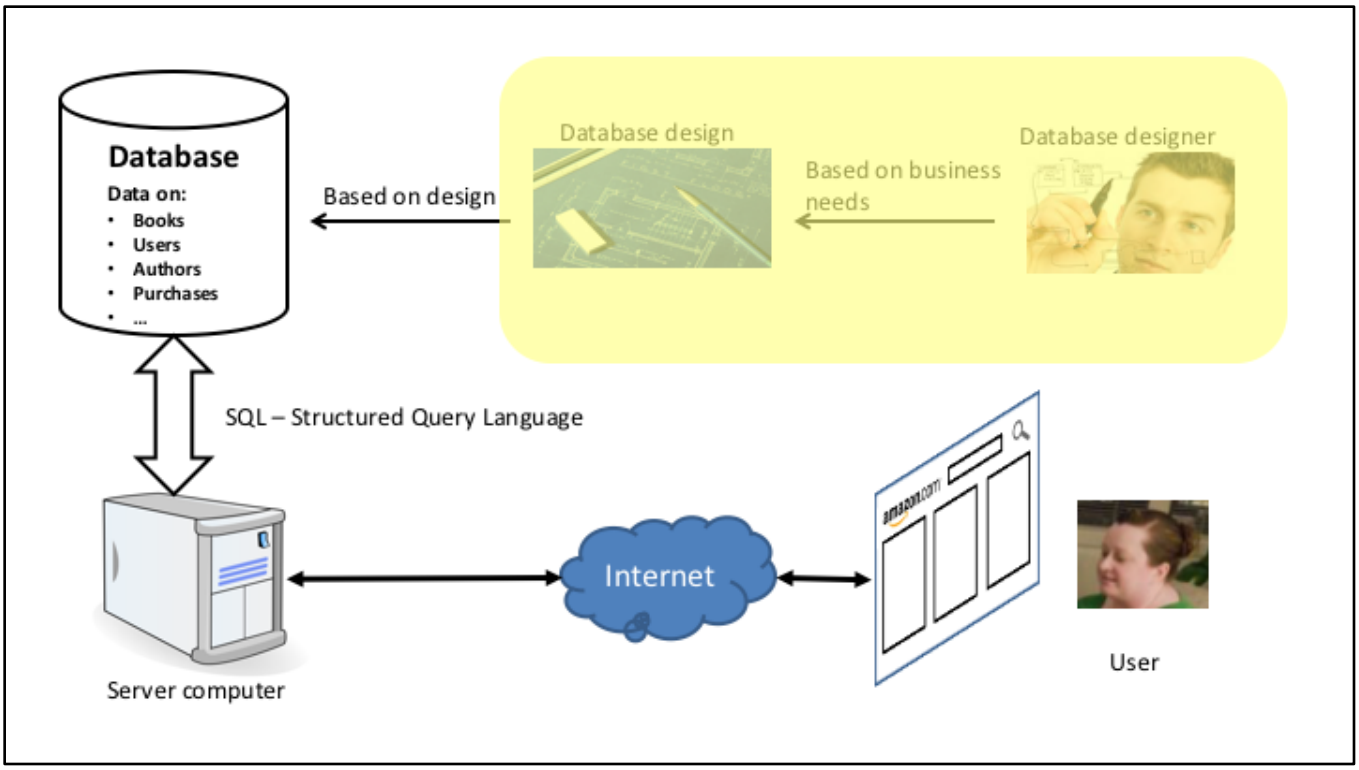
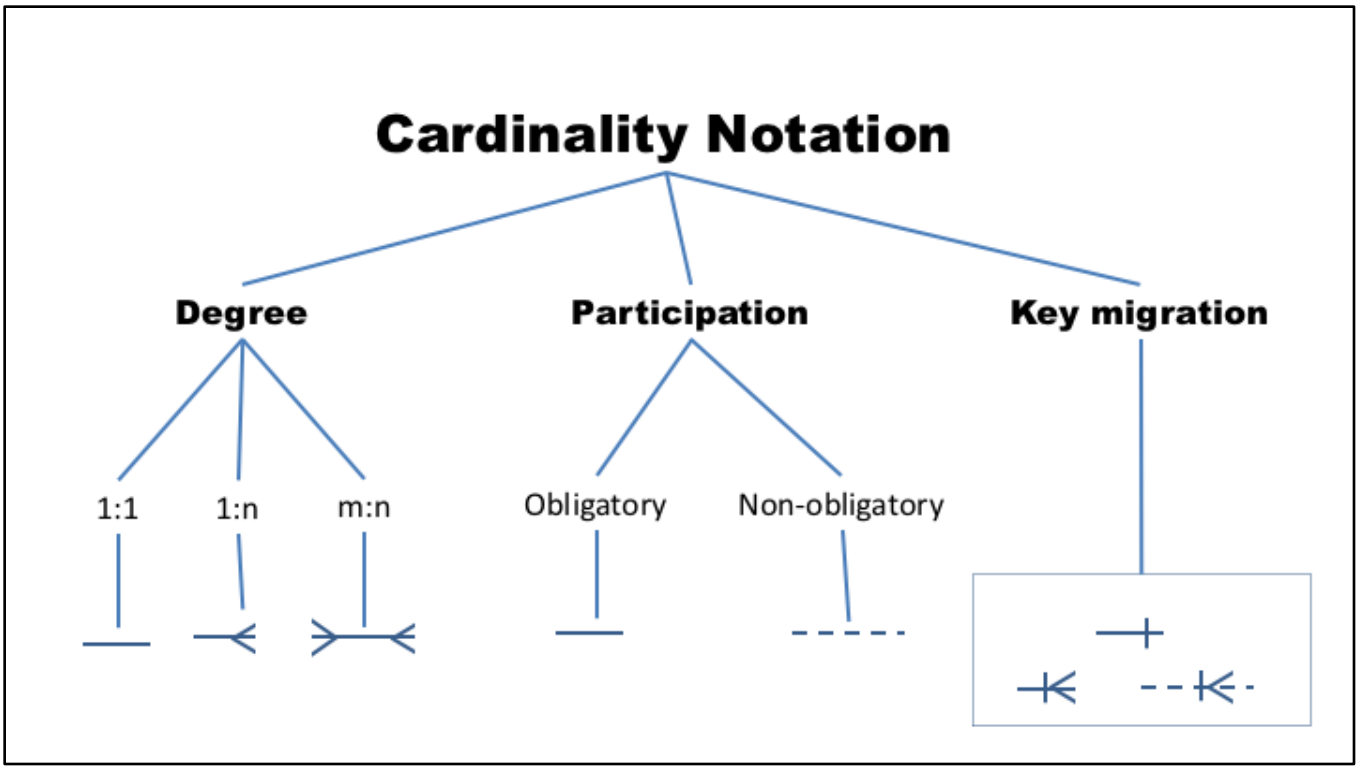
Database Design: Part 2

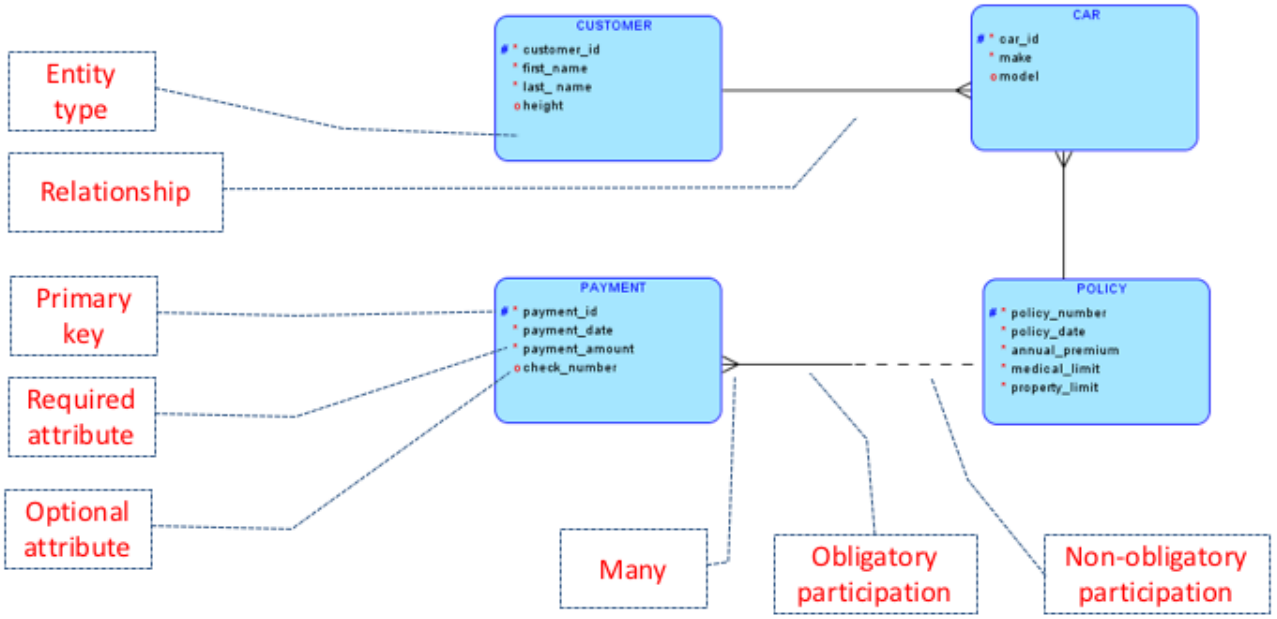


We have already looked at how SQL can help us to retrieve almost any information we need from a relational database.

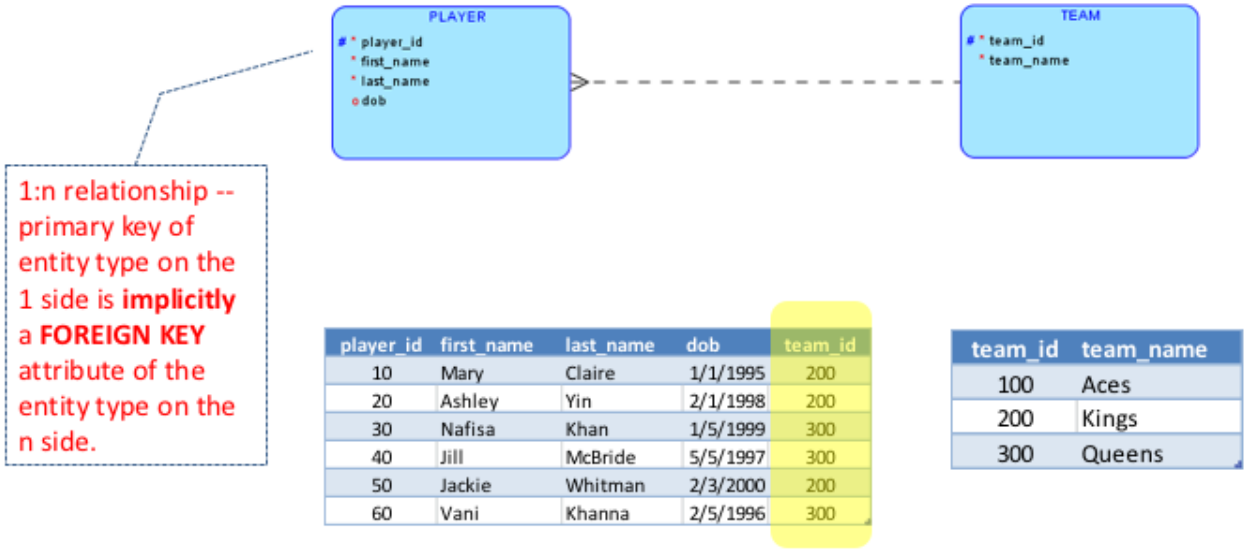
We have started looking at how we can arrive at a database design in the form of an Entity Relationship Diagram for a given business situation.


