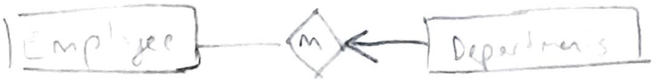
each dept. has at **most one** manager

week 1.5



manages ilişkisinde 1 dept'in adı sadece 1 kere geçebilir.

Manages

1) Manages

ssn | did | - -

FK ssn

FK did

PK did

Çünkü ~~1~~ 1 dept en fazla 1 kere bulunur

OR

2) Departments

did | - ssn

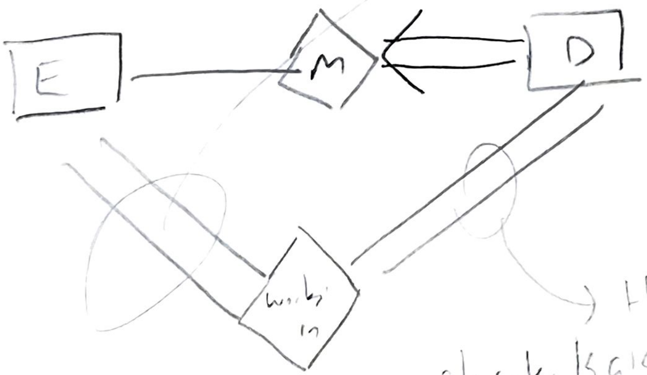
PK did

F.K ssn

Total participation = at least one manager

→ at most one manager

Every dept has a manager



Works_in → her employee bu ilişkide en az 1 kere olacak.

1 emp. 1'den fazla departmanda çalışabilir.

Her dept bu ilişkide en az 1 kere olacak. Yoksa dept olmaz!

gation

sum, count, min, max, avg
↳ except count, all agg apply to single attribute

- Find total quantity for all products over 1$ by product.

Product	price	Quan

```
Select product, sum(quantity)
from purchase where price > 1
group by product
```

Bagel	3	20
Bagel	1.5	20
~~Banana~~	~~0.5~~	~~50~~
"	2	10
"	4	10

→

Bagel	40
Banana	20

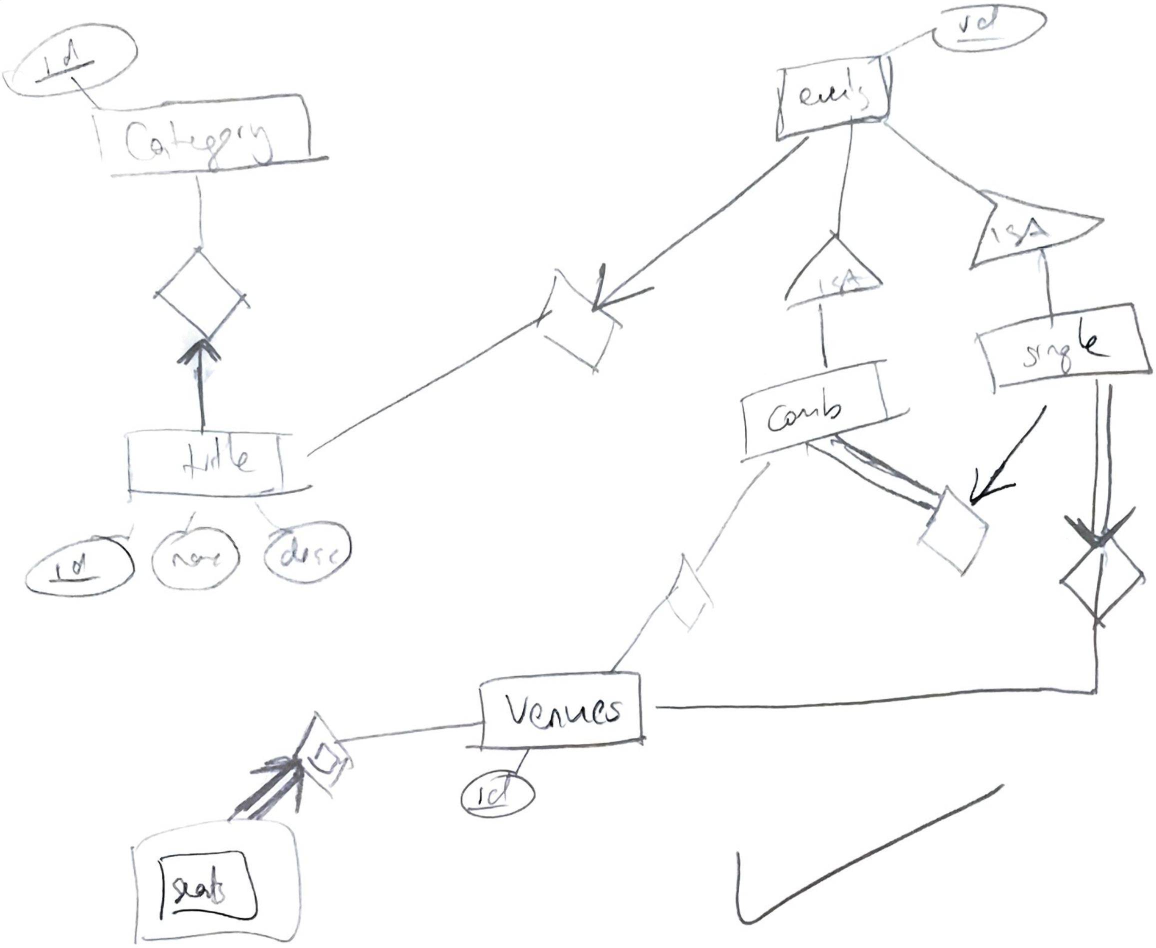
Having = Conditions on aggregates.

