## E-R Guidelines (p.1)

1. **Fidelity**
   - Be logically true to the real-world domain that we are modeling.
   - Capture all of the real-world properties (*semantics*) of the domain as is possible.
2. **Brevity** (of the data)
   - The model should not require that a piece of information be represented more than once.
     - Should not be the possibility that the same information will appear many times in the same entity set.
     - If information can be *logically inferred* from other information in the model, it should not be kept.

## E-R Guidelines (p.2)

3. **Simplicity** (of the model)
   - *Occam's razor*: Keep the design as simple as possible.
     - Should contain no elements—entities, relationships, attributes, etc.—that are not necessary.
     - Should have as few connections as possible.
     - *Precedence*: Use attributes before entities, and relationships before entities.
4. **Naturalness**
   - Model as naturally as possible the domain.
     - Entities ought to correspond with real things.
     - Relationships should be understandable.

These guidelines are in order of precedence. For example, **brevity** takes precedence over **simplicity**.

## An Initial Design:

### A Starting Point

**What is your domain?**

- What questions does the database need to be able to answer?
- What *data processing* activities does the database need to support?

**What is your design?**

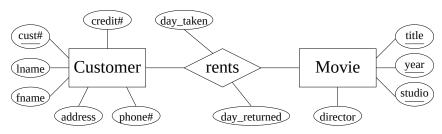
- Choose your entities, attributes, and key attributes.
- What are the important relationships among them?
- Any logical problems, or anything missing? If so, then refine the design.
- Can the design accommodate the questions and activities? If not, then refine the design.
- Can we violate any real-world constraints? If so, re-design (if possible) so we cannot.
- Simplify.

## An Example Domain

### A Movie Rental Database

**Scenerio**: We are running a movie rental store.

- We have *movies*.
- We have *customers* (we hope!).
- Customers *rent* movies from us.



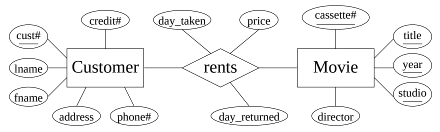
## A Good Design?

- Any logical problems, or anything missing?
- Can the design accommodate the questions and activities?
- Can we violate any real-world constraints?

**Anything missing?**

- Can add attributes, rel-ships, or entities.

**Problem**: We may have several copies of a given movie. More than one copy of a movie might be rented at the same time. We need to know which customer has which copy.

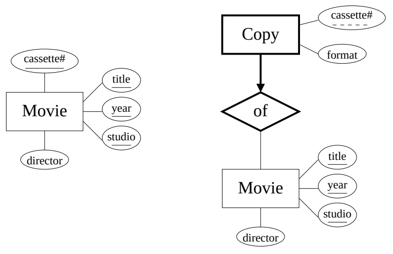


## Attribute vs. Entity

**When to promote an attribute to an entity?**

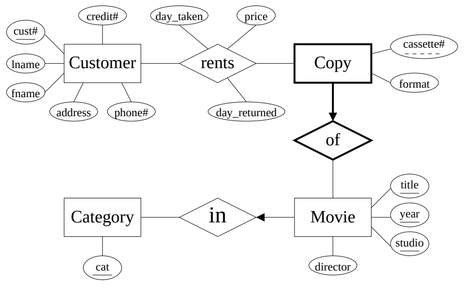
1. It needs to participate in rel-ships itself.
   - It must be an entity.
2. The values of the attribute in its entity would be repeated often, or there are other attributes associated with this one. (**Brevity**)
   E.g., professor has office#.
   - It probably should be an entity.
3. The values that the attribute may take are restricted. (*strong typing*)
   - Can accomplish this making it an entity.

## Movies and Copies



- The information about a movie is repeated for as many *copies* of that movie we have. This seems awkward.
  - Later, we will formalize why this is redundancy is actually problematic, and not just inelegant.
  - What if we want to catalog a movie but we have no copies yet?
- To fix this, we separate the notion of a *copy of a movie* and the *movie itself* (the listing of the movie). So we add an entity **Copy**.

## A Better Design?



- I also added an entity **Category** for movies (e.g., *comedy*, *drama*, *scifi*, etc.). This strongly types the *category* a movie can be in.

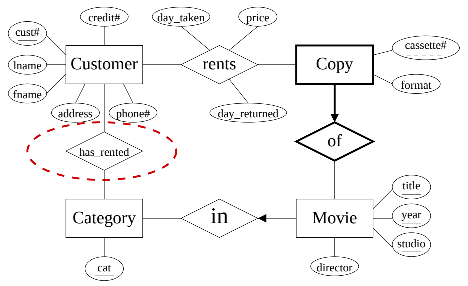
Are we done yet? Can we stop?

Not even close.

## New Relationships

### Careful for Redundancy

A requirment for our movie rental database is that we can determine which categories of movies a customer has rented (for promotions and such). Have we overlooked this?



**No!** We had not overlooked this. Which categories a customer has rented is *derivable* from which movies he or she has rented.

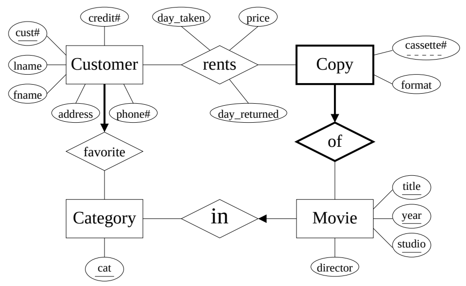
The rel-ship has_rented would be redundant. It represents a new rel-ship that could be populated with completely unrelated data. So, has_rented should *not* be added.