Cardinality Notation
Degree
Participation
Key migration
1:1
1:n
m:n
Obligatory
Non-obligatory

Recap 1 -- Basics



Recap 2 -- Implicit Foreign Key in 1:n Relationship



player_id	first_name	last_name	dob	team_id
10	Mary	Claire	1/1/1995	200
20	Ashley	Yin	2/1/1998	200
30	Nafisa	Khan	1/5/1999	300
40	Jill	McBride	5/5/1997	300
50	Jackie	Whitman	2/3/2000	200
60	Vani	Khanna	2/5/1996	300

team_id	team_name
100	Aces
200	Kings
300	Queens

Whenever we have a 1:n relationship, we can see that representing the relationship in tables requires adding the primary key of the entity type on the 1 side as a foreign key to the entity type on the n side.

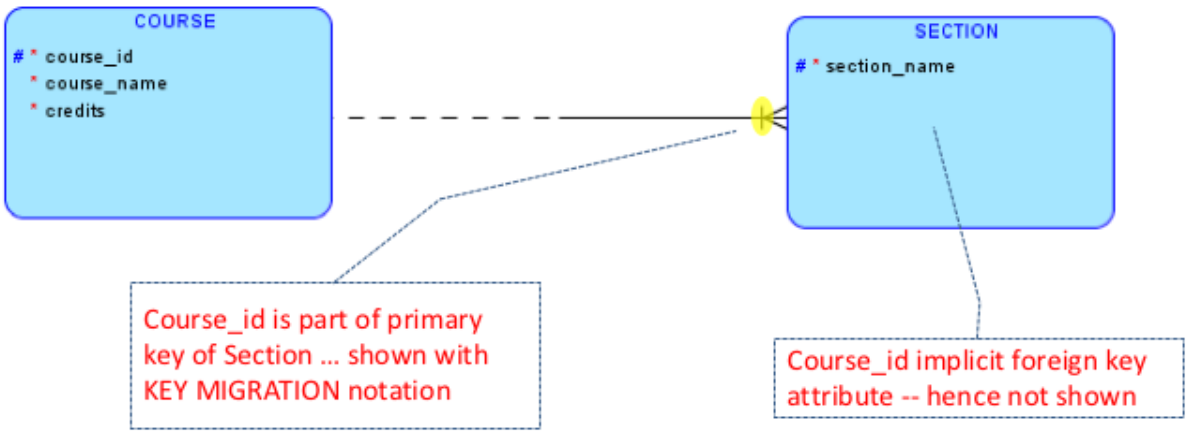
You can easily convince yourself that adding the foreign key to the entity type on the 1 side will not work.

For example, in the above case, adding player_id as a foreign key to the teams table will not work, because a team could have many players and adding a foreign key will allow us to only show one player per team.

However, adding the team_id to the players table works because each player can belong to at most one team and adding a foreign key column allows this.

ODM Demo

Recap 3 – Key Migration



Don't confuse the implicit foreign key attribute with the key migration notation.

The implicit foreign key attribute always arises in all 1:n relationships.

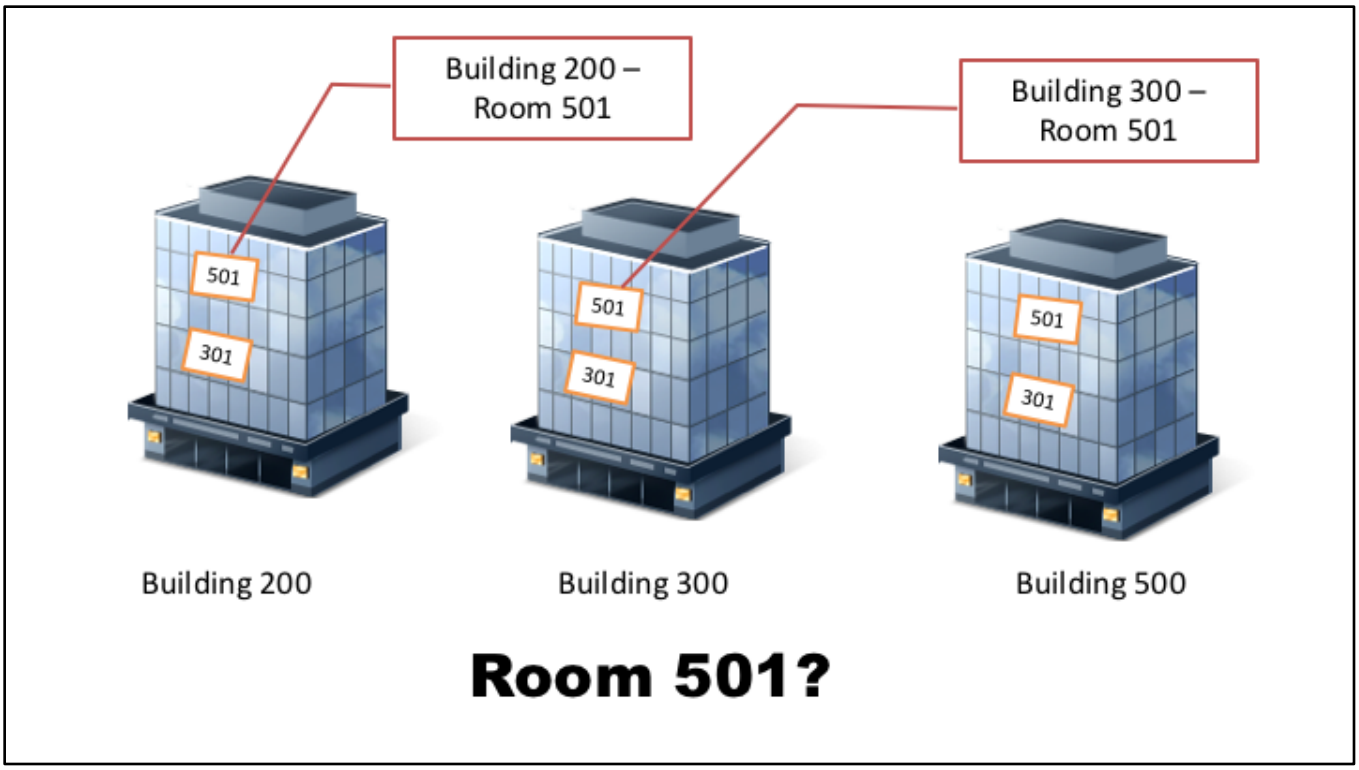
We use the key migration notation only when the foreign key also happens to be part of the primary key in the table on the n side of the relationship. Be sure to understand this from the many examples provided earlier.

A company has many buildings and each has many rooms. To track the assets installed in each room, the company needs to track all rooms. The company has assigned each building a unique building_id and stores the square footage, GPS coordinate and height of each building.

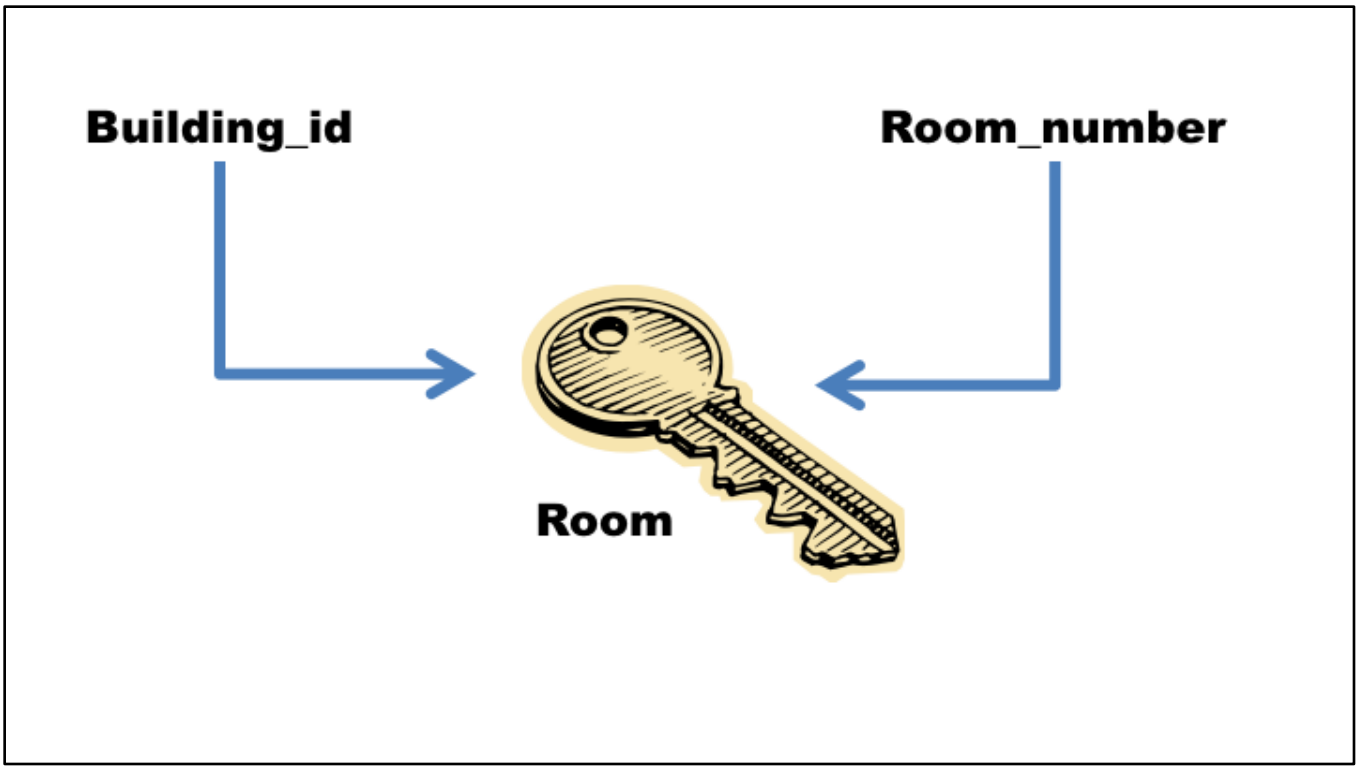
Each building has many rooms and each room has a room number. Room numbers are unique within a building, but not across buildings. Thus building 200 could have a room 103 and building 300 could also have a room 103.

To uniquely identify a room across the company we have to use both the building id and the room number.

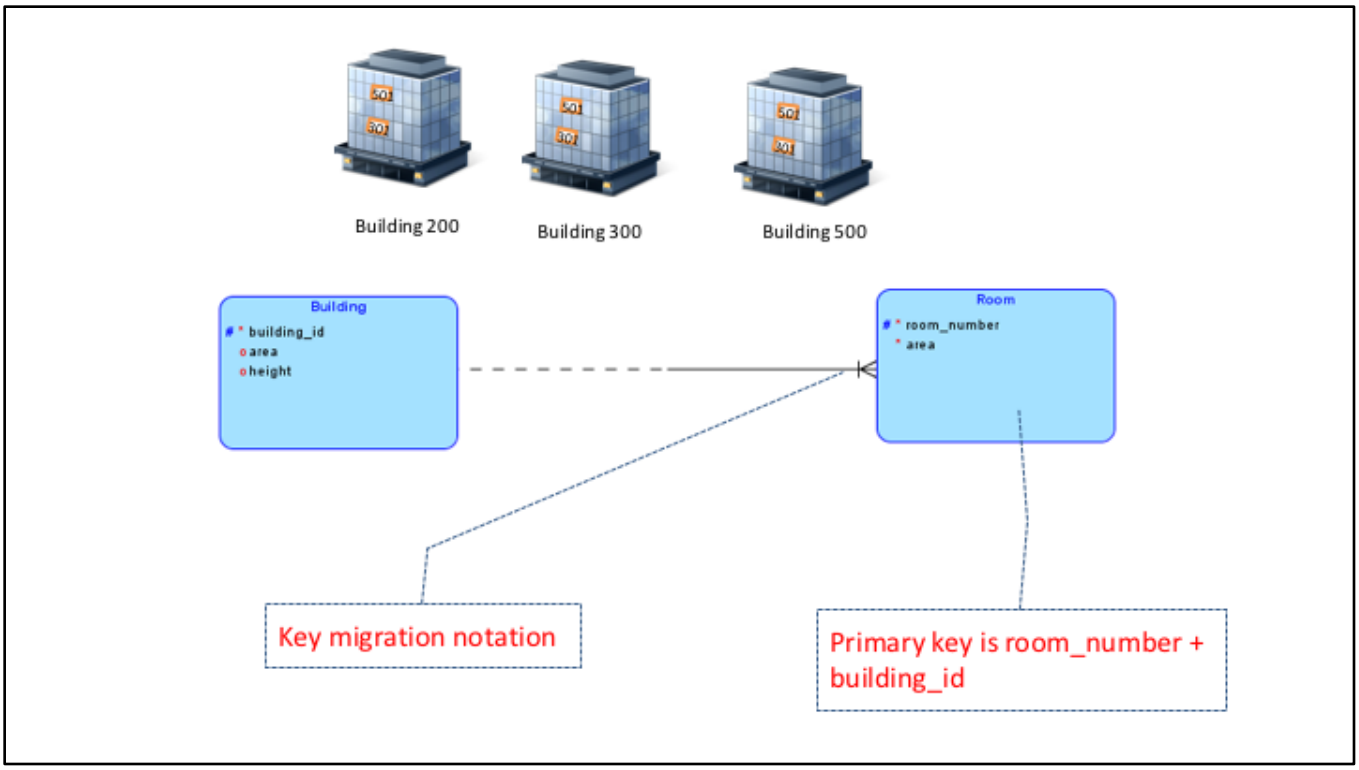
A company wants to track the assets installed in various rooms of its office buildings. Hence it needs to have a way to store information about buildings and rooms.