Having Sum(quantity) > 30

General Form of Grouping and Aggregation

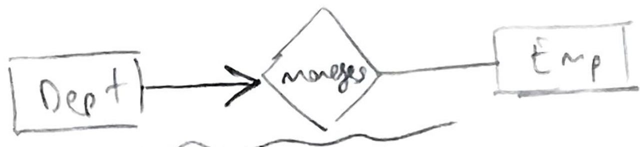
```
Select S              S → may contain a1 -- ak but no other attr.
From R1, -- Rn        C1 → cond on R1 -- Rn
where C1              C2 → cond on a1 -- ak
Groupby a1 --- ak
having C2
```

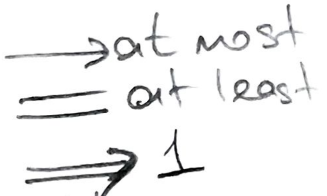
evaluation = From, where apply C1
Group by a1 -- ak
Apply C2 to each group
Compute aggregates, result


id
Category
events
id
ISA
ISA
title
comb
single
id
name
desc
Venues
id
seats

week 2

each dept has at most one manager





⇓

A dept can only be one (unique) in the manages relation

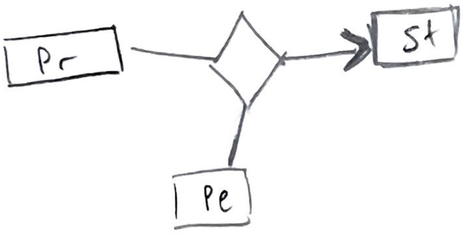
We can do this in the SQL:

1) Manages table
ssn | did | ...
Foreign key ssn
// did
Primary key did

OR

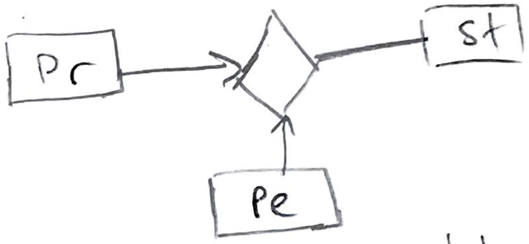
2) Dept table
did | ssn | ...
did PK
ssn F.K

Slide notation

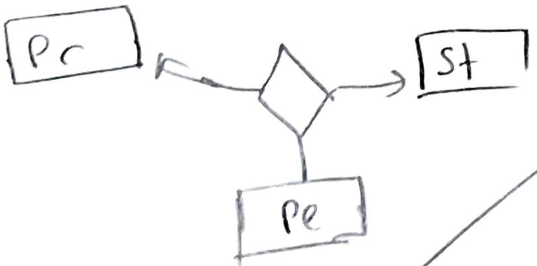


Read as: Given person and product it must be bought/purchased from at most 1 store.
If I buy kitkat from Sok, I cannot buy kitkat from Migros

Textbook notation



Read as: In the purchase relation, for given product and person, there is at most one store.



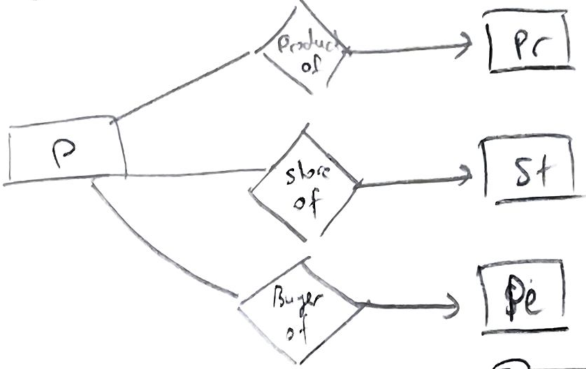
Read as: A given person buys a given product from at most one store and a given person buys at most one product from a given store.

