

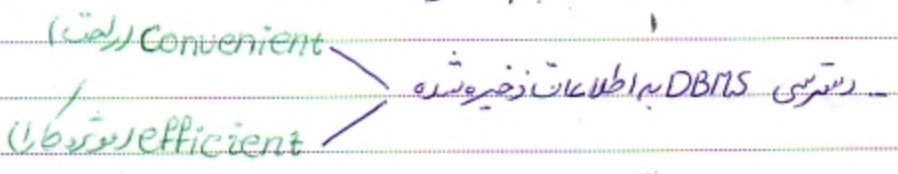
چنانچه اندک را از خود با نسیب عشق آتش نگیرد...

جلسه اول 86.6.31 ← فصل دهم →

DBMS: Data Base Management System

وظایف DBMS:

- 1. حجم عظیم از اطلاعات را در فایل نگهداری می کند
- 2. به وسیله یک سری از برنامه ها به این اطلاعات (داده های ذخیره شده) دسترسی پیدا می کند
- و این دسترسی به دو صورت انجام می گیرد:



معایب عدم وجود DBMS:

- 1. Data redundancy (افزونی داده ها) - حافظه اضافی اشغال می شود و باعث حجم بالای شود
- 2. inconsistency (عدم همخوانی) - قسمت های مختلف سیستم با هم هماهنگی ندارند
- 3. مشکل در دسترسی داده ها

E.F. Codd

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

- (1) Atomicity (تجزیه ناپذیری)؛ یک عمل یا کامل انجام می شود و یا اصلاً انجام نمی شود
 - (2) Consistency (هم خوانی)؛ قسمت های مختلف با هم هماهنگ هستند
 - (3) Isolation (جدا پذیری)؛ همروزی کاربرها تأثیر دیگری نمی گذارند (بر روی دیگری)
 - (4) Durability (دوام پذیری)؛ برای داده های ذخیره شده حفاظت امنیتی و فیزیکی صورت گیرد
- داده ها از بین نروند. (داده ها باید با ایلاتر باشند)

Abstraction: نقش برداشت هر لایه واقعیت را گویند یعنی به جزئیات توجه نمی داریم

Data Abstraction: (طرح داده) (انتزاعی)

- 1, Physical level → (داده چه چیزی ذخیره می شود) نحوه ذخیره شدن داده ها
 - 2, Logical level → (اجزای داده ای ذخیره می شود) نوع داده های ذخیره شده
 - 3, View level → (بر اساس نیاز کاربر اطلاعات مورد نیاز را در اختیار می گذارد)
- ↓ بالاترین level

Schema: ساختار data base بدون در نظر گرفتن داده ها را schema گویند و کمترین تغییرات در schema تغییر کند یعنی رکورد ها اضافه کرده باشیم

Schema instance: نمونه هایی از schema در هر لحظه (یعنی داده ها در این لحظه Schema هستند)

Data models:

Data models مجموعه ای از ابزارهای مفهومی برای توصیف داده ها، ارتباط داده ها، مفاهیم نقش داده ها (semantic)، معنایی و محدودیت (Constraint) داده ها برای حفظ یکپارچگی داده ها است؛

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

جلسه دوم 86.7.7 ← فصل (6) →

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

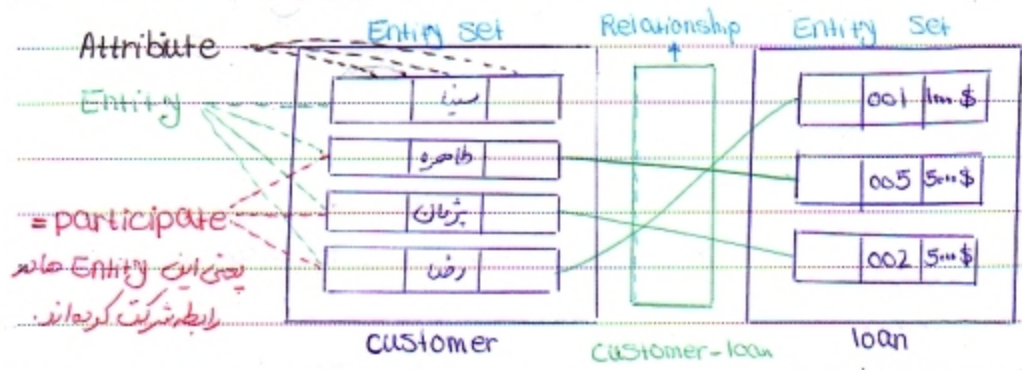
رابطه موجودیت

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

Set (مجموعه): گروهی از موجودیت‌ها (چیزها) که ویژگی مشترکی دارند را یک set گوئیم.

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

مثال: در زیر دو Entity Set به نام‌های Customer (مشتری) و loan (وام) وجود دارد که هر کدام از آن‌ها به ترتیب شامل 4 و 3 Entity هستند که بین این دو Entity sets روابط تعریف شده است این Relationship Customer-loan نام می‌نمایم.



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

Attributes: اطلاعاتی که به Entity به نامی دهد و به طور کلی به دسته های زیر تقسیم می شود:

Simple (ساده): صفت ساده مثل اسم، شماره دانشجویی و...

Composite (مترکب): صفتی که بتوان آن را به صفت های کوچکتر تقسیم کرد مثل آدرس

Single valued (یک مقداری): صفتی که هر Entity تنها یک مقدار را بتواند قبول کند

مثلاً شماره دانشجویی برای هر دانشجو یک صفت یک مقداری است.

Multi-valued (چند مقداری): صفتی که هر Entity بتواند چند مقدار آن را قبول کند

مثلاً شماره تلفن که هر Entity می تواند بیش از یک شماره تلفن داشته باشد.

Derived (درستق شده): یعنی مقدار آن از At دیگری گرفته شده باشد مثلاً At

سن که از روی At تاریخ تولدی سببی شود (سن از تاریخ تولدستق شده) و یا معدل که

از روی نمرات دروس درستق شده است.

Relationship Set: در زیر به تعریف ریاضی مجموعه روابط می پردازیم، باید توجه داشته باشیم که

Relationship set مجموعه ای از چندتایی ها است.

Relationship Set:

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

از چندتایی ها / Entity Sets entity: x

دفعات قبل دو Entity Sets تابع ← درجه 2 است.

