتومبیل؛ پس شده این دو جدول کاملاً شبیه باشند.

← هرجا ارتباط many-to-one داشته باشیم که many آن total باشد می‌توانیم جدول Relationship را حذف کنیم.

به مثال زیر توجه کنید:

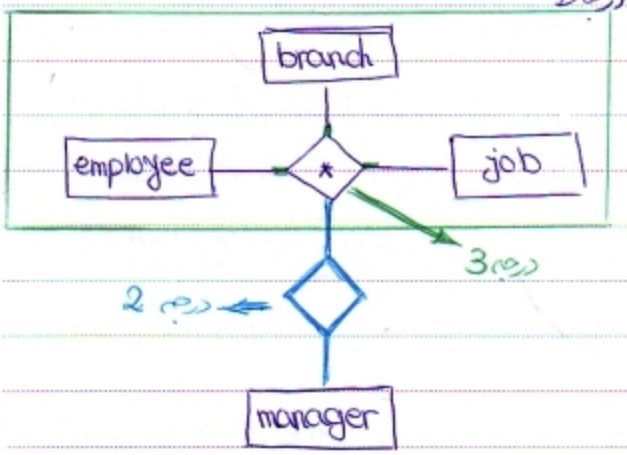


طبق نکته ذکر شده در بالا جدول ارتباط قابل حذف است اما باید ستون name را به جدول employee اضافه کنیم.

Subject :
Year . Month . Date . ( )

: (Extended ER) EER

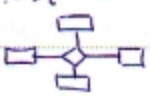
aggregation : (تجمع) یعنی یک نسبت ارتباط



* این ارتباط مشخص می کند که چه کسی ، چه شغلی دارد و در کدام شعبه مشغول به کار است ؛ ممکن است یک شخص در دو شعبه در دو شغل متفاوت داشته باشد اما در هر شعبه یک رئیس جدا گانه دارد بنابراین نمی توانیم بین کارمند و رئیس relationship بگذاریم لذا مجبوریم از طریق تعریف یک Relationship دیگر Entity رئوسا را به این Diagram اضافه کنیم .

((شعبه ، شغل و کارمند) و رئیس)

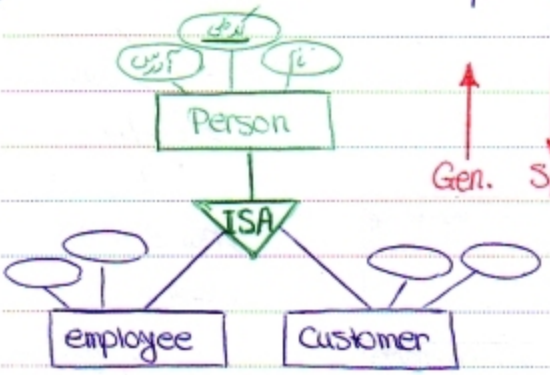
و در این ارتباط تعریف نکنیم ارتباط * در 4 خواهد بود و در آن صورت هر ارتباط با 4 icon مشخص می شود (شعبه ، شغل ، کارمند ، رئیس) که اشتباه است .



: Specialization & Generalization

جزئی ، کلی

گاهی اتفاق می افتد که دو Entity Set دارای attribute های مشترک و کلیدی باشند در این صورت بهتر است یک انتیتی ست جدا تعریف کنیم و att. های مشترک را به آن اختصاص دهیم .



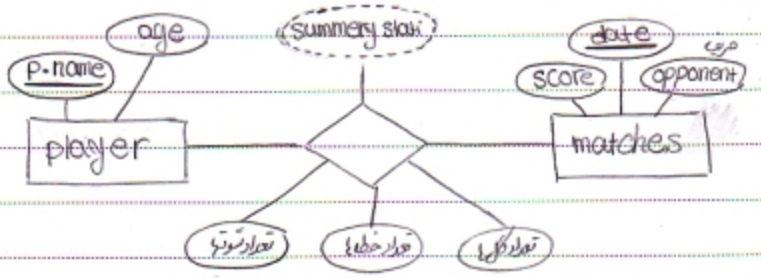
مثال : مثلاً کارمند و مشتری هر دو انسانند و دارای att. مشترکی مثل نام ، آدرس ، ... هستند بنابراین مانند شکل رو برد آن را پیاده سازی می کنیم :

یعنی از  استفاده می کنیم .

Subject: اصول طراحی پایگاه داده ها
Year. 86 Month. 7 Date. 28 8

تمرین P. 256 / 6.4

دیاگرامی رسم کنید که تیم مورد علاقه تان، بازیکنان، امتیازی که در هر بازی کسب می کند، بازیکنان تیم در هر بازی و آمار بازی ذخیره شود.