The same room number can occur in several buildings and hence the room number alone does not suffice to uniquely identify a room across all the buildings.

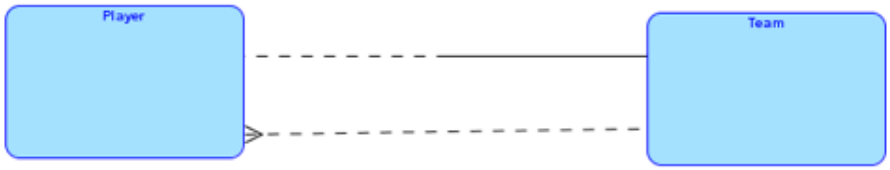


The combination of room number and building number provides a unique identifier for a room. The combination will be unique across all roooms of all buildings.


Building 200
Building 300
Building 500
Building
* building_id
o area
o height
Room
* room_number
* area
Key migration notation
Primary key is room_number + building_id

Key Migration Demo in ODM

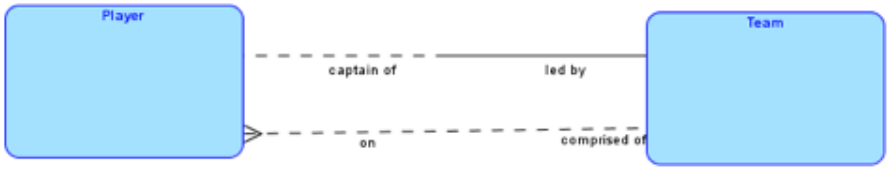
Relationship Naming



Two relationships? What do they stand for?

In situations where we know the application domain very well, we might not need to be told what each relationship represents. However, in unfamiliar domains and sometimes even in familiar domains, we might not understand what a relationship signifies. In the above ERD, we might ve a little confused by the two relationships between Player and Team.

To clarify the meaning of relationships, we need to name them.



Now we see why we have two relationships

Need names to clarify ERDs

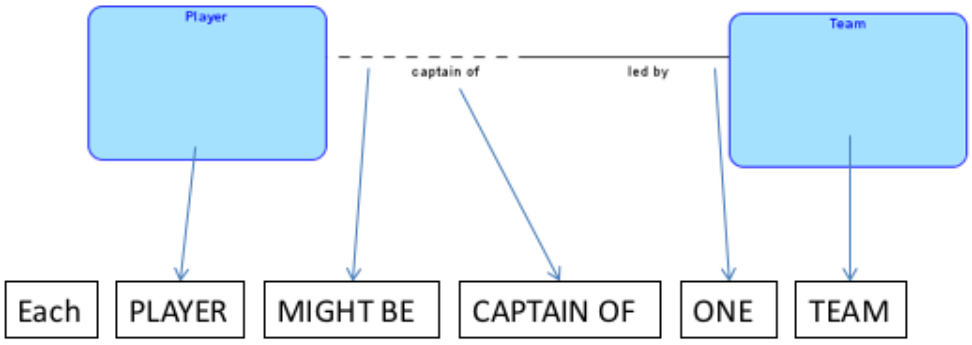
The names on the relationships tell us what each relationship signifies from the viewpoint of the application domain.

One shows that a player is a regular player on a team and the other tells us that each team has one player as captain.

We place names on relationship lines to indicate the role that each entity type plays in the relationship.

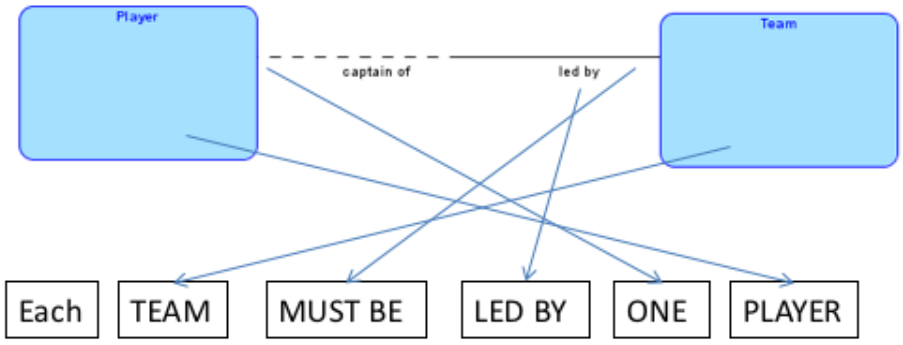
Reading Relationships

Reading from left …

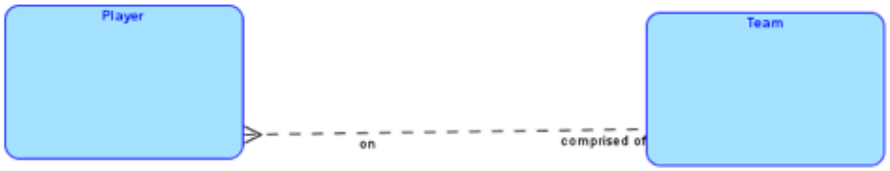


Reading Relationships

Reading from right …



Reading Relationships



Reading from left …

Each PLAYER MIGHT BE ON ONE TEAM

Reading from right …

Each TEAM MIGHT BE COMPRISED OF ONE OR MORE PLAYERS

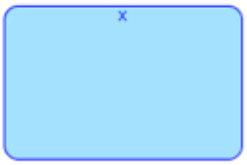
Naming Relationships

Reading from left to right – name the relationship on the left side so as to properly complete the following sentence:

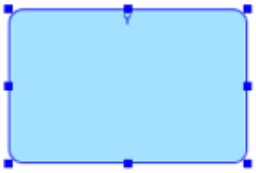
Each X | must be / might be | Name-1 | one or many / one | Y(s)

Solid or dashed on left

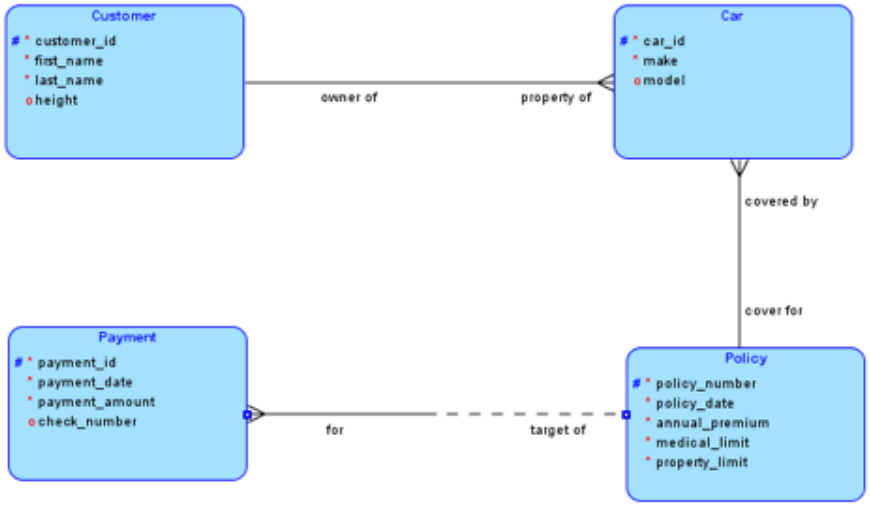
Crow-foot or not on right



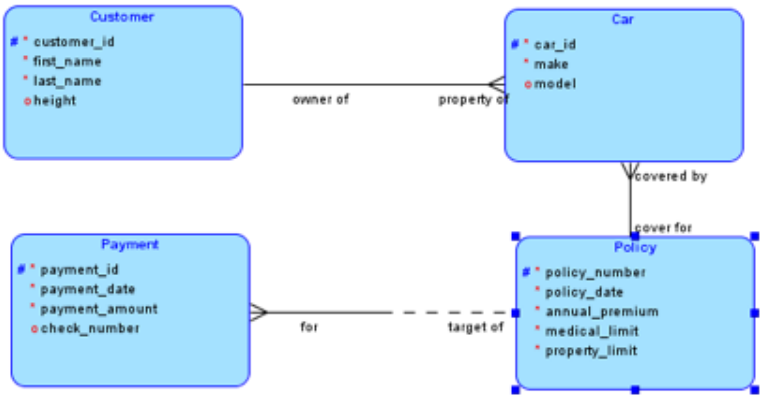
Name-1



Reading Relationships – Your turn



Reading Relationships – Your turn



Each customer must be owner of one or many cars

Each car must be property of one customer

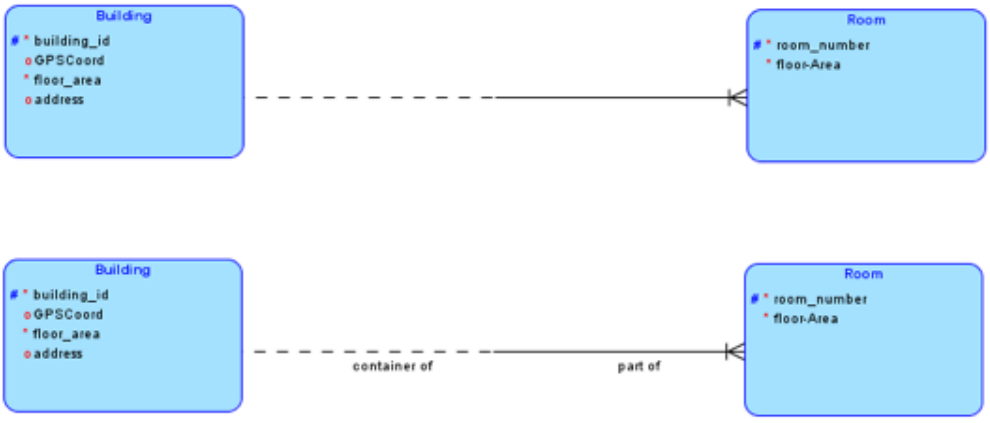
Each car must be covered by one policy

Each policy must be the cover for one or more cars

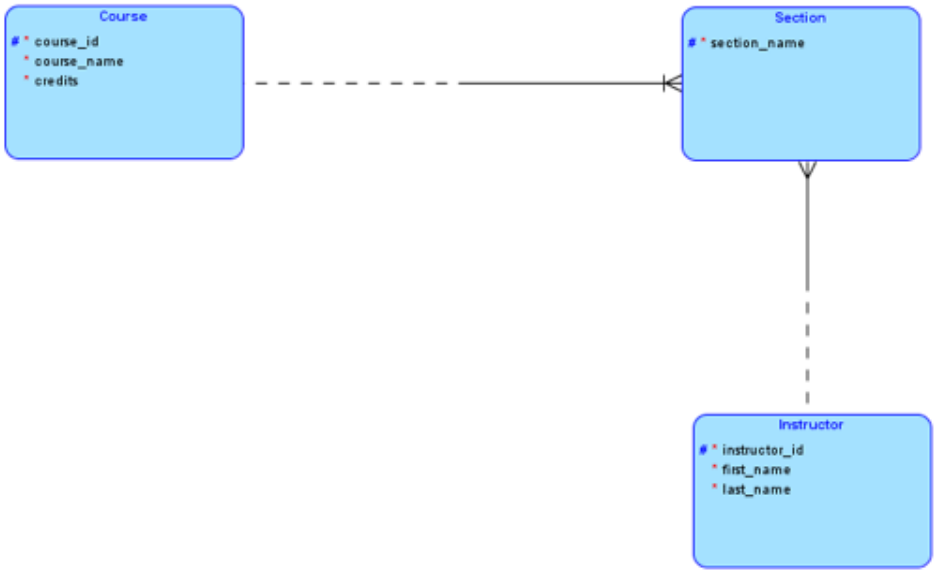
Each policy might be the target of one or more payments