← تعداد Entity Set = درجه Relationship Set

مجموعه 2 تایی

customer - loan = { (002, پیمان), (001, رضا), (005, طاهر) }

customer = { پیمان, سينا, رضا, طاهر }

loan = { 001, 002, 005 }

* **محدودیت ها (Constraints)**: بر سر مقدار:

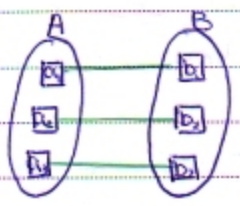
1* mapping Cardinalities → صحبت روی تعداد map ها را بنویسید

تعداد پذیری نگاشت

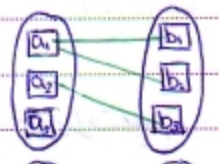
mapping: نگاشت بین دو چیز و نسبت دادن را mapping می نامیم.

Cardinalities: یعنی تعداد پذیری به این معنی که اجازه داشته باشیم یک چیز را چند بار map کنیم. مثلاً در مثال قبل به رضا می نامیم و ام 001 و سیم و ام 002 تعلق بگیرد.

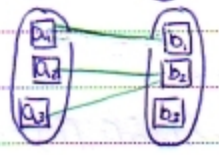
* در زیر به بررسی 4 mapping Cardinalities می پردازیم: (با توجه به مثال قبل)



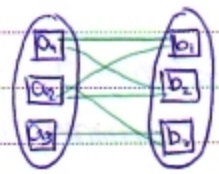
One to One: هر مشتری تنها می تواند یک وام بگیرد و هر وام تنها می تواند به یک مشتری تعلق گیرد.



One to many: هر مشتری می تواند صفر یا چند وام بگیرد ولی هر وام تنها باید به 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)

محدودیت شرکت کردن

مثلاً تعیین حداقل و حداکثر شرکت کردن به مشتری در وام گرفتن (بسیار مرتبط با mapping)

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

3 * keys (کلیدها): کلیدی Entity Sets می تواند یک یا چند Attribute باشد که آن At. باید منحصر به فرد باشد.

Super keys (ابر کلید): به کلیدی گفته می شود که بتواند به کلیدهای وابسته شود یعنی چند کلید در زیر مجموعه خود داشته باشد به طور کلی به کلیدی Super keys گفته می شود که در دل خود یک یا چند کلید داشته باشد. مثلاً اگر فرض کنیم نام آردان خانوادگی و شماره دانشجویی کلید باشد حتی Super keys خواهد بود.

Candidate keys: به کلیدی که قابلیت انتخاب شدن دارد Candidate keys گوئیم، در واقع Super keys که شکسته نشود Candidate keys نامند. مثلاً کلید (اسم، شماره، شماره شناسنامه) یک Candidate keys است زیرا چنانچه این کلید را بشکنیم و هر قسمتش را براریم دیگر کلیدی مهمی داشت.

توجه: جهت است Attribute یعنی نام عنوان Candidate keys در نظر بگیریم. مثلاً بین (نام آردان خانوادگی و شماره شناسنامه) و (شماره دانشجویی) جهت 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.

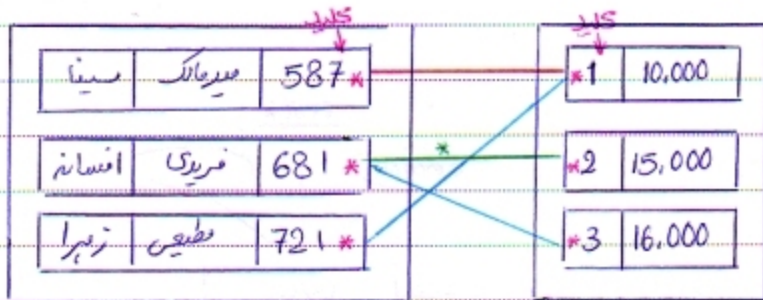
کلاس 14.7.86

8 نکته: $primary\ key \subseteq candidate\ key \subseteq super\ key$

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

نکته: در Relationship set برای primary key هم است

مثال: دو Entity set زیر را در نظر بگیرید



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

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

باتوجه به mapping cardinalities ممکن است زوج (681, □) تکرار شود.

- one-to-one: هیچ زوجی تکرار نمی شود و کلید ارتباط می تواند کلید Customer loan باشد
- one-to-many*: زوج تکراری خواهد داشت؛ مثل (681, 3) و (681, 2)
- many-to-one*: زوج تکراری خواهد داشت؛ مثل (721, 1) و (587, 1)

ممانعتی که می بینیم در سمت one است تکرار وجود دارد بنابراین برای حل این مشکل
 کلید سمت many را به عنوان کلید ارتباط در نظر می گیریم.
 در حالت many-to-many اجتماع کلید بردو سمت کلید ارتباط را تشکیل می دهند.

Entity Relationship Diagrams:

- قراردادهای رسم Diagram به صورت زیر است:
- منظور: Entity set نشان می دهد.
- نوعی: Relationship نشان می دهد.
- بیضی: Attribute نشان می دهد.

خطوط و نشانه‌های مختلف diagram را بچشم وصل می‌کنند:



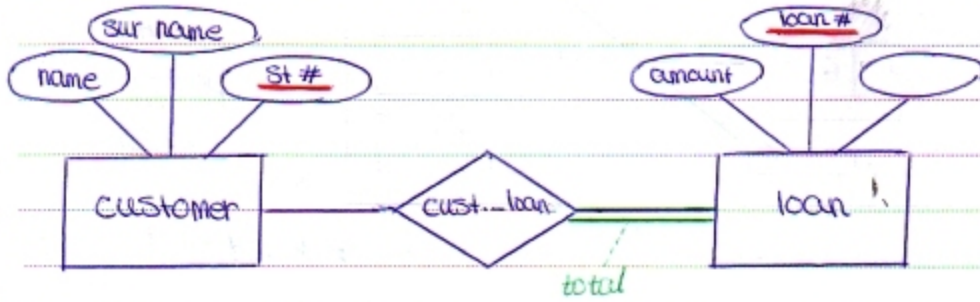
نشانه دهنده many است.

نشانه دهنده one است.

نور فلش به سمت ستاره

one-to-many : loan به Customer

many-to-one : Customer به loan



total: در Entity set که همه Entity هایش باید حتماً یکبار در رابطه شرکت کنند آن Entity set را total می‌نامند و خط دوجمله نشان داده می‌شود. مثلاً در اینجا چه‌رومی باید حداقل یکبار در ارتباط شرکت کند.



multivalued att.: Double ellipses نشان می‌دهد.



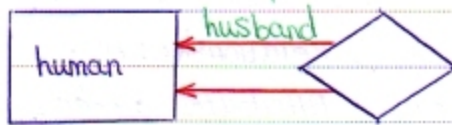
derived att.: Dashed ellipses نشان می‌دهد.

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

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

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

entity نقش: role



one-to-one

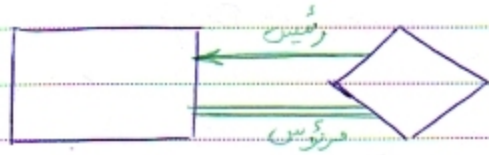
نکته: در برخی موارد لازم است از روابط بازگشتی استفاده کنیم.