OR
every store sells to every person at most one product

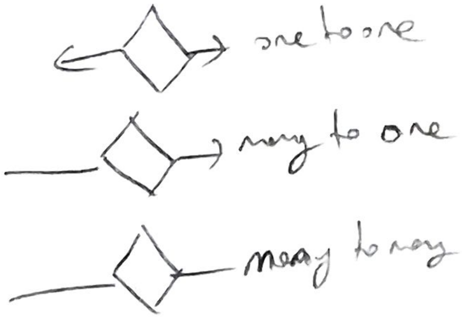
p1	pr1	s1	
p1	pr2	s1	✗
p1	pr1	s2	✗
p1	pr2	s2	✓
p2	pr1	s1	✓

onversion of Multi-way to Binary

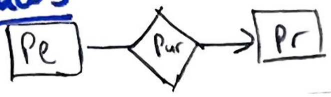
- Every person shops at most one store!



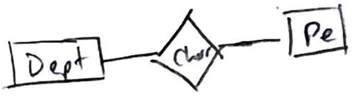
Slide notation



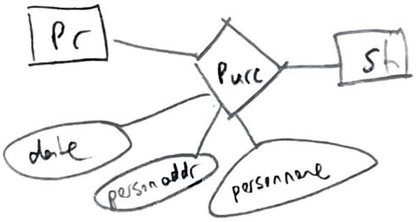
Flaws



A given person can buy at most one product.

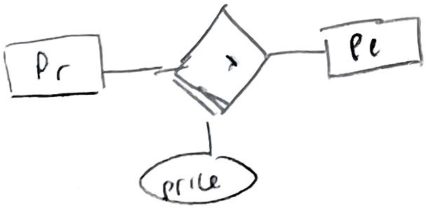


Some dept can have no men in their chair.
Some people can be chair of multiple dept.

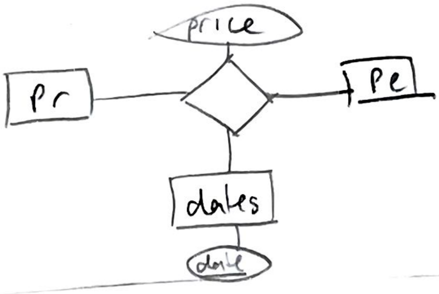


Pr1	st1	d1	p1addr	p1name
Pr2	st1	d1	//	//
pr3	//	//	//	//

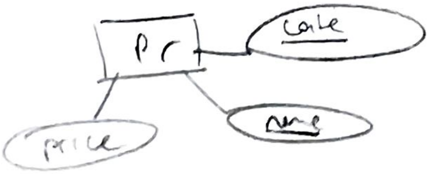
→ unnecessary fields



If I want to add date

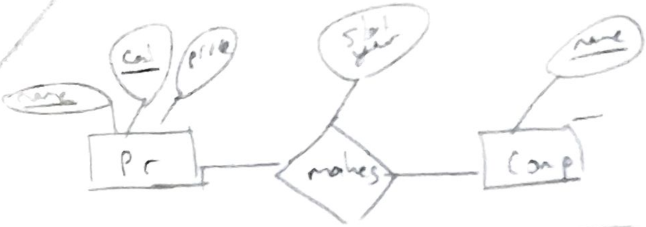


Entity Set Relation



Product (name, cat, pr)

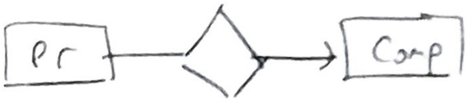
M relations



Makes (pname, pcat, cname, startyear)

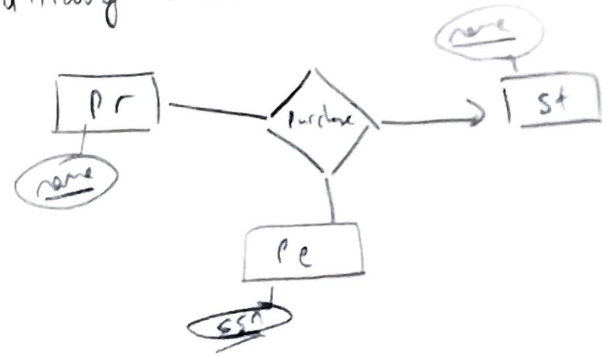
M-1 relation

A given product can only made by at most 1 company. Therefore, we don't need makes table.