- داریم دنبال یک تیم (تیم مورد علاقه مان) هستیم بنابراین لازم نیست مثلاً تیم را ذخیره کنیم فقط کافی است نام حریف را ذخیره کنیم و در این صورت date می تواند کلید باشد چون یک تیم در یک تاریخ فقط می تواند دیگ مسابقه بگذارد بنابراین date برای matches یک primary key است.

و آن بازی که att. هایی مثل تعداد خطاها، تعداد توپ ها ... تیم ما، بازیکن و تیم حریف بستگی دارد باید آنها را در Relationship تعریف کرد.

جلسه پنجم 86.8.5 ← فصل (2)

Relational Model: این مدل، یک مدل موفق است (اولین مدل برای درخواستهای تجاری) و مهم ترین عامل در این مدل ارتباطات است.

Domain: دامنهٔ هر attribute مقادیر مجازی است که هر att. می تواند داشته باشد (یعنی مقادیر مجازی که در هر ستون می تواند قرار گیرد)

(header هر ستون نام attribute است)

توجه: هر relational DB مجموعه ای از جداول است که هر جدول یک نام مشخص دارد و هر ردیف (row) این جدول یک relationship را نشان می دهد و به طور غیر مستقیم هر جدول یک Entity set و هر ستون یک attribut است.

Subject:
Year. Month. Date. ( )

Basic Structure: در این جدول account relation را می‌بینیم که سه ستون دارد که
account # : مقادیر مجاز D_1 ، branch-name : مقادیر مجاز D_2 و
balance : مقادیر مجاز D_3 را شامل می‌شود.
هر سطر جدول شامل سه tuple است. (v_1, v_2, v_3) tuple = ردیف‌های جدول

← بنابراین account زیرمجموعه‌ای از $D_1 \times D_2 \times D_3$ است.

نتیجه: در حالت کلی، یک table شامل n attribute زیرمجموعه‌ای از $D_1 \times D_2 \times \dots \times D_n$ می‌باشد.

account

account #	branc-name	balance
A_1	میرداماد	۱۰۰۰۰۰/-
A_2	ونک	∅
A_3	ونک	۱۰۰۰/-

رابطه $= D_1 \times D_2 \times D_3$

رابطه $= \{A_1, A_2, A_3\} \times \{$ونک، میرداماد$\}$
$\times \{1000000/-, \emptyset, 1000/-\}$

$= \{(A_1$، میرداماد، $1000000/-)$ $(A_2$، ونک، $\emptyset)$ $(A_3$، ونک، $1000/-)\}$

tuple tuple

نکته: برای نمایش tuple variable از نماد t[ ] استفاده می‌شود.

مثال:
t [account #] = "A_1"
t [branch-name] = "میرداماد"

tuple variable relation
$t \in r$

نشان می‌دهد رابطه چیست: Schema
att. هایی دارد و با حرف بزرگ آغاز می‌شود، مثال:

Account_schema = (account #, branch-name, balance)

مثل int
a مثل
account (Account_schema)

account (account #, branch-name, balance) OR
نام جدول — کلید

در اینجا یک رابطه از نوعی که خودمان تعریف کردیم را ایجاد کردیم مثل int a,b
relation

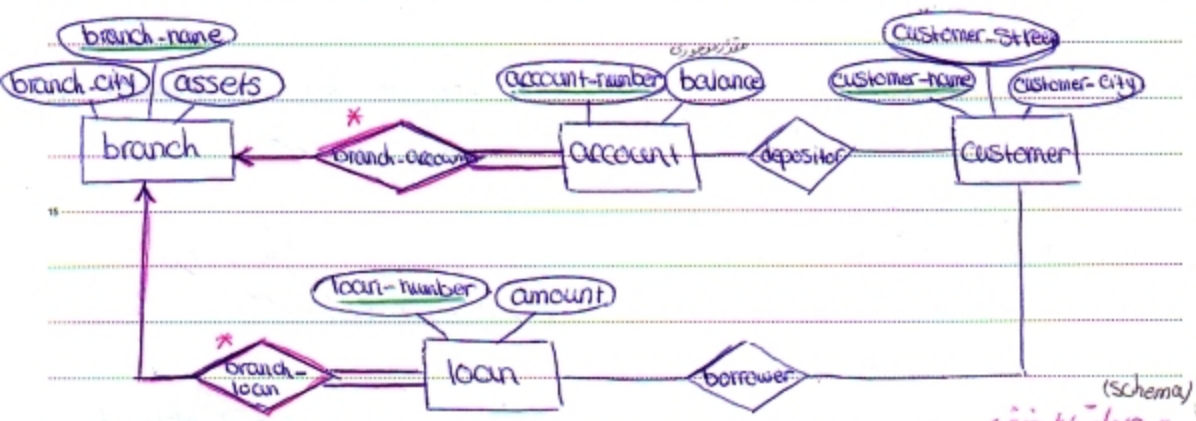
توضیحا: همانطوریکه قبلاً اشاره کردیم هر tuple variable با نماد t[ ] نشان داده می‌شود (یک سطر از یک جدول که ممکن است در هر لحظه یک مقدار داشته باشد)