مثلاً اگر مجموعه انسانها را در نظر بگیریم برای نمایش

ارتباط زنانه‌سویی، ارتباط دوستی و ... باید از خطوط

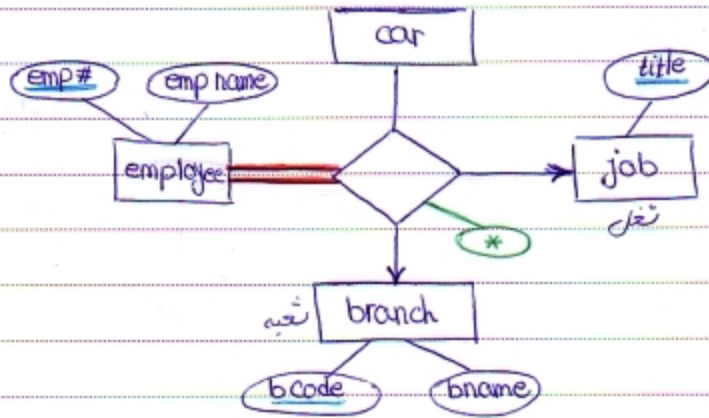
بازگشتی مطابق شکل استفاده کنیم زیرا زن و شوهر هر دو در

همان مجموعه انسانها هستند



از معاون به رئیس many-to-one
 از رئیس به معاون one-to-many

سوال: رابطه رئیس و معاون بودن هم یک ارتباط بازگشتی است. با توجه به اینکه کسی نمی تواند در ارتباط معاون بودن شرکت کند بنابراین total است. و فرض می کنیم هر کس یک رئیس مستقیم داشته باشد.



سوال: در اینجا یک Relationship چهار تایی داریم

* همانطور که می بینیم می توانیم برای Relationship خاص attribute داشته بگیریم و این بدیهه وقتی رخ می دهد که آن At. به همه entity set مربوط باشد

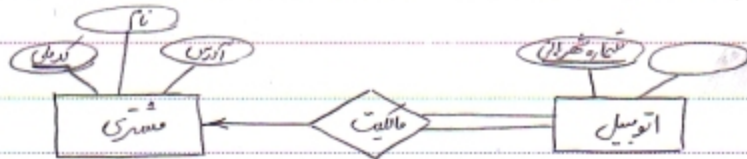
* total است زیرا هر طرف باید job داشته باشد یعنی در همه ارتباطات باید شرکت کند زیرا در غیر این صورت دیگر شخص طرفند نیست.

توجه: این سوال دارای ابهام است زیرا دارای دارای دو بخش است (one) مثلا می توان ارتباط یک ترکیب از employee و car — با یک ترکیب از job و branch تعیین کرد. یا یک job و یک branch تعیین کرد.

برای رفع ابهام در ارتباطات بیشتر از دوایی باید تنها یک ارتباط را با فلش تعیین کنیم در غیر این صورت مبهم خواهد بود.

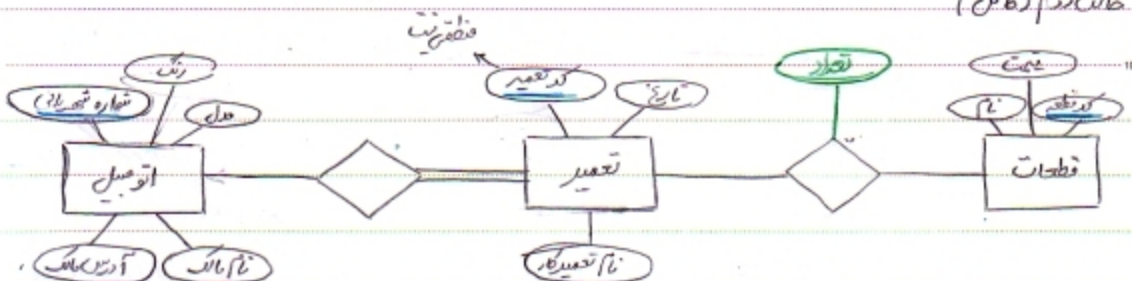
مثال : Diagram تعمیرگاه را رسم کنید که اطلاعات متبجها، اتوسیل ها و قطعات تعویضی در آن نشان داده شود.

می توانیم متبجی را یک entity set در نظر بگیریم و یا متبجی را به عنوان یک entity set در نظر بگیریم که حالت دوم بهتر است.



حالت اول (زیر)

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



حالت دوم (اصلی)

تعداد قطعات هم به تعمیر و هم به قطعات بستگی دارد بنابراین برای هر دو Relationship قرار می گیرد.

توجه : اگر ارتباط اولی را به تایی در نظر می گیریم و قطعات را هم به آن منقل می گیریم Diagram اشتباه بود زیرا در آن صورت هر ماشین که به تعمیرگاه می ریزد باید قطعه ای را هم تعویض می کرد در صورتی که امکان دارد اتوسیل بدون تعویض قطعه تعمیر شود.

تعداد 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 منقل کرد (به راحتی) به همین دلیل ER بسیار مورد توجه و استفاده قرار گرفته است.

مثال: می خواهیم مثال قبلی 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
 Entity set ستونهای جدول را تشکیل می دهند و ستونهای جدول
 Relationship set را کلید ارتباط و همچنین At: های روی ارتباط تشکیل می دهند و سطوحی
 این جدول شرکت کننده هستند.

جدول اتومبیل

شماره شهربانی	رنگ	مدل	نام مالک	آدرس مالک
۲۵۳ ۶۴۵	سبز	۱۳۷۹	علی	تهران
۱۵ ق ۱۴۱	نقره‌ای	۱۳۸۹	سارا	تهران
۴۵۳ ۱۹۹	سبز	۱۳۸۲	مینا	تهران

جدول تعمیر

کد تعمیر	تاریخ	نام تعمیرکار
1	۱۹/۷/۱۲	رضا
2	۱۹/۹/۹	محمد
3	۱۹/۳/۵	من
4	۱۹/۷/۹	محمد

جدول قطعات

کد قطعه	نام	تیمت
1	سکینت	
2	شمع	
3		

چون رابطه اول many-to-many طرد هر دو
 Entity set را باید در جدول بنویسیم + (دو ستون)
 در ارتباط دوم سه ستون داریم زیرا علاوه بر کلید
 یک ستون هم به At: تعداد تعلق می گیریم.