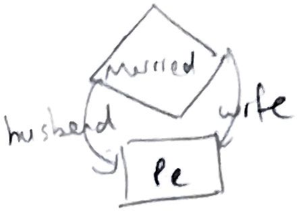
Product (name, cat, price, startyear, company)

Multiway relation

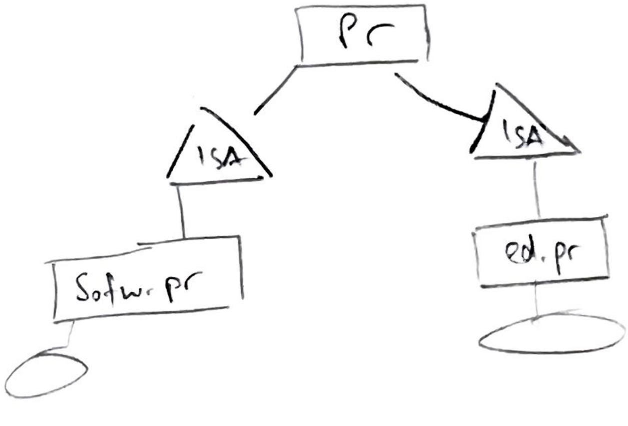


Purchase (pname, pename, stname)

Roles



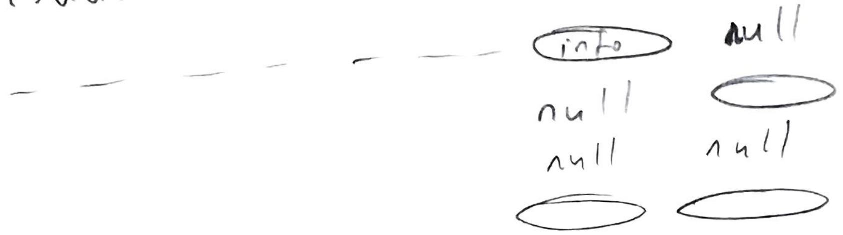
Subclasses



the subclasses ~~we~~ can have these:

1)
1- product (name, price, cat)
2- suproduct (name, platforms)
3- edproduct (name, age)

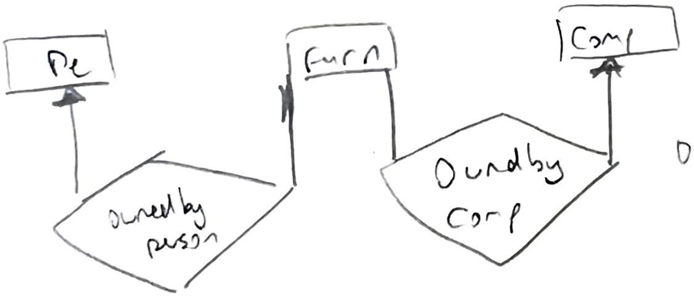
2) Product (name, price, cat, platforms, age)



3)
Suproduct (name, price, cat, platform)
Edproduct (name, price, cat, age)
→ Too many repeating

Each piece of furniture is owned either by a person or by a company

1st attempt

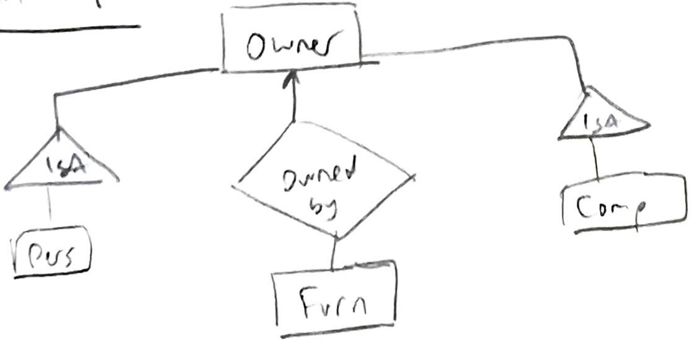


In this way, we can have this:

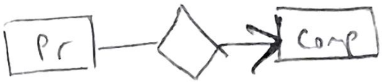
own by comp: f1, c1
f2, c1
f3, c2

own by pe: f1, p1
f4, p2

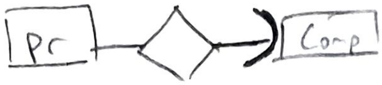
2nd attempt



ides

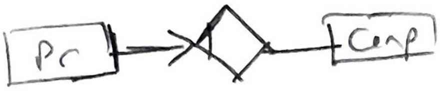


1) Each product made by at most 1 comp.

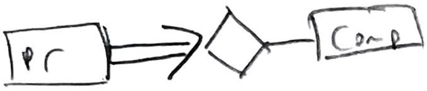


2) Each product made by exactly 1 comp.