R_Stance : مقداری که در ارتباط مورد نظرمان در هر لحظه می‌گیرد.

اگر R یک Relational schema و k یک super key باشد داریم:

$t_1 \neq t_2 \Rightarrow t_1[k] \neq t_2[k]$ $\quad K \subseteq R$

مثال:



(schema)

* جدول با فرض

branch (branch-name, branch-city, assets)

account (account-number, balance, branch-name)

customer (customer-name, customer-street, customer-city)

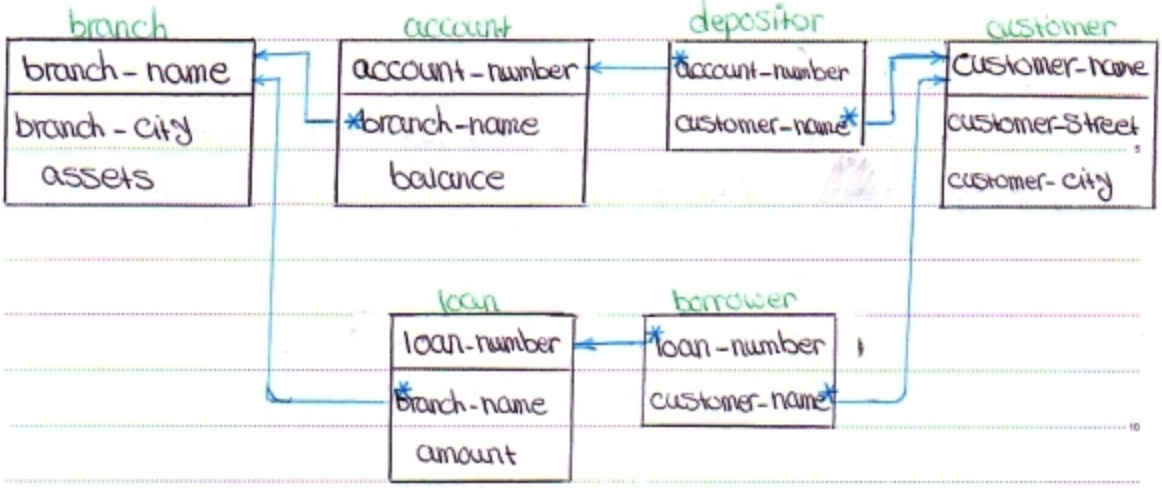
loan (loan-number, amount, branch-name)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Subject:
Year. Month. Date. ( )

حال به Schema Diagram مثال قبل توجه کنید:



* این مقادیر foreign key هستند یعنی مقادیرشان وابسته بوده و از جداول دیگر تامین می‌گردد. یعنی در یک schema دیگر کلید هستند.

توجه: در این دیاگرام زیر کلیدها خط می‌کشیم و از آنجایی که depositor, borrower دو کلید دارند هر دو آنها را به عنوان کلید زیرشان خط می‌کشیم.

بازیابی اطلاعات:

به طور کلی سه نوع پرس‌وجو در مدل رابطه‌ای وجود دارد:

1. Relational Algebra (جبر رابطه‌ای)
2. Tuple Relational calculats (حساب رابطه‌ای رکوردی (تاپلی)) TRC
3. Domain Relational calculats (حساب رابطه‌ای دامنه‌ای) DRC

1. Relational Algebra: جبر رابطه‌ای دارای 6 عملگر اصلی (operations) می‌باشد که سه‌تای آنها unary و سه‌تای دیگر binary هستند عملگرهای

P4PCO

Subject: اصول طراحی پایگاه داده‌ها
Year. 86 Month. 8 Date. 5 (10)

عملوند = (رابطه)
روی یک relation اجرا می‌شوند و عمگرهای binary روی دو عمگر اجرا می‌گیرند؛

و این شش عملگر عبارتند از: (شرط)

1. Selection (unary) → $\sigma_p(r)$ (روی Relation)
2. projection (unary) → $\Pi_A(r)$ $A \subseteq R$
3. Union (binary) → $\cup$
4. Set difference (binary) → $-$
5. Cartesiomproduct (binary) → $\times$
6. rename (unary) → $\rho_x(E)$ exp (نام دلخواه)

1، Selection: این عمگر unary به ما این امکان را می‌دهد که tuple هایی با شرایط خاص (مدنظرمان) را انتخاب کرده و آنها را از بقیه جدا کنیم؛ در قسمت شرط σ می‌توانیم از علائم $=, \neq, <, >, \leq, \geq$ و (and) $\wedge$، (or) $\vee$، (not) $\neg$ استفاده کنیم.