کلید اتومبیل

شماره شهربانی	کد تعمیر
۲۵۳ ۶۴۵	2
۲۵۳ ۶۴۵	3
۴۵۳ ۱۹۹	1
۱۵ ق ۱۴۱	4

کلید قطعات

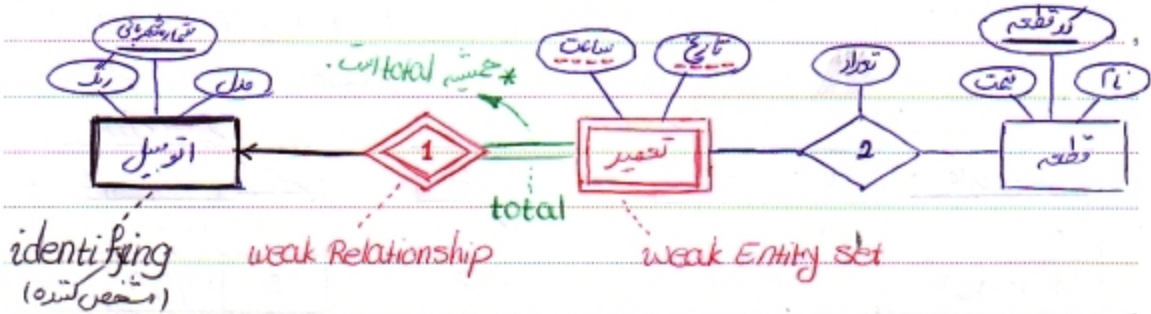
تعداد	کد قطعه	کد تعمیر
1	1	2
4	2	2

جدول ارتباط (R)

جدول ارتباط (R)

جلسه چهارم 28.7.86

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

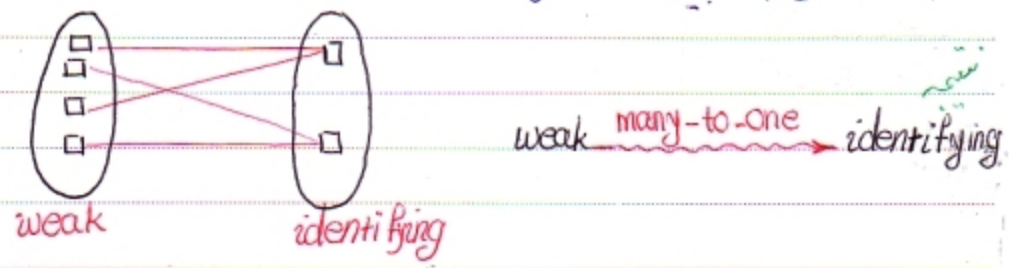


طید weak entity set یا partial key می نامند چون ذاتاً طید نیست اما در تشکیل طید
دخالت دارد مثلاً در اینجا (ساعت، تاریخ) partial key هستند.

نرمه: برای تعیین طید weak entity set کافی است partial key + attribute
وابسته را با هم در نظر بگیریم، یعنی در اینجا طید انوسیل است (تعمیر ساعت، تاریخ شماره شغلی) است.

نکته: اگر چند attribute با هم یک primary key را در یک Entity set تشکیل دهند
آن Entity set نمی تواند weak باشد مثلاً ضابطه در مثال قبل attribute
دیگر به عنوان نا تغییر کار نمی بود دیگر تغییر یک weak ent. نبود.

* هر Entity در Entity set تغییر فقط و فقط به یک انوسیل وابسته است بنابراین
ارتباط از تعمیر به انوسیل many-to-one است.



Strong Entity Set: هر انیتی ستی که weak نباشد Strong است یعنی انیتی ستی که primary key دارد مستقل است Strong گفته می شود.

مثال قبل را در نظر بگیرید:

جدول تغییر

جدول ارتباط (1)

جدول ارتباط (2)

تعداد	گروه	ساعت	تاریخ	شماره شعبه	ساعت	تاریخ	شماره شعبه

قابل حذف

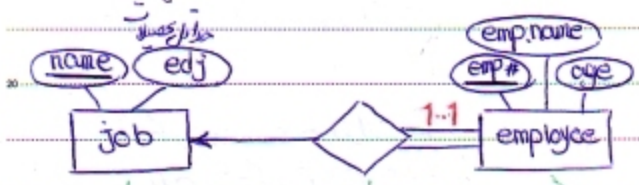
وقتی می توانیم چیزی را حذف کنیم که اضافی باشد پس از حذف اطلاعاتی از سیستم بار نماند. خوشبختی به سیستم وارد نماند در اینجا هم به این صورتی بینیم Table ارتباط (1) و تغییر کاملاً مثل هم بوده یعنی از آنجا قابل حذف است.

دلیل نگاهت دو جدول: ۱- ارتباط مربوط به Entity Set تغییر total است و ۲- ارتباط many-to-one از تغییر به اتومیل: سب شده این دو جدول کاملاً شبیه است.

چون ارتباط many-to-one داشته باشیم که many این total باشد می توانیم 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! employee به جدول اضافه کنیم.

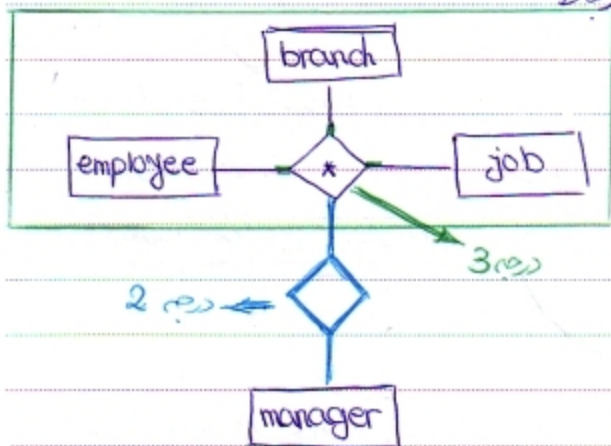
name	edj	name	emp.#	emp.#	age	emp.name

قابل حذف

name	emp.#	age	emp.name

: (Extended ER) EER

aggregation: (تجمع) یعنی یک سمت ارتباط



* این ارتباط مشخص می کند چه کسی چه شغل دارد و در کدام شعبه مشغول به کار است. ممکن است یک شخص در دو شعبه در مشغول متفاوت داشته باشد اما در هر شعبه یک رئیس خواهد بود. بنابراین نمی توانیم بین کارمند و رئیس relationship بگذاریم لذا مجبوریم از یک Relationship دیگر استفاده کنیم.