Each payment must be for one policy

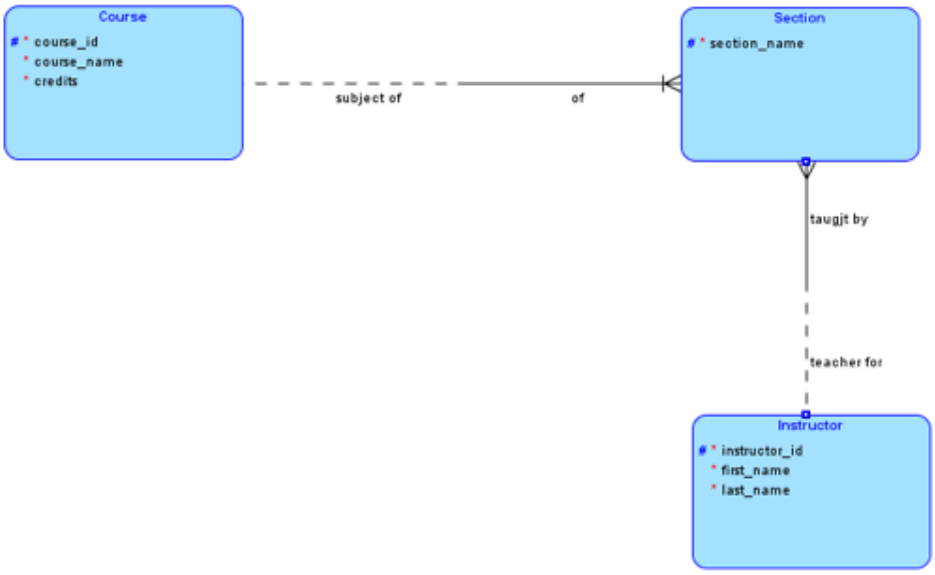
Naming Relationships – Your Turn: Find suitable names for the relationship – you need two names, one on each side.



Naming Relationships – Your Turn (continued): Find suitable names for the relationships – you need two names per relationship.



Naming Relationships – Your Turn (continued): Find suitable names for the relationships – you need two names per relationship.



Naming Relationships – Your Turn (continued): Find suitable names for the relationships – you need two names per relationship.

Relationship Naming Demo in ODM

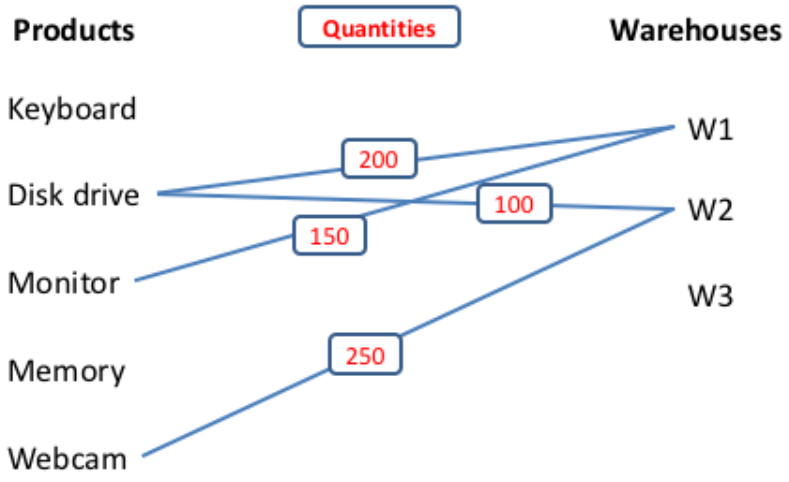
ERD Ninjahood

Master m:n Relationships

Inventory management scenario:

- A company keeps many products in stock.
- The company has many warehouses and stores many products in each warehouse.
- Each product could also be stored in several warehouses.
- At various points in time the company could run out of stock of some products.
- At various points in time, some warehouses could be empty.

Inventory management scenario

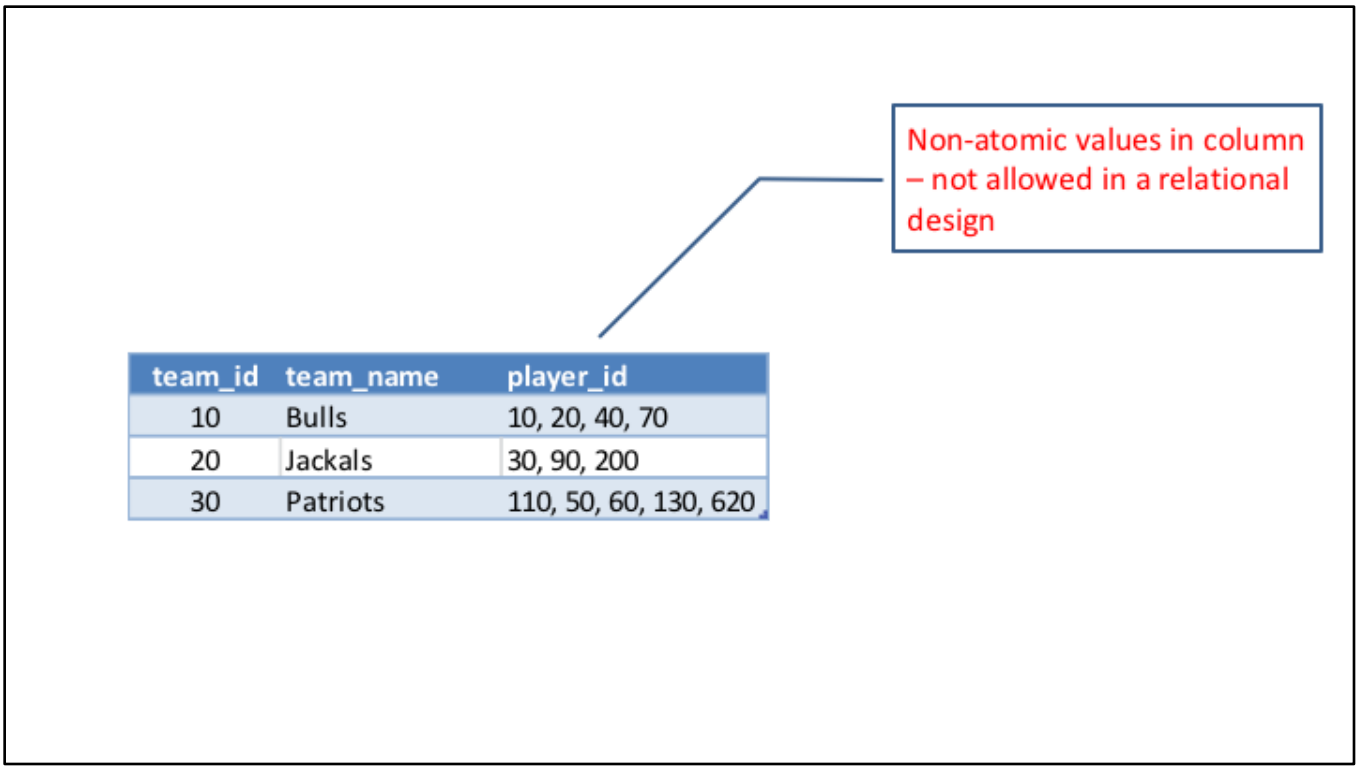


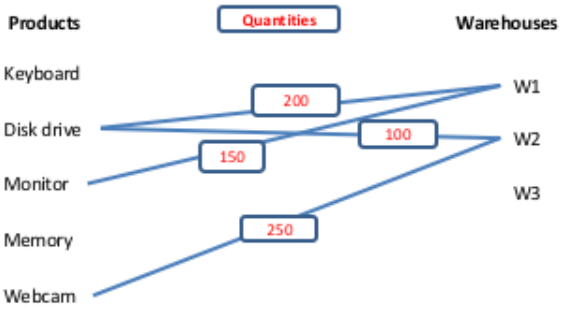
Attribute quantity?

Attribute values

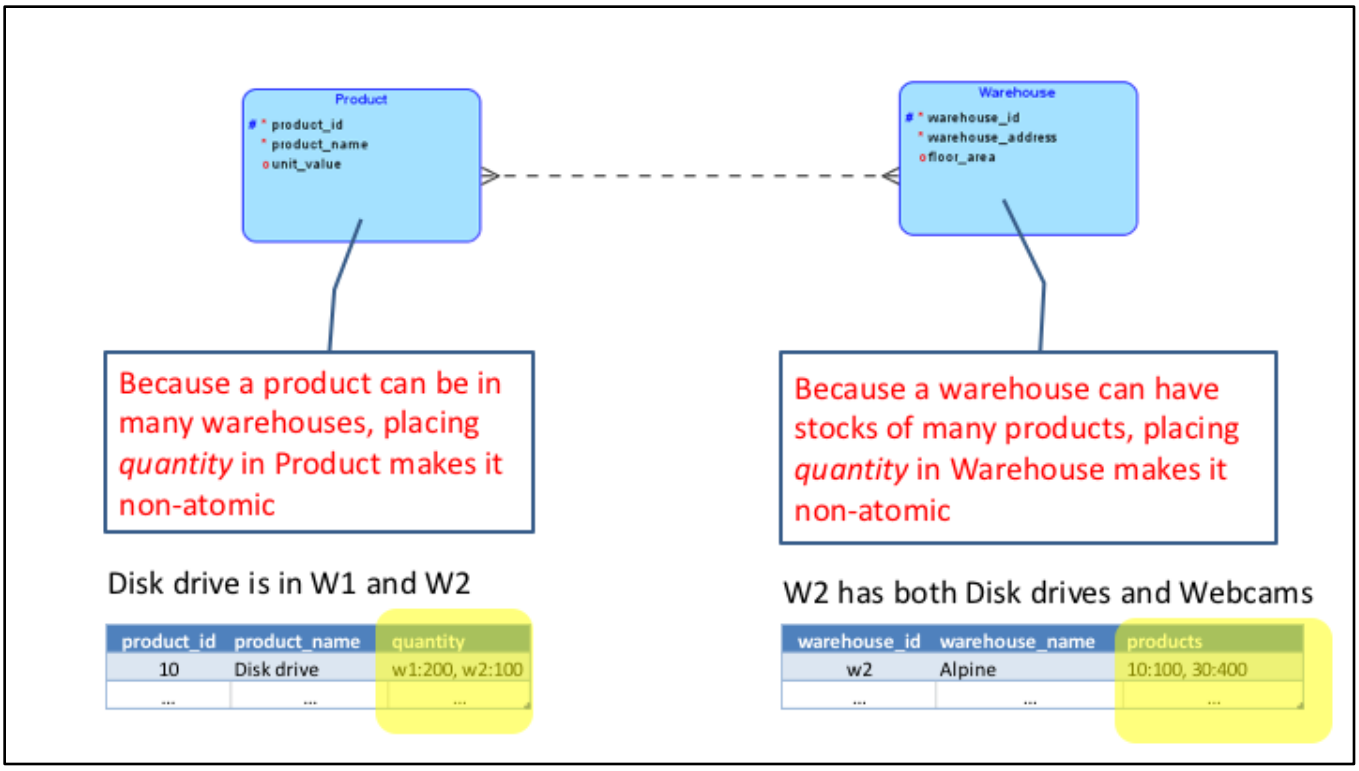
Atomic

Entity instances must have **indivisible** values for each attribute


Non-atomic values in column – not allowed in a relational design
team_id
team_name
player_id
10
Bulls
10, 20, 40, 70
20
Jackals
30, 90, 200
30
Patriots
110, 50, 60, 130, 620



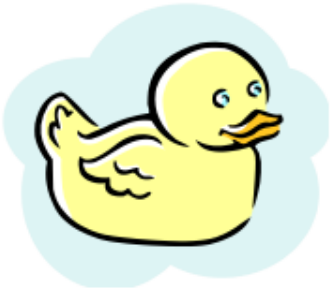
Where does attribute quantity belong? Product or Warehouse?