مثلاً می‌خواهیم کسانی که در شعبهٔ vanak بوده و مقدار وامشان از 1000 بیشتر است را انتخاب کنیم

* $\sigma_{amount > 1000 \wedge branch\text{-}name = "vanak"}(loan)$ *

که جدول به صورت زیر خواهد شد:

loan-number	branch-name	amount
L-15	vanak	1500
L-16	vanak	1300

2، projection: این عمگر unary این امکان را به ما می‌دهد که ستونی که به آن احتیاج نداریم را از جدول حذف کنیم مثلاً فرض کنید می‌خواهیم لیست همهٔ شماره وام‌ها را تهیه کنیم و شعبه برای ما اهمیت ندارد در آن صورت داریم:

* schema تعیین می‌کند: $\Pi_{loan\text{-}number, amount}(loan)$ ⇒

و با اعمال این دستور در واقع ستون مربوط به branch-name حذف شده و جدول موردنظر ما با 2 ستون ایجاد خواهد شد.

PqPCO

Subject:
Year. Month. Date. ( )

مثال: می خواهیم نام مشتریهایی که در "Harrison" زندگی می کنند را بیابیم.
برای این منظور باید از ترکیب دو عملگر به صورت زیر استفاده کنیم:

$$\Pi_{customer\text{-}name}\left(\sigma_{customer\text{-}city = \text{"Harrison"}}(customer)\right)$$

انتخاب مشتریهایی که در Harrison زندگی می کنند

جداکردن ستون نام مشتری ها و نمایش آن

همانطور که می بینیم باید ستون نام مشتریها را توسط Π از بقیه جدا کرده و با σ تمهیدات نظر را انتخاب کنیم تا در پایان یک جدول تک ستونی (نام مشتری) داشته باشیم.

3. union: این عملگر (اجتماع) به ما این امکان را می دهد که چند گروه مختلف (با شرایط گوناگون) را به طور همزمان در نظر بگیریم. مثلاً می خواهیم نام مشتریهایی که وام گرفته اند و یا نام مشتریهایی که حساب دارند را تعیین کنیم:

$$\Pi_{customer\text{-}name}(borrower) \cup \Pi_{customer\text{-}name}(depositor)$$

* که وام گرفته اند — * که حساب دارند

* نکته: union بین مجموعه هایی استفاده می شود که schema های آنها از نظر تعداد و دامنه یکسان باشند

4. set difference: این عملگر (تفریق) به ما اجازه می دهد tuple هایی را پیدا کنیم که در یک relation هستند اما در دیگری وجود ندارند. مثلاً می خواهیم نام مشتریهایی که حساب دارند ولی وام نگرفته اند را پیدا کنیم.

$$\Pi_{customer\text{-}name}(depositor) - \Pi_{customer\text{-}name}(borrower)$$

* که حساب دارند — * که وام گرفته اند

P4PCO

جلسه ششم 86.8.12

5. Cartesian-product : این عملگر به ما این امکان را می‌دهد که دو مجموعه رابطه را در هم ضرب (دکارتی) کنیم (۲ یا بیشتر) که نتیجه این ضرب دکارتی باید یک **رابطه** باشد.

مثال: فرض کنید می‌خواهیم دو رابطه loan و borrower را در هم ضرب کنیم:

borrower (loan-number, customer-name)

loan (loan-number, branch-nam, amount)

از آنجایی که loan-number در هر دو رابطه تکرار شده است باید در ضرب رفع ابهام کنیم:

borrower × loan =

(borrower.loan-number , borrower. customer-name ,
loan. loan-number , loan. branch-name, loan. amount)

چون بقیه موارد att. تکراری وجود ندارد بنابراین ابهامی نخواهیم داشت لذا دیگر احتیاجی نیست نام رابطه را در کنار att.ها بیاوریم.