(شعبه، شغل و کارمند) و رئیس

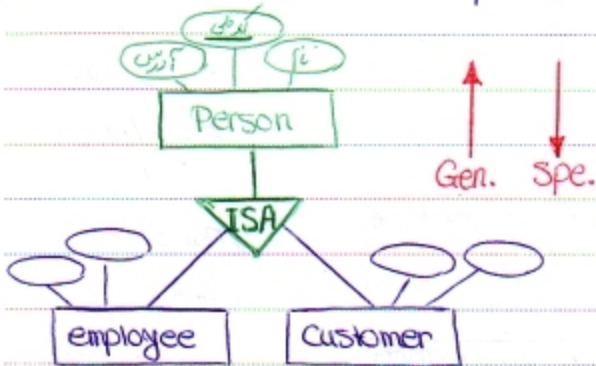
Entity روشی برای این Diagram اضافه کنیم

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



: Specialization & Generalization

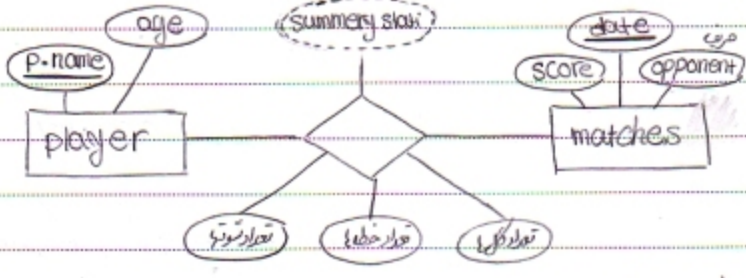
گاهی اتفاق می افتد که دو Entity Set دارای attribute های مشترک و مشابه باشند. در این صورت بهتر است یک انیتی ست جدا تعریف کنیم و attribute های مشترک را به آن اختصاص دهیم.



مثال: مثلاً کارمند و مشتری هر دو انسانند و دارای attribute مشترک مثل نام، آدرس و... هستند. بنابراین ما نباید شکل رو به روی آن را پیاده سازی می کنیم. یعنی از استفاده می کنیم.

تمرین 6.4 / P. 256

حداکثر آمی رسم کنید که تیم مورد علاقه تان ، سابقاتش ، امتیازاتی که در هر مسابقه کسب می کند ، بازیکنان ، تیم در هر مسابقه ، آمار بازی ذخیره شود .



داریم دو عدد یک تیم (تیم مورد علاقه تان) صحبت می کنیم بنابراین لازم نیست مثلاً نام تیم را ذخیره کنیم فقط کافی است نام حرفه را ذخیره کنیم و در این صورت date می تواند کلید باشد چون یک تیم در یک تاریخ فقط می تواند در یک مسابقه شرکت کند بنابراین date برای matches یک primary key است.
 و آنجا جایی که Att های مثل تعداد خطاها و تعداد شوتها ... تیم به بازیکن و تیم به مسابقه بستگی دارد باید آنجا را در Relationship تعریف کرد.

جلسه پنجم 8.5 . 86 ← فصل (2) →

Relational model : این مدل ، یک مدل موفق است لادین مدل برای درخواستهای تجاری (فهم ترین عامل در این مدل ارتباطات است).

Domain : دامنه 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 می تواند داشته باشد 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 attribute است.

Basic Structure: در زیر جدول account relation را می بینیم که سه ستون دارد که

D_1 : مقدار برای $account \#$ ، D_2 : مقدار برای $branch-name$ و D_3 : مقدار برای $balance$ را شامل می شود و

هر سطر جدول شامل سه tuple (اول، دوم، و سوم) است. هر سطر $tuple =$ ^{موردی} _{حزب}

بنابراین $account$ زیر مجموعه ای از $D_1 \times D_2 \times D_3$ است.

تیمم: در حالتی که این table شامل n attribute زیر مجموعه ای از $D_1 \times D_2 \times \dots \times D_n$ می باشد.



account

account #	branch-name	balance
A1	میرباد	1...1-1-
A2	ون	0
A3	ون	1...1-

رابطه = $D_1 \times D_2 \times D_3$
 { ون و میرباد } \times { A1, A2, A3 } \times { 1...1-1-, 0, 1...1- }
 = { (A1, میرباد, 1...1-1-), (A2, ون, 0), (A3, ون, 1...1-) }

tuple tuple

نکته: برای نمایش tuple variable از نماد $t[]$ استفاده می شود.

$t[account \#] = "A1"$

$t[branch-name] = "میرباد"$

tuple variable relation $t \in r$

Schema: نشان می دهد رابطه چه attr. هایی دارد و با صرف نظر از نامی شود مثال:

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

مثال int

مثال a

$account(Account_schema)$

$account(account \#, branch-name, balance)$

OR

مثال int

مثال a

در اینجا رابطه از نوعی که خودمان تعریف کردیم را ایجاد کردیم مثل

relation

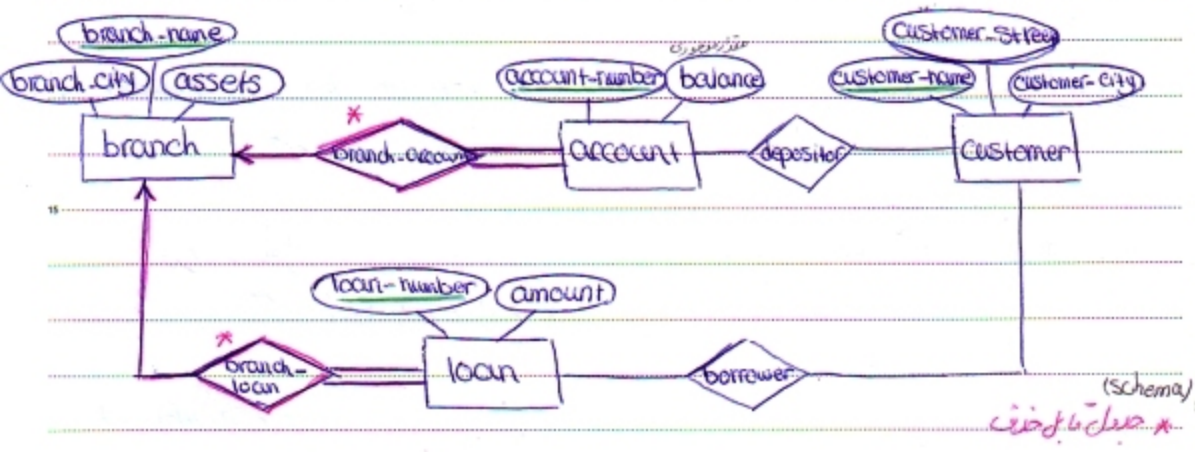
تویها: همانطور که قبلاً گفته شد هر tuple variable یا تاپل t نشان دهنده یک مورد است. این طراز یک جدول که ممکن است در هر لحظه یک مقدار داشته باشد.

R-stance: مقادیری که ارتباط موردتویها در هر لحظه میگیرد.