Product
* product_id
* product_name
o unit_value
Warehouse
* warehouse_id
* warehouse_address
o floor_area
Because a product can be in many warehouses, placing *quantity* in Product makes it non-atomic
Because a warehouse can have stocks of many products, placing *quantity* in Warehouse makes it non-atomic
Disk drive is in W1 and W2
product_id | product_name | quantity
10 | Disk drive | w1:200, w2:100
... | ... | ...
W2 has both Disk drives and Webcams
warehouse_id | warehouse_name | products
w2 | Alpine | 10:100, 30:400
... | ... | ...

Where does attribute quantity belong? Product or Warehouse?


Products
Quantities
Warehouses
Keyboard
Disk drive
Monitor
Memory
Webcam
200
100
150
250
W1
W2
W3
Quantity is an attribute of the relationship between Product and Warehouse

Duck test



If it swims like a duck and quacks like a duck it must be a duck

If a relationship has its own attribute, then it must be an entity type

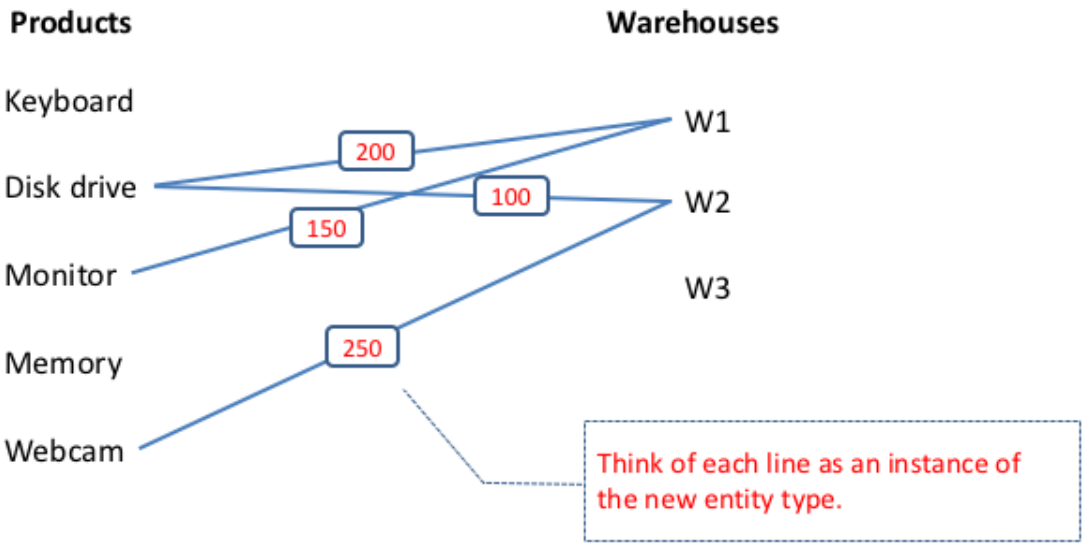
New entity type that has ***quantity*** as an attribute

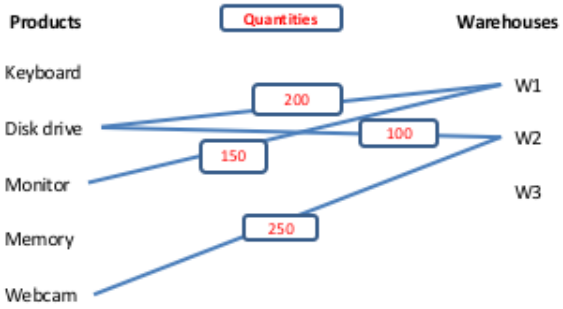
ALWAYS create new entity type for m:n relationship

ALWAYS create new entity type for m:n relationship …

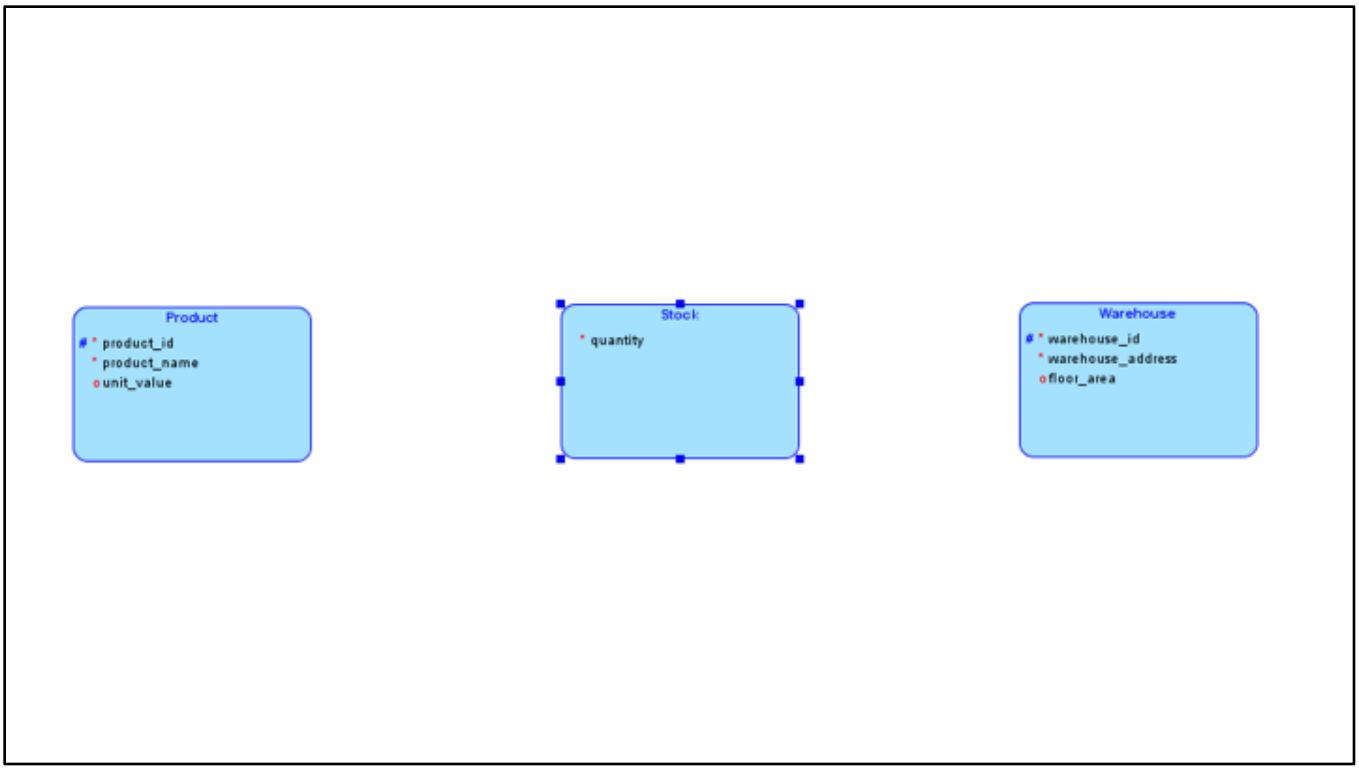
Even if it does not seem to have an attribute.

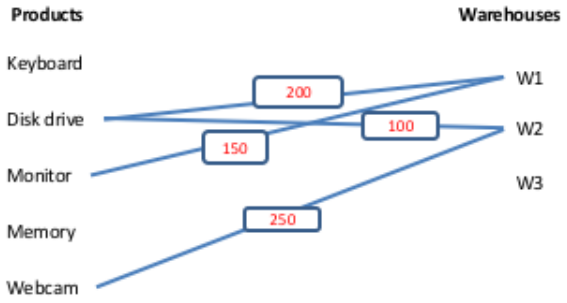
Inventory management scenario





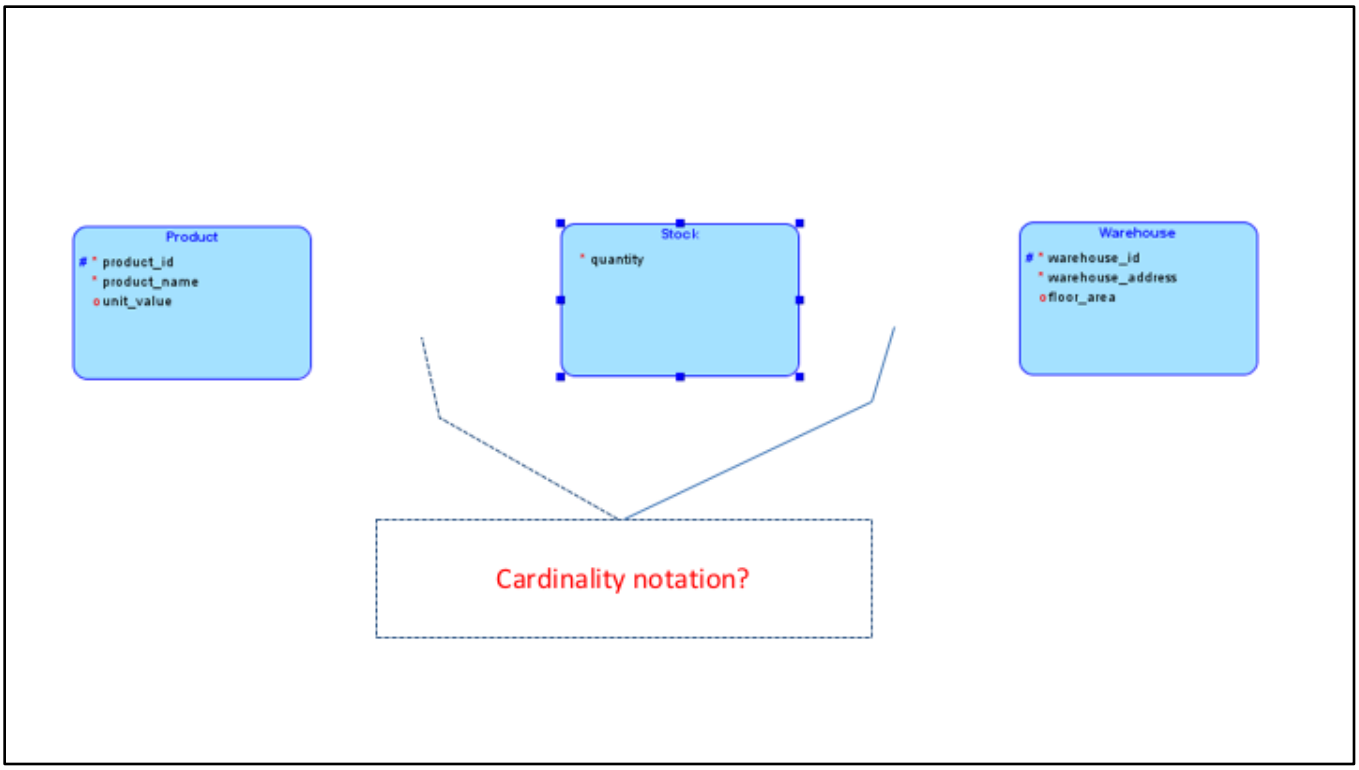
Each instance represents the storage of a certain quantity of a product at a warehouse – ***Stock***


Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area

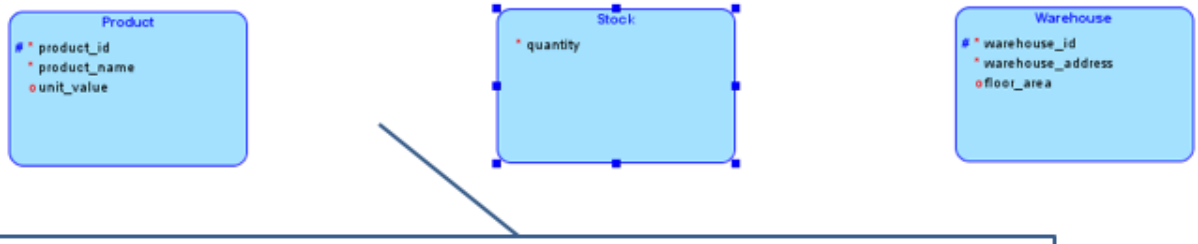


Stock instance: of a particular product at a particular warehouse

product_id + warehouse_id can be primary key for Stock


Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area
Cardinality notation?