نکته: در واقع ضرب دکارتی به ما این امکان را می‌دهد که ستونهای روابط مختلف را در کنار هم ایجاد کرده و در حالت‌های تقارت آنها را با هم مقایسه کنیم. هر چند ضرب یک عمل بسیار زمانبر و پرهزینه است.

borrower

loan-#	customer-name
L_1	آرین
L_2	علی
L_3	افسانه

loan

loan-#	branch-name	amount
L_1	میرداماد	10
L_2	ونک	20
L_3	میرداماد	30

borrower × loan

borrower. loan #	customer -name	loan. loan#	branch- name	amount
L_1	آرین	L_1	میرداماد	10
L_1	آرین	L_2	ونک	20
L_1	آرین	L_3	میرداماد	30
L_2	علی	L_1	میرداماد	10
L_2	علی	L_2	ونک	20
L_2	علی	L_3	میرداماد	30
L_3	افسانه	L_1	میرداماد	10
L_3	افسانه	L_2	ونک	20
L_3	افسانه	L_3	میرداماد	30

Subject:
Year. Month. Date. ( )

مثال: نام افرادی که از شعبه «برادران» وام گرفته اند را بیابید.

نام افرادی که وام گرفته اند در رابطه borrower و شعبه ای که وام گرفته اند در loan نگهداری می شود
بنابراین باید از ضرب دکارتی این دو مجموعه استفاده کنیم

در borrower x loan در جدول باید سطرهایی را انتخاب کنیم که loan# یکسان دارند
زیرا اگر در غیر اینصورت loan.loan# و borrower.loan# با هم یکی نباشند آن سطر بی معنا خواهد بود بنابراین داریم:

$$\Pi_{\text{customer-name}}\left(\sigma_{\text{branch-name}=\text{"برادران"}}\left(\sigma_{\text{borrower.loan\#}=\text{loan.loan\#}}(\text{borrower} \times \text{loan})\right)\right)$$

customer-name
آرین
امانه

6. Rename: این عملگر اصلی به ما این امکان را می دهد که نام هر exp مورد نظر و حتی att های آن را در صورت نیاز تغییر نام دهیم. مثلاً فرض کنید می خواهیم borrower را در خودش ضرب کنیم در این صورت باید حتماً از عملگر rename استفاده کنیم زیرا در غیر این صورت اسامی تکراری خواهیم داشت.

$$\text{borrower} \times \text{borrower} = \text{borrower} \times \rho_{x}(\text{borrower})$$
$$= (\text{borrower.loan\#}, \text{borrower.customer-name}, x.\text{loan\#}, x.\text{cus}_{\text{name}})$$

توجه: می توانیم نام یک یک اجزای هر schema را نیز تغییر دهیم: مثال

$$\rho_{x\,A_1,A_2}(\text{borrower}) \Rightarrow x = \text{borrower};\quad A_1 = \text{loan-\#};\quad A_2 = \text{customer-name}$$

و چنانچه بخواهیم فقط نام یکی از att ها را تغییر دهیم باید نام بقیه را عیناً نوشته و نام مورد نظر را برای آن یک att بنویسیم. (مثلاً به جای A_1 همان loan# را بگذاریم)

تعریف رسمی از جبر رابطه ای: یک exp در جبر رابطه ای شامل ۱- رابطه در DB و ۲- یک رابطه ثابت (Constant) می باشد.

۱- اگر E_1 و E_2 در جبر رابطه ای expression باشند آنگاه تمامی روابط زیر نیز exp های جبر رابطه ای خواهند بود:

P4PCO

Subject: اصول طراحی پایگاه داده
Year. 86 Month. 8 Date. 12 12

→ $E_1 \cup E_2$ → $E_1 - E_2$ → $E_1 \times E_2$

→ $\sigma_p(E_1)$ → $\Pi_A(E_1)$ → $\rho_x(E_1)$

عملگرهای اضافی در جبر رابطه‌ای:

علاوه بر 6 عملگر اصلی در جبر رابطه‌ای، 4 عملگر اضافی نیز در جبر رابطه‌ای وجود دارد که عبارتند از:

1. The set-Intersection Opration → "∩"
2. The Natural-join Opration → "⋈"
3. The Division Opration → "÷"
4. The Assignment Opration → "←"

1. Set_Intersection: اولین عملگر اضافی ∩ (اشتراک) است. فرض کنید می‌خواهیم مشتریهایی را پیدا کنیم که هم حساب دارند و هم وام گرفته‌اند، در این صورت داریم:

$\Pi_{\text{customer-name}}(\text{borrower}) \cap \Pi_{\text{customer\_name}}(\text{depositor})$

نکته: عملگر فرعی ∩ را می‌توان به کمک چند عملگر اصلی پیاده‌سازی کرد:

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

2. Natural join: در خروجی، att. های غیرمشترک و att. های مشترک (بدون تکرار) ظاهر خواهند شد؛ یعنی از att. های مشترک فقط یک نمونه نگه داری می‌شود.