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

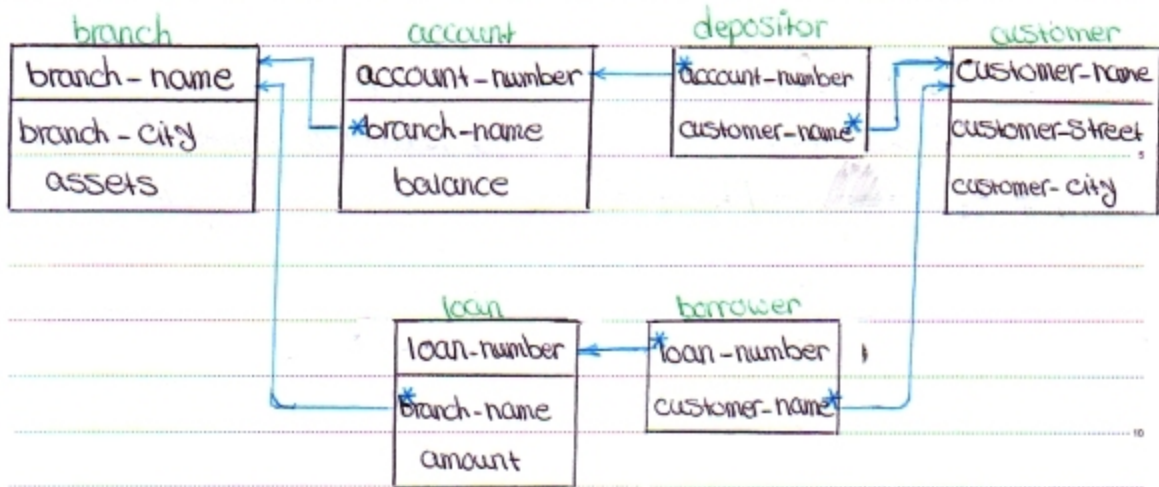
$$t_1 \neq t_2 \Rightarrow t_1[k] \neq t_2[k] \quad K \subseteq R$$

مثال:



- 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)

طرح Schema Diagram مثال قبل توضیح کنید:



* این مقادیر foreign key هستند یعنی مقادیرشان وابسته بوده و از جدول دیگر تائید می‌گردد یعنی در این schema دیگر کلید هستند

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

بازیابی اطلاعات:

به طور کلی سه نوع پرس و جو در مدل رابطی وجود دارد:

1- Relational Algebra (جبر رابطی)

2- Tuple Relational calculats (حساب رابطی رکوردی) (تاپلی)

3- Domain Relational calculats (حساب رابطی دامنه‌ای) (DRC)

1- Relational Algebra: جبر رابطی دارای 6 عمل اصلی (operations) می‌باشد

که سه‌تایی آنها unary و سه‌تایی دیگر binary هستند عمل‌های unary

رابطه = (رابطه)
 روی بی relation انجام می شوند و عملهای binary روی دو عمل انجام می گیرند؛
 و این نشان عمل عبارتند از:

- 1. Selection (unary) $\rightarrow \sigma_p^{\uparrow}(r)$ Relation رویی
- 2. projection (unary) $\rightarrow \Pi_A(r)$ A $\subseteq R$
- 3. Union (binary) $\rightarrow U$
- 4. Set difference (binary)
- 5. Cartesian product (binary) $\rightarrow \times$
- 6. rename (unary) $\rightarrow \rho^{\uparrow}(E)$ exp. نام دهنده

1. Selection این عمل unary به ما این امکان را می دهد که tuple هایی با شرایط خاص را انتخاب کنیم (را انتخاب کرده و آنهایی را بقیه جدا کنیم؛ در قسمت شرط می توانیم از علائم =، ≠، >، <، >=، <=، <<، >>، &، &or و (or) استفاده کنیم - (not) استفاده کنیم
 مثلاً می خواهیم کسانی که در شعبه vanak بوده و مقدار وامشان از 1000 بیشتر است را انتخاب کنیم

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

که جدول به صورت زیر خواهد شد:

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

2. projection این عمل unary این امکان را به ما می دهد که ستونی که به آن احتیاج نداریم را از جدول حذف کنیم مثلاً فرض کنید می خواهیم نسبت جهت شماره و مقدار وام را بقیه کنیم و شعبه برای ما اهمیت ندارد در آن صورت داریم:

* $\Pi_{loan-number, amount}(loan)$ schema تغییر می کند

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

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

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

انتخاب مشتریهایی که در Harrison زندگی می کنند

همانگونه ستون نام مشتریها و نامش آن

چنانچه می بینیم باید ستون نام مشتریها را توسط Π از بقیه جدا کرده و با σ هم ورودیها را انتخاب کنیم تا در این یک جدول یک ستون (نام مشتری) داشته باشیم.

3. Union: این عملگر اجتماع به ما این امکان را می دهد که چندین جدول 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:

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

* نکته: Union بین مجموعه هایی استفاده می شود که Schema آنها از نظر تعداد و نامند یکسان باشد.

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

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

جلسه ششم 86.8.12

5. Cartesian-product. این عملیه با این امکان برای دهنده که دو مجموعه رابطه را در هم ضرب (رابطه‌ای) کنیم (۲ یا بیشتر) که نتیجه این ضرب رابطه‌ای باید یک رابطه باشد.

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

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

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

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

چون نتیجه موارد att تکراری وجود دارد بنابراین اجهامی هم نخواهیم داشت لذا دیگر اجهام نیست
 نام رابطه را در کنار att‌ها این می‌نویسیم.

نکته: در واقع ضرب رابطه‌ای به ما این امکان را می‌دهد که دست‌نخورده‌های روابط مختلف را در کنار هم ایجاد کنیم.

در حالت‌های تفاوت آنها را با هم مقایسه کنیم. هر چند ضرب یک عمل بسیار ارزشمند و ضروری است.

borrower		borrower x loan				
loan-#	customer-name	borrower.loan #	customer-name	loan.loan #	branch-name	amount
L ₁	آرین	L ₁	آرین	L ₁	میرداماد	10
L ₂	علی	L ₁	آرین	L ₂	ونف	20
L ₃	افسانه	L ₁	آرین	L ₃	میرداماد	30
		L ₂	علی	L ₁	میرداماد	10
		L ₂	علی	L ₂	ونف	20
		L ₂	علی	L ₃	میرداماد	30
		L ₃	افسانه	L ₁	میرداماد	10
		L ₃	افسانه	L ₂	ونف	20
		L ₃	افسانه	L ₃	میرداماد	30

سؤال: نام افرادی که از شعبه میراماد وام گرفته اند را بیامید.