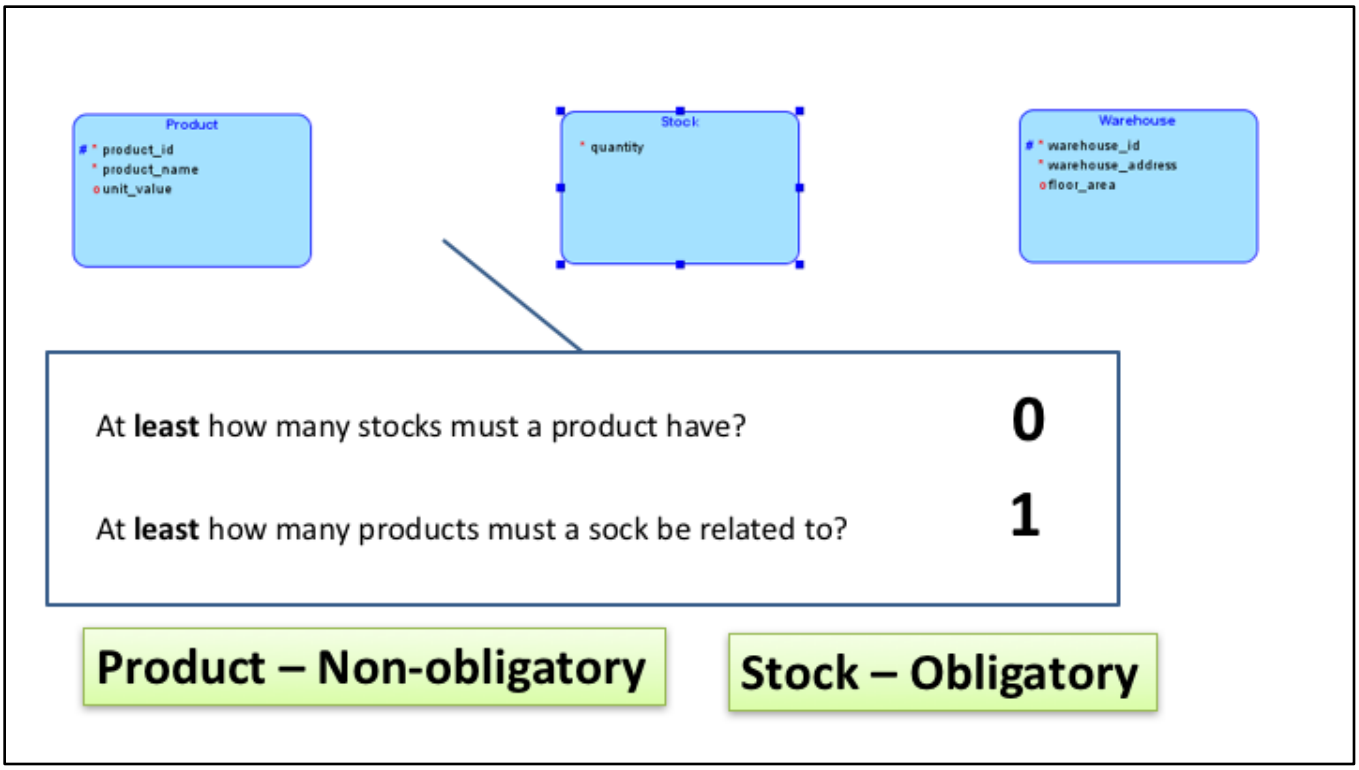
Relationship Degree

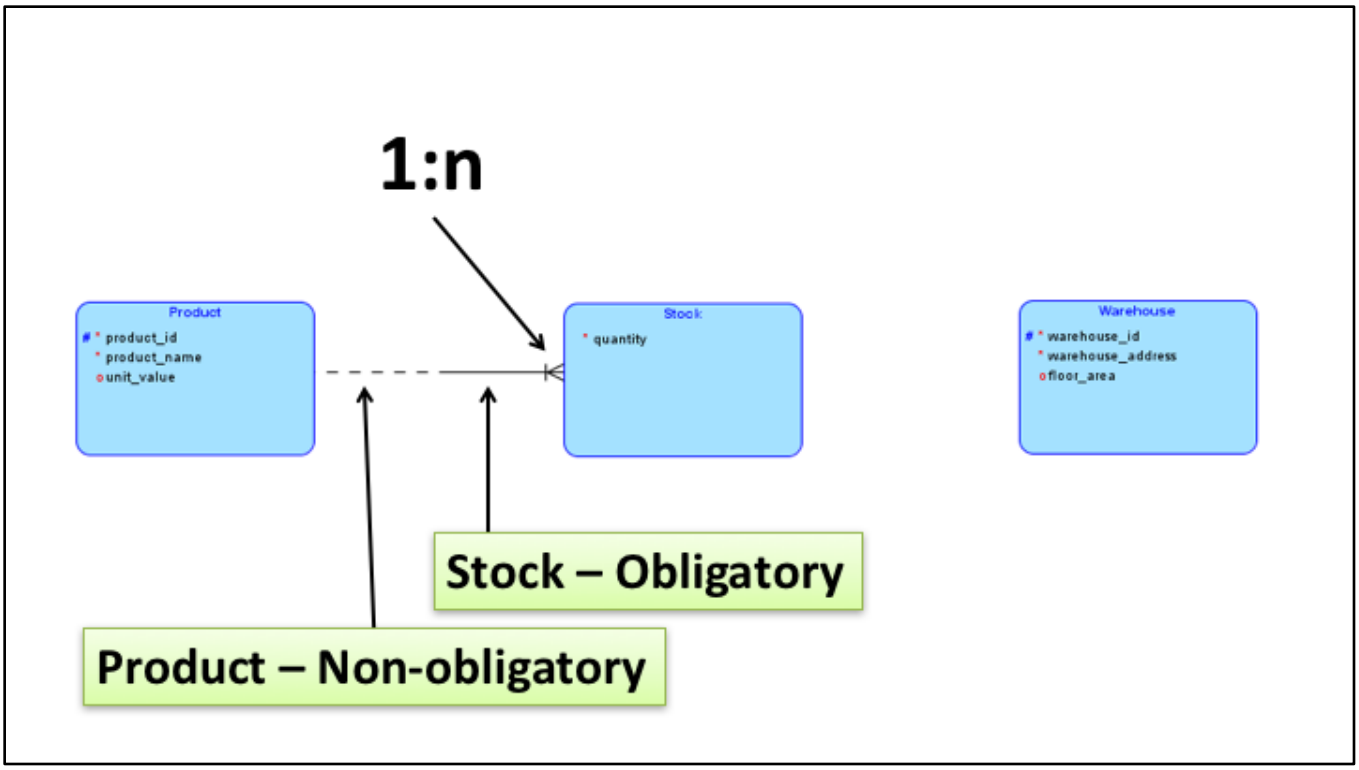


At most how many stocks can a product have? **n**

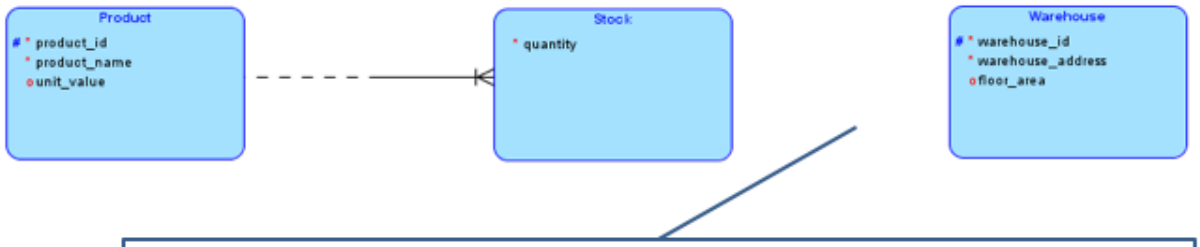
At most how many products can a stock be related to? **1**

1:n


Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area
At **least** how many stocks must a product have?
0
At **least** how many products must a sock be related to?
1
Product – Non-obligatory
Stock – Obligatory


1:n
Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area
Stock – Obligatory
Product – Non-obligatory