این عملگر فرعی نیز ترکیبی از σ و × است:

$r \bowtie S = \sigma_{r.A=S.A \wedge r.B=S.B \wedge \dots}(r \times S)$

شرط: att. های هم‌نام مقایسه‌شان سازگار باشد

$R \cap S = \{A, B, C, \dots\}$ $r(R)$; $s(S)$;

مثال: نام مشتریهایی که از میرداماد وام گرفته‌اند؟

نقطه مشترک است: loan# ← این دو رابطه را انتخاب می‌کنند

$\Pi_{\text{customer-name}}\left(\sigma_{\text{branch-name} = \text{"میرداماد"}}(\text{borrower} \bowtie \text{loan})\right)$

از این رابطه یک رابطه خروجی می‌گیرد

PAPCO

Subject :
Year . Month . Date . ( )

نکته: علاوه بر natural join یک عملگر دیگر به نام θ join نیز وجود دارد که تنها فرق این دو join در شرطشان است در θ join هر شرط دلخواهی می‌توانیم استفاده کنیم اما در natural join شرط این است که attr های همنامی که مقادیرشان سازی است مدنظر قرار گیرند

$$\sigma_{\theta}(r \times s) = r \bowtie_{\theta} s$$
(شرط دلخواه ← θ)

مثال: شماره حساب مشتری که Max مقدار موجودی را دارد پیدا کنید.

برای پیدا کردن این مشتری باید balance مشتریهای مختلف را با هم مقایسه کنیم بنابراین ضرب دکارتی به یک نیاز داریم که attr هایی که می‌تواند در هم ضرب شوند پس باید account را در خودش ضرب کنیم و چون نمی‌توانیم مستقیماً Max را با هم بگیریم کوچکترها به دست آورده و از کل کم کنیم (max هیچ گاه کوچکتر نمی‌شود)

(شماره حساب مشتری با Max موجودی)
$$r_1 = \prod_{account\text{-}\#}(account) - \prod_{account.account\text{-}number}\left(\sigma_{account.balance < x.balance}(account \times \rho_{x}(account))\right)$$
(6 ستون دارد)

$$\Rightarrow \text{نام مشتری با Max موجودی} = \prod_{customer\text{-}name}(r_1 \bowtie depositor)$$

مثال: علاوه بر Division، فرض کنید می‌خواهیم نام مشتریهایی که در تمام شعب تهران حساب دارند را پیدا کنیم.

(فرض: تهران سه شعبه دارد)

$$\prod_{customer\text{-}name,\ branch\text{-}name}(account \bowtie depositor) \div \prod_{branch\text{-}name}\left(\sigma_{branch\text{-}city = \text{"Tehran"}}(branch)\right)$$
(4 ستون دارد — دو ستونش را حذف می‌کند)

← در تقسیم ستونهای مشترک اعمال می‌شود.

r_1

customer-name	branch-name
علی	ونک
علی	میرداماد
علی	نازی‌آباد
رضا	ونک
پرهان	

÷

r_2

branch-name
ونک
میرداماد
نازی‌آباد

→

customer-name
علی

Subject: اصول طراحی پایگاه داده‌ها
Year. 86 Month. 8 Date. 19 (13)

روش دوم جلسه قبل: یک مرجع مصنوعی ایجاد می‌کنیم یعنی فرض می‌کنیم که تمام مشتریها در تمام شعب تهران حساب دارند و آنهایی که جواب نیستند از این مجموعه مرجع کم می‌کنیم.

r_1 ← نام مشتریها با نام حسابشان
r_2 ← نام شعب تهران

فرض: تمام افراد در تمام شعب تهران حساب دارند

$$\prod_{\text{customer-name}}(r_1) - \left(\left(\prod_{\text{customer-name}}(r_1) \times r_2\right) - r_1\right)$$

اسم مشتریهایی که جواب نیستند

Assignment, 4: این عملگر درست ماند " ← " در زبان های برنامه نویسی عمل می‌کند
مثلاً
$temp1 \leftarrow \Pi_{R-S}(r_1)$

جلسه هفتم 86.8.19

مثال (P. 57) نام تمام شعب با مشتریهایی را که در Hamison زندگی می‌کنند و در بانک حساب دارند پیدا کنید.