نام افرادی که وام گرفته اند در رابطه borrower و شعبه ای که وام گرفته اند در loan نامده طری میشود بنابراین باید از ضرب طری این دو مجموعه استفاده کنیم

در $borrower \times loan$ در هر جدول باید ویژگی‌هایی را انتخاب کنیم که # loan یکسان دارند زیرا اگر $borrower$ ، $loan$ و $loan$ با هم یکی نباشند آن طری بی‌معنا خواهد بود بنابراین داریم:

$$\Pi_{\text{customer-name}} \left(\sigma_{\left(\begin{array}{l} borrower.loan\# = loan.loan\# \\ \text{branch-name} = \text{"میراماد"} \end{array} \right)} (borrower \times loan) \right) =$$

customer-name
آمین
امانه

6. Rename: این عمل اصلی به ما این امکان را می‌دهد که نام هر exp مورد نظر حتی Att. جای آن را در صورت نیاز تغییر نام دهیم. مثلاً فرض کنید می‌خواهیم borrower را در خودش ضرب کنیم در این صورت باید حتماً از عملر rename استفاده کنیم زیرا در غیر این صورت اساس تکراری خواهیم داشت.

$$borrower \times borrower = \rho_x (borrower)$$

$$= (borrower.loan\#, borrower.customer-name, x.loan\#, x.cus\#)$$

توجه: می‌توانیم نام Att برای هر schema را نیز تغییر دهیم؛ مثال:

$$\rho_x (borrower) \rightarrow x = borrower;$$

$$A_1 = loan\# \quad \text{و} \quad A_2 = customer-name$$

و ضایحه خواهیم فقط نام یکبار از Att. ها را تغییر دهیم باید نام تغییر را عیناً نوشته و نام مورد نظر برای آن Att بنویسیم. (مثلاً به جای A_1 همان $loan\#$ را بنویسیم)

تعریف: ریس از جبر رابطی: یک exp در جبر رابطی شامل ۱- رابطه در DB و ۲- یک رابطه ثابت (constant) می‌باشد.

اگر E_1 ، E_2 در جبر رابطی expression باشند آنگاه تمامی روابط زیر نیز exp های جبر رابطی خواهند بود:

- $E_1 \cup E_2$ → $E_1 - E_2$ → $E_1 \times E_2$
- $\sigma_p(E_1)$ → $\Pi_R(E_1)$ → $\rho_x(E_1)$

عملگرهای اضافی در جبر رابطی:

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

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

1. Set-Intersection: اولین عملگر اضافی ∩ (انترسکشن) است فرض کنیدی خواهم شیهای را بیابانیم که هم حساب دارند و هم وام گرفته اند در این صورت داریم:



توجه: عملگرهای ∩ را می توان به کمک چند عملگر اصلی پیاده سازی کرد: $R \cap S = R - (R - S)$

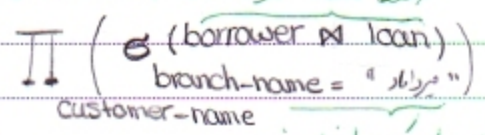
2. Natural join: در زوجی $(R \bowtie S)$ ، Att.های غیر مشترک و Att.های مشترک (لازم تکرار) ظاهر خواهند شد یعنی از Att.های مشترک فقط یک نمونه برداری می شود این عملگر فرعی نیز ترکیبی از ∪ و × است.

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

 شرط: Att.های هم نام مقادیرشان مساوی باشد.

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

مثال: نام مشترکهای که از میرزا محمد و امیرترانه اند؟



۴ از سه شرط به یک نظر خط می خورد

نکته: علاوه بر natural join یک عملگر دیگر نام join θ نیز وجود دارد که تنها فرق این دو join در شرطشان است در join θ هر شرط دلخواهی می توانیم استفاده کنیم اما در natural join شرط این است که θ های مختلفی که مقادیر آن مساوی است بدست آوریم

$$\sigma_{\theta}(R \times S) = R \bowtie_{\theta} S$$

شرط دلخواه $\theta \rightarrow$

مسئله: شماره حساب مشتری که Max مقدار موجودی را دارد برابر با r_1 برای پیدا کردن این مشتری باید balance مشتریهای مختلف را با هم مقایسه کنیم بنابراین ضرب دوطرفی به کار می آید و آنجایی که θ نمی تواند در هم ضرب شود پس باید account را در خودش ضرب کنیم و چون نمی توانیم مستقیماً Max را با هم بخرانیم که چیزیها به دست آورده و از کل کم کنیم (Max هیچ کس کوچکتر نمی شود)

متوسط

$$r_1 = \Pi_{\text{account} - \#}(\text{account}) - \Pi_{\text{account} - \text{account} - \text{number}}(\sigma_{\text{account} \times \text{balance} < x \cdot \text{balance}}(\text{account}))$$

شماره حساب مشتری Max r_1

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

3. عملگر "Division": فرض کنید خواص نام مشتریهایی که در تمام شعب تهران حساب دارند را پیدا کنیم (فرض: تهران سه شعبه دارد) 4. ستون برد

$$\Pi_{\text{customer} - \text{name}, \text{branch} - \text{name}}(\text{account} \bowtie \text{depositor}) \div \Pi_{\text{branch} - \text{name}}(\sigma_{\text{branch} - \text{city} = \text{"Tehran"}}(\text{branch}))$$

دو ستون را جدا می کند

در تقسیم نتایج مورد اعمال می شود.

r_1		r_2	
customer-name	branch-name	branch-name	customer-name
علی	ون	ون	علی
علی	پیراوار	میرزاآباد	
علی	نازکی آبار	نازکی آبار	
رضا	قم		
پرتان	ون		

در صورتی که در جدول مرجع و صوری ایجاد می کنیم یعنی فرض می کنیم که تمام مشتریان در تمام شعب تهران حساب دارند و آنهایی که جواب نیستند از این مجموعه مرجع کم می کنیم
 فرض: تمام افراد در تمام شعب تهران حساب دارند

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

اسم مشتری پای که جواب نیستند

Assignment 4: این عملگر درست باشد " در زبان های برنامه نویسی مثل می اند
 نتیجه $\leftarrow \prod_{R-S}(r_1)$

جلسه هفتم 86.8.19

مثال (P. 57) نام تمام شعب با مشتریانی را که در Harrison زندگی می کنند و در این حساب دارند با هم بیابند.