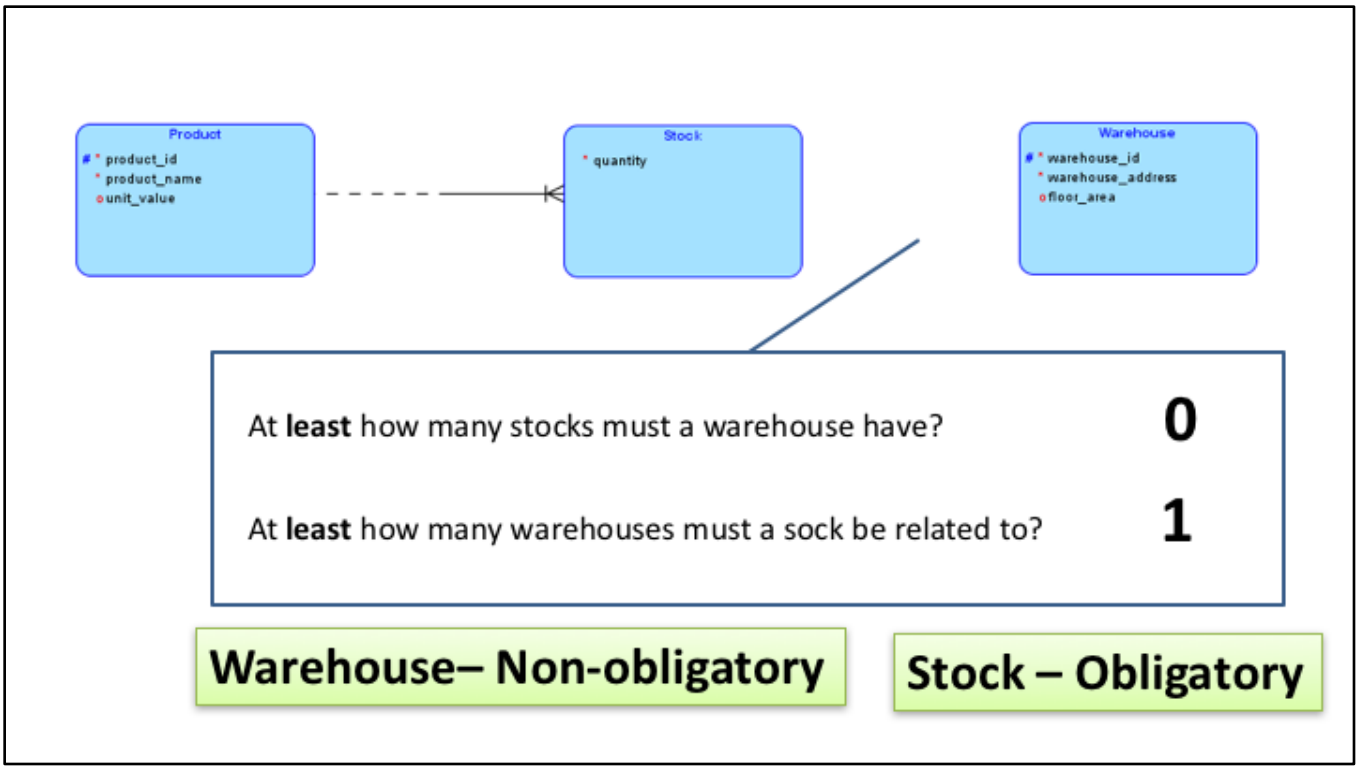
Relationship Degree

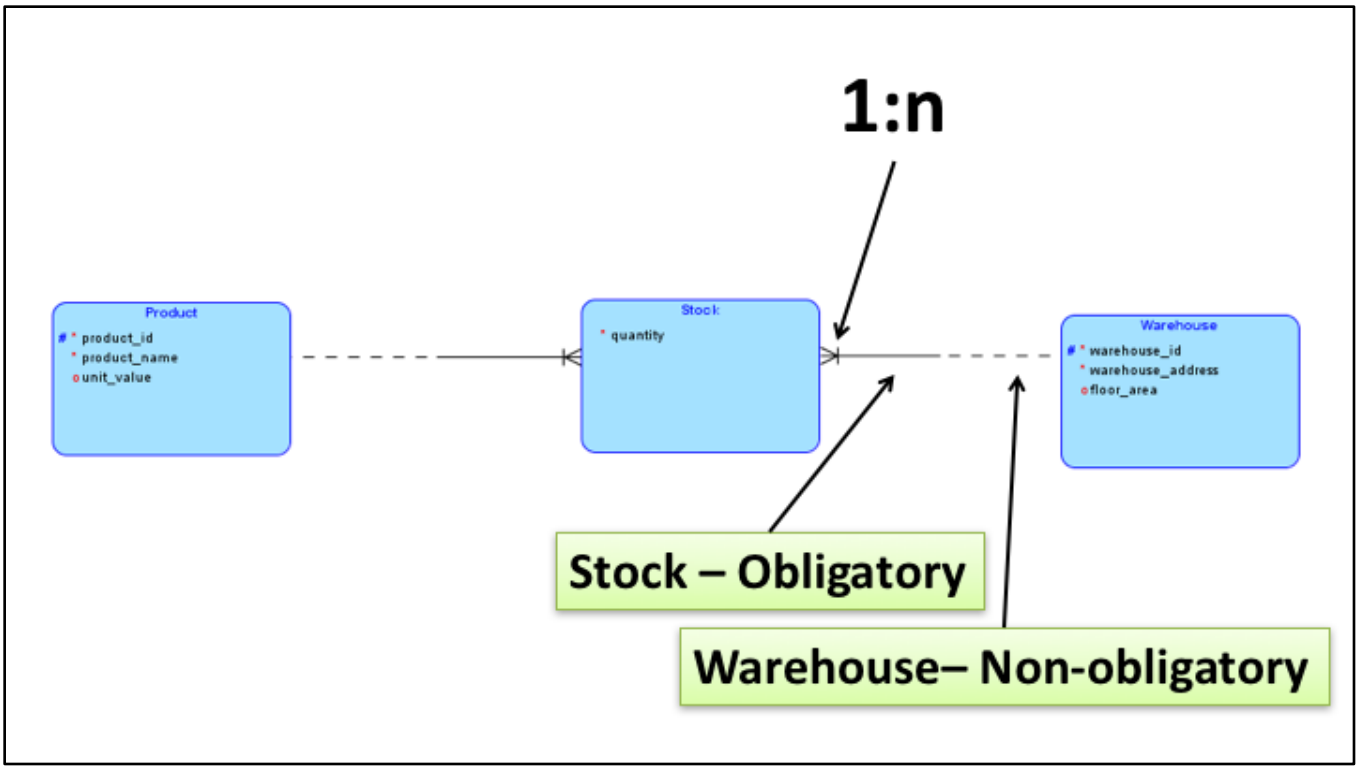


At most how many stocks can a warehouse have? **n**

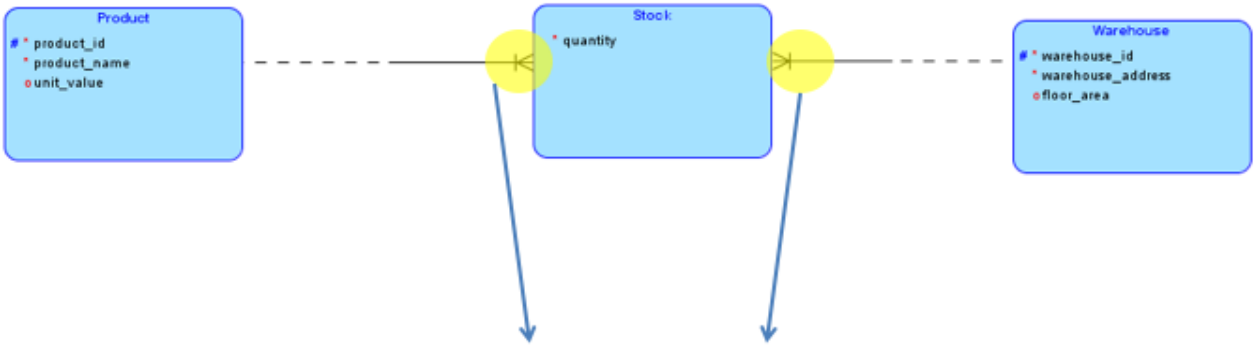
At most how many warehouses can a stock be related to? **1**

1:n

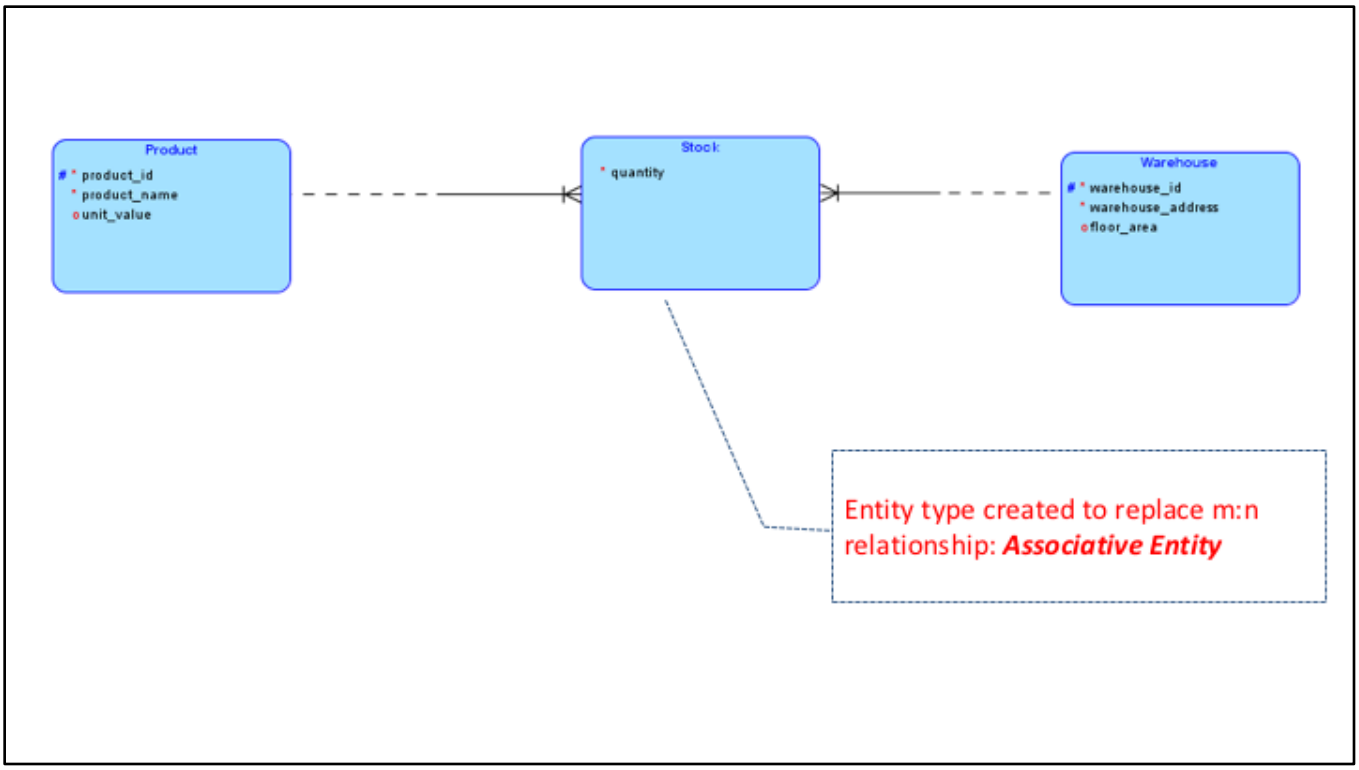

Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area
At **least** how many stocks must a warehouse have?
0
At **least** how many warehouses must a sock be related to?
1
Warehouse– Non-obligatory
Stock – Obligatory

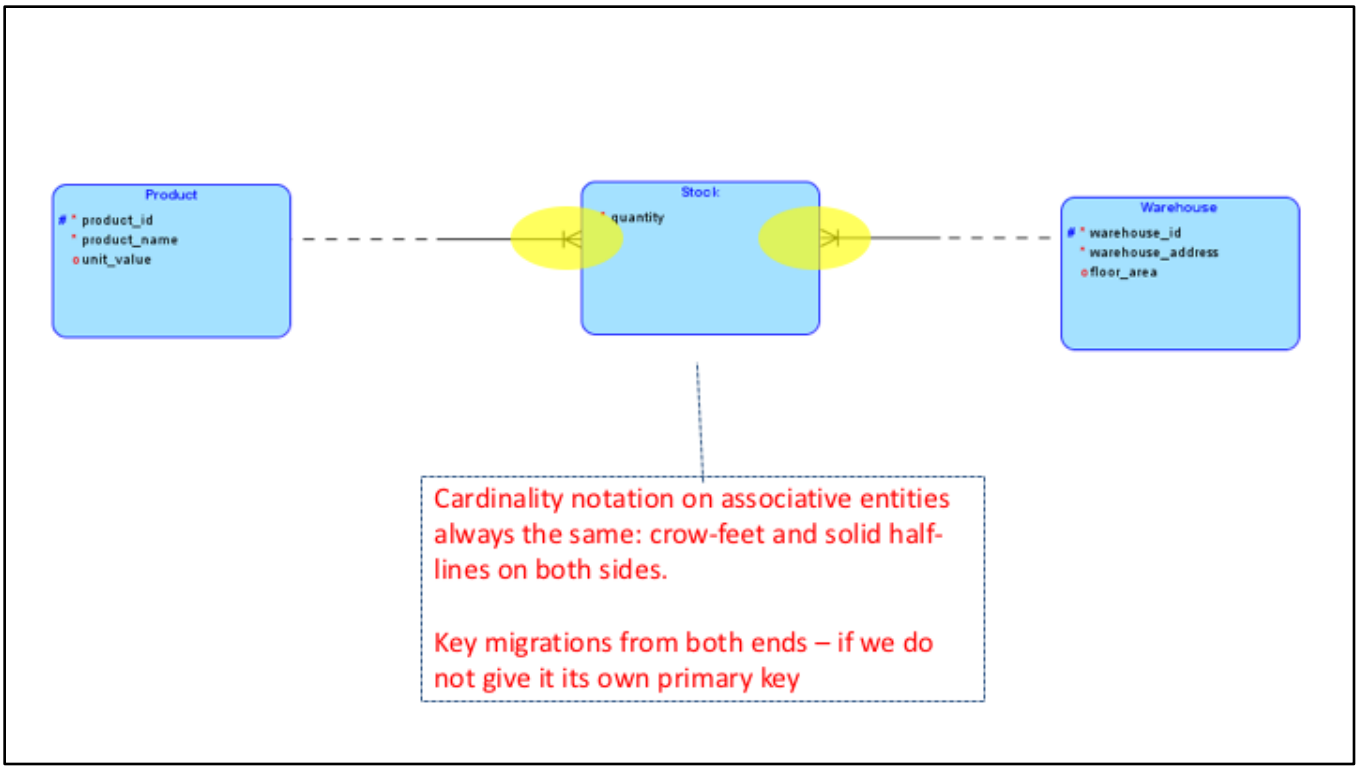

1:n
Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area
Stock – Obligatory
Warehouse– Non-obligatory

Primary Key for Stock?