از صورت مسئله می‌فهمیم که به branch_name و customer_city نیاز داریم که در دو schema Customer و Account وجود دارند و schema این دو depositor را به هم مربوط می‌سازد پس داریم:

Custome (customer_name, customer_street, *Customer_City)
account (account_number, *branch_name, balance)
depositor (customer_name, account_number)

از natural join این سه schema 6 ستون حاصل خواهد شد که تنها به یک ستون احتیاج داریم. ابتدا به صورت سطری انتخاب می‌کنیم سطرهایی را که customer_city آنها Hamison، سپس ستون branch_name را پراجکت می‌کنیم:

$$\Pi_{\text{branch-name}}\left(\sigma_{\text{customer-city}=\text{"Hamison"}}(\text{customer} \bowtie \text{account} \bowtie \text{depositor})\right)$$

Subject:
Year. Month. Date. ( )

مثال (P.58) نام مشتریهایی را پیدا کنید که هم وام گرفته‌اند و هم در بانک حساب دارند.

دو schema داریم:
depositor ← مشتریهایی که در بانک حساب دارند.
borrower ← مشتریهایی که از بانک وام گرفته‌اند.
اگر از عملگر ⋈ استفاده نکنیم و Π و ∩ می‌توانیم پاسخ صحیح را به دست آوریم اما ⋈ کار را آسان می‌کند

$$\Pi_{\text{customer-name}}(\text{depositor}) \cap \Pi_{\text{customer-name}}(\text{borrower})$$

$$\rightarrow \Pi_{\text{customer-name}}(\text{depositor} \bowtie \text{borrower})$$

نکته: چنانچه دو رابطه r(R) و s(S) هیچ attribute مشترکی نداشته باشند آنگاه:

$R \cap S = \emptyset \Rightarrow r \bowtie s = r \times s$

توجه: عملگر ÷ مناسب Queries است که در آنها "For all" آمده، البته این عبارت همیشه موارد صدق نمی‌کند.

جدول Natural join:

فرض کنید می‌خواهیم برای exp. زیر table رسم کنیم:

$$\Pi_{\text{customer-name, branch-name}}(\text{depositor} \bowtie \text{account})$$

$(A_1$, میرباد, 10)		(A_1, آرین)		(آرین, 10, میرباد)	
(A_2, وفا, 20)	⋈	(A_2, آرین)	=	(آرین, 20, وفا)	
(A_3, آبادان, ∅)		(A_3, رضا)		(رضا, ∅, آبادان)	
مشترک				Π	Π

توجه: G مقایسه می کند اما از آنجایی که یک عملگر unary است باید دو رابطه در ابتدا ضرب شوند. G رکورد به رکورد شرط مورد نظر را چک می کند و پس از اتمام رکوردها پاسخ هایی را به ما می دهد.

Extended Relational Model:

عملگرهای جبر رابطه ای از چندین روش گسترش می یابند یکی از ساده ترین روش ها استفاده از عملگرهای محاسباتی است. دو روش مهم برای توسعه مدل رابطه ای عبارتند از:

1. aggregate op.
2. outer-join op.

Generalized projection:

$\Pi_{F_1, F_2, \ldots, F_n}(E)$

E: می تواند هر exp. دلخواهی باشد

F_i: هر عبارت جبری می تواند باشد

مثال: می خواهیم ببینیم هر شخص چقدر اعتبار دارد؟ یعنی باید مقدار موجودی شخص را از limit (مقداری که بانک برای هر شخص می پردازد) آن شخص کم کنیم و این کار را با Gen. proj. امکان پذیر می باشد.

$\Pi_{\text{customer-name},\ \text{Limit} - \text{credit-balance}\ \text{as}\ \text{available}}(\text{credit-info})$

«credit-info»

customer-name	Limit	credit-balance
اسماعیل	2000	1500
سنیا	1000	500
علی	1500	800

→

customer-name	available
اسماعیل	500
سنیا	500
علی	700

limit ≥ credit-balance

limit ← کل مقداری که هر شخص می تواند خرج کند

credit_balance ← کل پولی که هر شخص در حساب دارد

همانطور که در مثال می بینیم در Π از عبارت جبری تفاضل استفاده کردیم و حاصل این تفاضل را با کلمه کلیدی as نامگذاری کردیم و پاسخ Π دو ستون به نام های customer_name و available دارد.

تفاوت set و Multiset:

در set تکرار معنا ندارد یعنی در مجموعه تکرار مهم نیست اما در Multiset تکرار مهم است:

set → {1,1,2,2} = {1,2}

Multiset → {1,1,2,2} ≠ {1,1,2,2}

Aggregate Functions

$G_{F(A_1),\dots,F(A_n)}(E)$

مجموعه‌ای از مقادیر را می‌گیرد و یک مقدار را برمی‌گرداند؛ aggregate Fun.

aggregate Fun. در Multiset کار می‌کند یعنی تکرار اهمیت دارد.