از صورت مسئله می فهمیم که به branch-name و customer-city نیاز داریم که دو schema Account و Customer را به نامی دهد و depositor این دو schema را به هم مربوط می سازد پس داریم:

- Customer (customer-name, customer-street, *Customer-city)
- account (*account-number, branch-name, balance)
- depositor (customer-name, account-number)

از natural join این سه schema 6 نتون خواهیم داشت که تنها به بیستونش احتیاج داریم. ابتدا به صورت برطی انتخاب می کنیم برطی را که در Harrison و سپس نتون branch-name را پر اجتن می کنیم.

$$\prod (\sigma_{customer-city = "Harrison"} (customer \bowtie account \bowtie depositor))$$

branch-name

مثال (p-58) نام مسترهای را بیابید که هم وام گرفته اند و هم در بانک حساب دارند.

دو schema داریم:
 depositor ← مسترهای که در بانک حساب دارند.
 borrower ← مسترهای که وام گرفته اند.
 اگر از عملگر \bowtie استفاده نکنیم و \cap می توانیم پاسخ صحیح را بدست آوریم اما \bowtie کار آسان تر است.

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

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

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 \bowtie مناسب Queries است که در آنها "For all" آمده، البته این مورد در همه موارد صدق نمی کند.

مدل natural join:

توجه کنیدی خواهم برای exp زیر table رسم کنیم:

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

$$\begin{matrix} & & \Pi & & \Pi \\ & & \downarrow & & \downarrow \\ (A_1, 10, \text{میراندار}) & (A_1, \text{آرین}) & = & (A_1, 10, \text{میراندار}, \text{آرین}) \\ (A_2, 20, \text{ون}) & (A_2, \text{آرین}) & & (A_2, 20, \text{ون}, \text{آرین}) \\ (A_3, \emptyset, \text{آمالان}) & (A_3, \text{رضا}) & & (A_3, \emptyset, \text{آمالان}, \text{رضا}) \end{matrix}$$

توجه: σ مقایسه می کند اما از آنجایی که یک σ unary است باید دو رابطه در هم فیلد شوند. σ 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 جدید رابطی از چندین روش گسترش می یابند که از ساده ترین روش ها استفاده از عملگرهای محاسباتی است. دور روش مهم برای توسعه یک رابطه ای عبارتند از:

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

Generalized projection:

$$\Pi (E)_{F_1, F_2, \dots, F_n}$$

E : می تواند هر exp. دیتابیس باشد
 F_i : هر عبارت جبری می تواند باشد

مثال: می خواهیم ببینیم بر 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.

$$\Pi (Credit - info)$$

customer-name, Limit-credit-balance as available

«credit-info»

customer-name	Limit	Credit-balance	customer-name	available
امیرعلی	2000	1500	امیرعلی	500
سینا	1000	500	سینا	500
علی	1500	800	علی	700

Limit \leftarrow کل مقداری که هر شخص می تواند خرج کند
 Credit-balance \leftarrow کل پولی که هر شخص در دستش دارد
 $Limit > Credit-balance$

میانگین در مثال می بینیم در II از عبارت جبری تفاضل استفاده کردیم و حاصل این تفاضل را با کلمه کلیدی **as** نامگذاری کردیم و پاسخ II دستور به نام های **customer-name** و **available** دارد.

تفاوت **set** ، **multiset** :

در **set**، تکرار معنادار یعنی در مجموعه تکرار مهم نیست اما در **multiset** تکرار مهم است :

set → { 1, 2 } = { 1, 2 }

Multiset → { 1, 1, 2, 2 } = { 1, 1, 2, 2 }

Aggregate Functions : $G(E)$ $F(A_1), \dots, F(A_n)$

aggregate Fun. مجموعه از مقادیر را می گیرد و یک مقدار را بر می گرداند :

aggregate Fun. در **multiset** کار می کند یعنی تکرار اهمیت دارد.

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

24 ← **sum** : aggregate function G_{sum} مجموع مقادیر را بر می گرداند

4 ← **avg** : aggregate function G_{avg} میانگین مقادیر را بر می گرداند

6 ← **count** : aggregate function G_{count} تعداد اعضای مجموعه را می دهد

1 ← **min** : aggregate function G_{min} کوچکترین عضو مجموعه را می دهد

11 ← **max** : aggregate function G_{max} بزرگترین عضو مجموعه را می دهد

4 ← **distinct** : aggregate function $G_{distinct}$ از هر تکراری که شمارد (تعداد متمایزها)

مثال : مجموع حقوق که شرکت به کارمندان می دهد

sum(salary) **as** sum-salary

در جدول (schema) employee

مجموع salary ها را بر می گرداند

ستون پاسخ را sum-salary می گویند و پاسخ تطبیق دهنده ستون دارد.

"Employee"

Employee-name	salary	branch-name
یوسف	1,000	میراباد
طاهره	5,000	زنک
پژمان	12,000	زنک
محمد	6,000	میراباد

sum-salary
24,000

مثال: بیشترین مقدار حقوق را مشخص کنید

G (account)
 max(balance), account #

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

G (employee)
 branch-name sum(salary) as sum-salary

بر حسب نام شعبه گروه بندی می کند

Employee-name	salary	branch-name
یوسف	1,000	میراباد
محمد	6,000	میراباد
طاهره	5,000	زنک
پژمان	12,000	زنک

branch-name	sum-salary
میراباد	7,000
زنک	17,000

مثال (p. 64) مجموع حقوق در هر شعبه و MAX حقوق در هر شعبه؟

G (employee)
 branch-name sum(salary) as A, max(salary) as B

branch-name	A	B
میراباد	7000	6000
زنک	17000	12000

پس به طور کلی داریم:

$(G_1, G_2, \dots, G_n) \rightarrow G(E) \rightarrow F_1(A_1), \dots, F_n(A_n)$

- G_i : Att. هایی که در سمت G_i می آیند باعث گروه بندی می شوند یعنی G ابتدا بر طبق Att. G_i و در هر یک را گروه بندی کرده و پس عملیات لازم را انجام می دهد
- F_i : عملیاتی است که باید انجام شود مثل sum و avg و ...
- A_i : نام Att. هایی 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 برشعبه شهر را بدهد

employee

City	branch-name	employ-name	salary
تهران	ونف	علیر	1000
تهران	ونف	رضا	5,000
اصفهان	بختی	زهرا	11,000
اصفهان	بختی	یونس	1,000
تهران	بختی	محمد	3,000
تهران	بختی	سینا	12,000

باید گروه بندی انجام شود یعنی:

$G_1 = \text{branch-name}$

$G_2 = \text{city}$

و دستت G داریم و

$F_1 = \text{sum}$

$A_1 = \text{salary}$

city, branch-name G (employee) \rightarrow sum(salary) as sum-salary

G_1 G_2 F_1 A_1

City	branch-name	sum-salary
تهران	ونف	6,000
تهران	بختی	15,000
اصفهان	بختی	12,000

یا می توانستیم داشت

که در دسترس ما

G_1 و G_2 تکیه می دهیم

و دستت نام $F_1(A_1)$