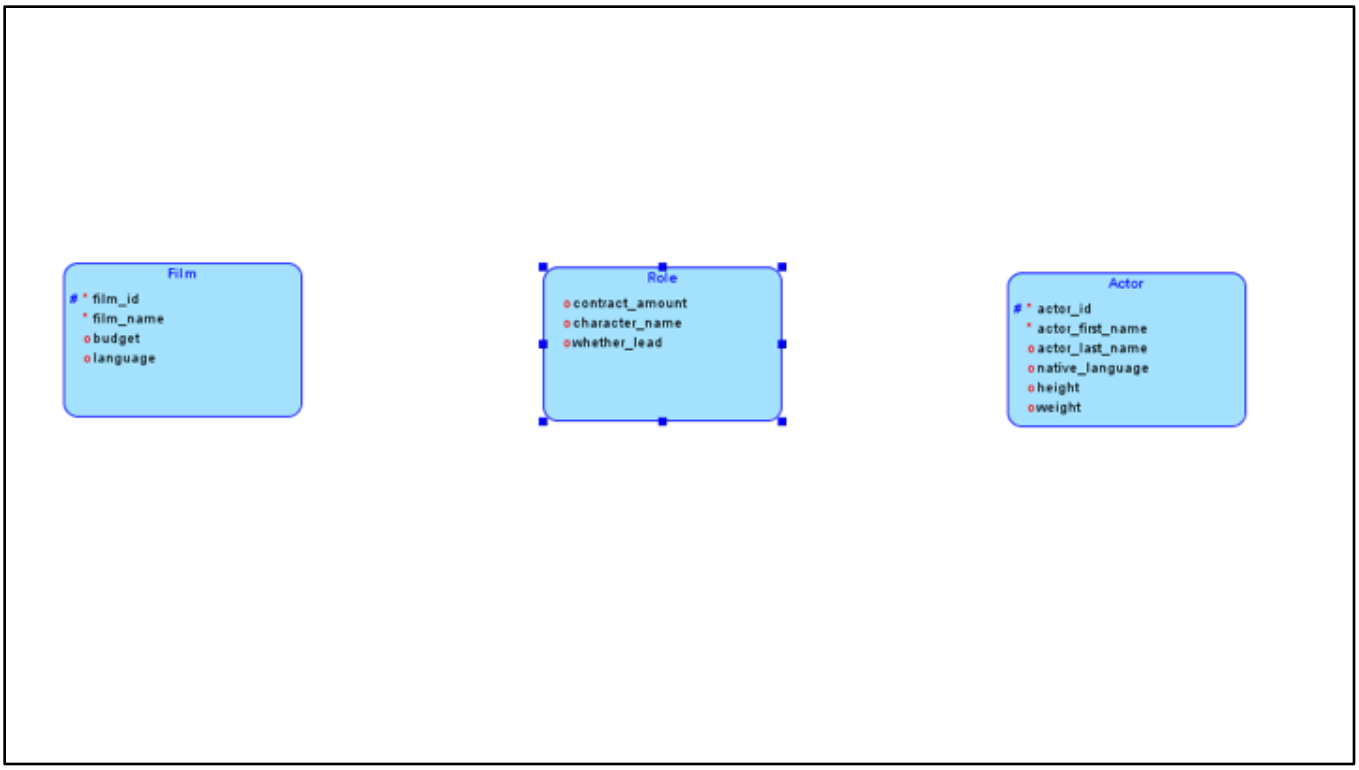
From key migrations: product_id + warehouse_id

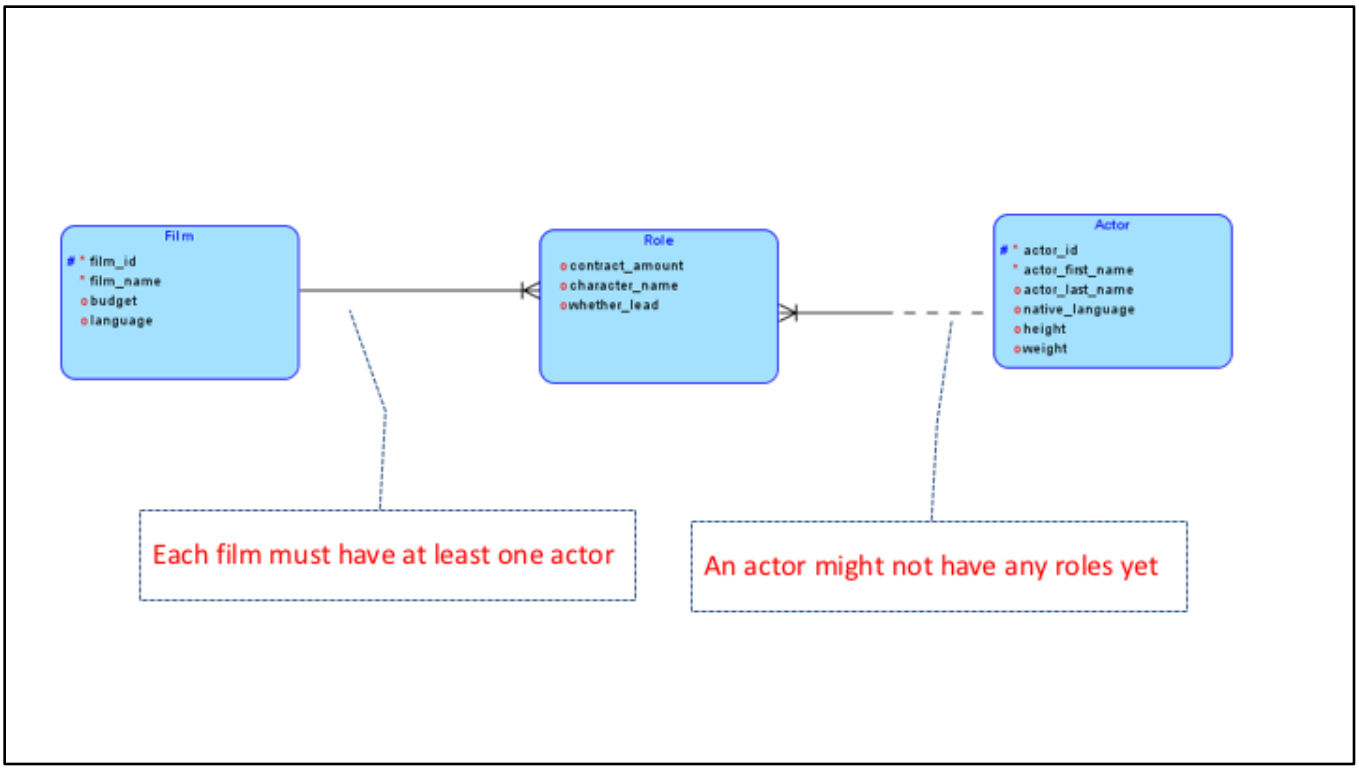

Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area
Entity type created to replace m:n relationship: Associative Entity

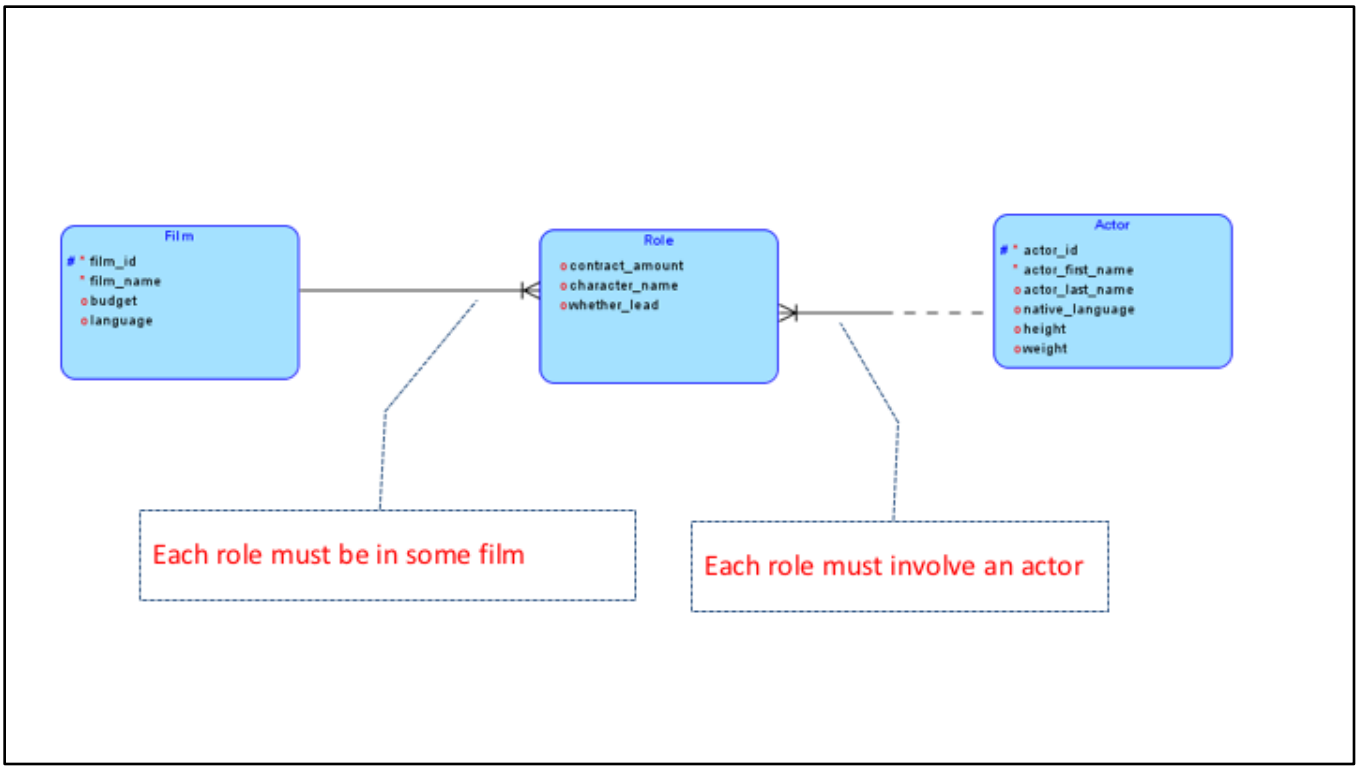

Product
* product_id
* product_name
o unit_value
Stock
* quantity
Warehouse
* warehouse_id
* warehouse_address
o floor_area
Cardinality notation on associative entities always the same: crow-feet and solid half-lines on both sides.
Key migrations from both ends – if we do not give it its own primary key

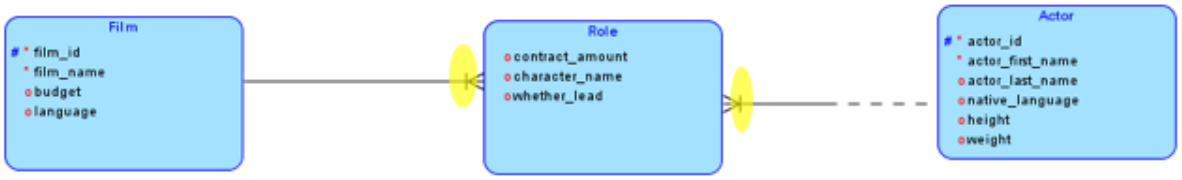
Hollywood scenario:

- A film studio produces many films.
- The studio has signed on many actors.
- Each actor might have a role on several films.
- Each film must involve one or more actors playing roles.
- Some actors might not yet have acted in any films.


Film
* film_id
* film_name
o budget
o language
Role
o contract_amount
o character_name
o whether_lead
Actor
* actor_id
* actor_first_name
o actor_last_name
o native_language
o height
o weight


Film
* film_id
* film_name
budget
language
Role
contract_amount
character_name
whether_lead
Actor
* actor_id
* actor_first_name
actor_last_name
native_language
height
weight
Each film must have at least one actor
An actor might not have any roles yet


Film
* film_id
* film_name
budget
language
Role
contract_amount
character_name
whether_lead
Actor
* actor_id
* actor_first_name
actor_last_name
native_language
height
weight
Each role must be in some film
Each role must involve an actor



Primary key for Role: film_id + actor_id