- مجموعه {1,1,3,4,4,11} را در نظر بگیرید؛

G_{sum} ← aggregate function sum : مجموع مقادیر را برمی‌گرداند ← 24

G_{avg} ← aggregate function avg : میانگین مقادیر را برمی‌گرداند ← 4

G_{count} ← aggregate function count : تعداد اعضای مجموعه را می‌دهد ← 6

G_{min} ← aggregate function min : کوچکترین عضو مجموعه را می‌دهد ← 1

G_{max} ← aggregate function max : بزرگترین عضو مجموعه را می‌دهد ← 11

$G_{distinct}$ ← aggregate function distinct : از هر تکرار یکی را می‌شمارد (تعداد تمایزها) ← 4

نکته: distinct همان count است اما در set

مثال: مجموع حقوقی که شرکت به کارمندانش می‌دهد

$G_{sum(salary)\ as\ sum\text{-}salary}(employee)$

در جدول (schema) employee

مجموع salary ها را برمی‌گرداند و نام ستون پاسخ را sum_salary می‌گذارد ← پاسخ فقط یک سطر و یک ستون دارد.

P4PCO

Subject: اصول طراحی پایگاه داده‌ها
Year 86 Month 8 Date 19 (15)

« employee »

employee-name	salary	branch-name
یوسف	1,000	میرداماد
طاهره	5,000	ونک
پژمان	12,000	ونک
محمد	6,000	میرداماد

→

sum-salary
24,000

مثال: بیشترین مقدار حقوق را مشخص کنید.

$\mathcal{G}_{max(balance)}$ (account) , account #

مثال: مجموع حقوق هر شعبه را مشخص کنید. (p. 62)

$\mathcal{G}_{branch-name}$ (employee)
sum(salary) as sum-salary

بر حسب نام شعبه گروه بندی می‌کند

employee-name	salary	branch-name
یوسف	1,000	میرداماد
محمد	6000	میرداماد
طاهره	5,000	ونک
پژمان	12,000	ونک

→

branch-name	sum-salary
میرداماد	7,000
ونک	17,000

مثال: مجموع حقوق در هر شعبه، Max حقوق در هر شعبه؟ (p. 67)

$\mathcal{G}_{branch-name}$ (employee)
sum(salary) as A , max(salary) as B

branch-name	A	B
میرداماد	7000	6000
ونک	17000	12000

P4PCO

Subject:
Year. Month. Date. ( )

پس به طور کلی داریم:

$$(G_1, G_2, \ldots, G_n)\ \mathcal{G}^{(E)}_{F_1(A_1), \ldots, F_n(A_n)}$$

G_i ← : attهایی که در سمت چپ $\mathcal{G}$ می‌آیند باعث گروه‌بندی می‌شوند یعنی $\mathcal{G}$ ابتدا بر طبق att. G_i ورودی‌ها را گروه‌بندی کرده و سپس عملیات لازم را انجام می‌دهد

F_i ← : عملیاتی است که باید انجام شود مثل sum و avg و...

A_i ← : attهایی است که عملیات F_i باید روی آن‌ها انجام شود

مثال: مجموع حقوق را بر حسب نام شعبه و نام شهر بدهد یعنی مجموع حقوق هر شعبه، هر شهر را بدهد

employee

city	branch-name	employy-name	salary
تهران	ونک	علی	1000
تهران	ونک	رضا	5,000
اصفهان	بهشتی	زهرا	11,000
اصفهان	بهشتی	یوسف	1,000
تهران	بهشتی	محمد	3,000
تهران	بهشتی	سینا	12,000

باید دو گروه‌بندی انجام شود یعنی:

G_1 = branch_name

G_2 = city

و خواست $\mathcal{G}$ داریم:

F_1 = sum

A_1 = salary

$$\text{city}, \text{branch-name}\ \mathcal{G}\ \text{sum(salary)}\ \text{as}\ \text{sum-salary}\ (\text{employee})$$

(G_1: city, G_2: branch-name, F_1: sum, A_1: salary)

city	branch-name	sum-salary
تهران	ونک	6,000
تهران	بهشتی	15000
اصفهان	بهشتی	12,000

پاسخ سه ستون دارد که درستی‌اش را G_1 و G_2 تضمین می‌دهند و ستون سوم را $F_1(A_1)$.

PAPCO