Textbook

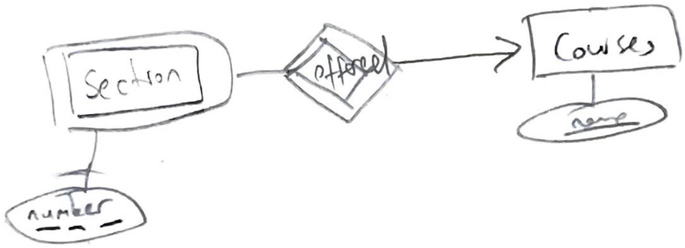


1)



2)

Weak Entity Sets

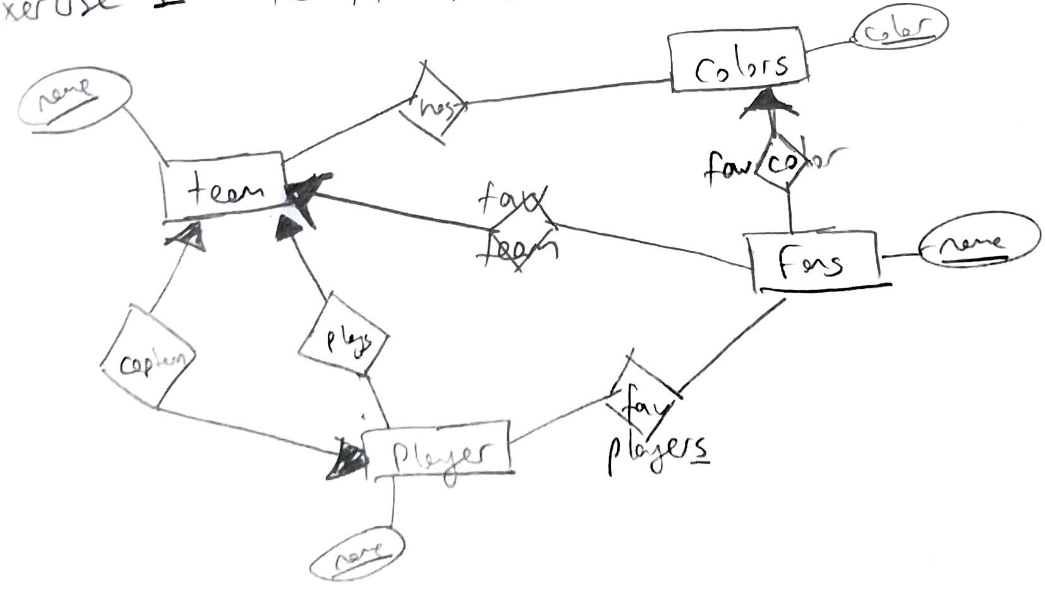


1) Courses(name, ---)

2) Section (number, name, ---)

no offered table

Exercise 1 – team, fan, player



y Constraints

```
Create table Product
    productID char(10),
    PK pId
    unique (name, cat))
```

```
C. T. Purchase
    pname - -
    references Product (name)
        or
    references Product → if name is PK
                            in Product table
    - - -
```

```
C. T. Product
    id, name, cat, price
    PK (name, cat)
    unique (id)
```

```
C.T. Purchase
id references Product (id) → Must
                                since id is
                                not PK in Product

id references Product does not work !
```

Policies for ref. integrity

No action → reject violating modf.

cascade → after delete / update do delete/update

set null → set foreign key to null

set default → // to default val

```
C.T. Purchase
  pname, cat, date
  FK (pname, cat) ref. Product (pname, cat)
  on update cascade → when product table entry is updated
                        purchase table is also updated.
  on delete set null
                     ↳ when the product is deleted
```

Purchase

null	null	date

It is possible because we did not implement P.K. in Purchase table

If we have
  + PK (prodname) or PK (prodname, date) → on delete set null will not work

straint on Attr. and Tuples

C.T R
A int <u>not null</u>
B int check ((B>50 and B<100)) → Cons. on attribute

Check (C>='d' or D>0) → cons. on tuple

<u>Check</u> is checking every <u>insert</u> and <u>update</u>.

<u>F.K</u> is checking every <u>insert</u>, <u>update</u> and delete.

C.T. Purchase
prodname -- Check (prodname in
select Product.name from
Product)
date not null

→ When 'product is deleted, the purchase table will <u>not check</u> for the cond.

C.T. Product (
name, cat, price
Check (name = 'gizmo' or price <= 5.00))

⇒ only gizmos can have more than 5.
⇒ other products must have less than 5.