## New Relationships

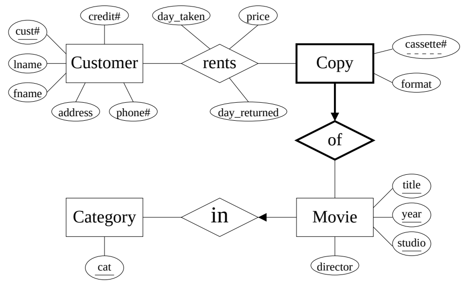
### But if not redundant. . .

However, if a new rel-ship represents new information which is not derivable from what we already have and which we need for our intended queries and applications, then we should add it.

For example, say we want to know a customer's favorite category (e.g., *romance*) for promotional purposes.



## Trouble and Ambiguity



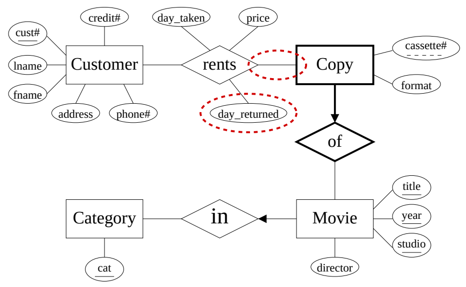
**Missed realities / business rules**:

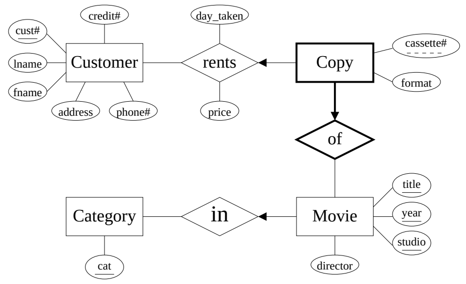
1. A given copy cannot be simulateously rented at the same time by two or more customers. (The in-two-places-at-once phenomenon!)
2. A given customer may rent the same copy of a movie more than one time.

Which do we miss? I don't know, but we do miss at least one of them!

## Interpretation One

### Only track movies that are out





## Interpretation Two

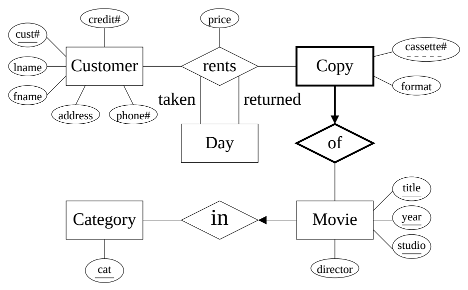
**Record all rentals forever**

Once a rental is recorded in the database, it stays.

**First Problem**: A customer may rent (the same copy of) a movie more than one time. Rel-ship rents does not allow this. Solutions?

- Have more entities participate in rents to fix this. (A ternary relationship.)
- Promote rents to be an entity (**Rental**?).

## A "Ternary" Relationship Attempt

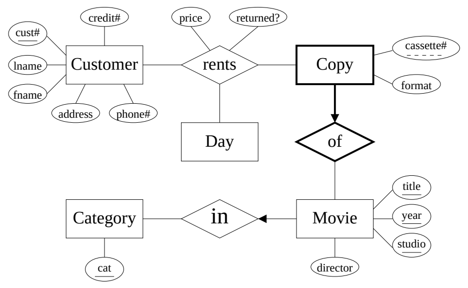


**Problems**:

- Should **Day** be an entity? Do we really want to "store" days explicitly? (Awkward)
- Rel-ship rents is **Customer** × **Copy** × **Day** × **Day**. But is returned always filled in? **No!**

## A Ternary Relationship Solution

**Slightly better...**



Fixes the awkward aspect that the *returned day* is not known for any copies currently out by making it an attribute again.

Two customers still can rent the same copy of a movie on the same day. Why did the previous design not handle this? Why does this design not handle this? Can you fix this?

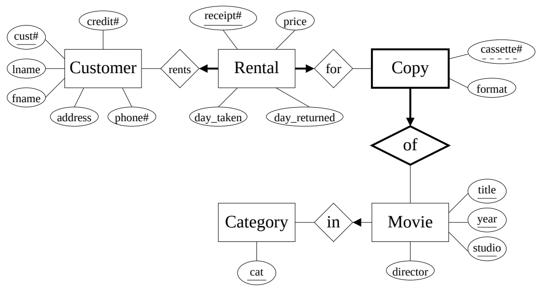
(And still overall an awkward design anyway.)

## Relationship vs. Entity:

**When to promote a rel-ship to entity?**

1. When the "key" of the rel-ship is too restrictive.
   - Can promote the rel-ship to be an entity instead. Then it has a key.
   - *Or* the rel-ship should involve more entities so that its "key" is adequate.
2. When it seems that we need this rel-ship to be involved itself in other rel-ships.
   - Promote the rel-ship to be an entity. Then it can participate in rel-ships with other entities.
   - *Or* use aggregation.

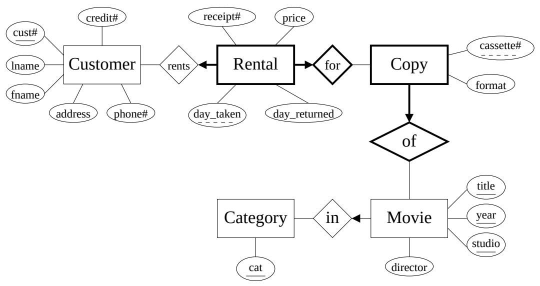
## An Entity Solution



- Handles problem one: a customer can rent the same copy a second time.
- Still does not handle problem two: Two customers may rent the same copy at the same time.

## An Entity Solution

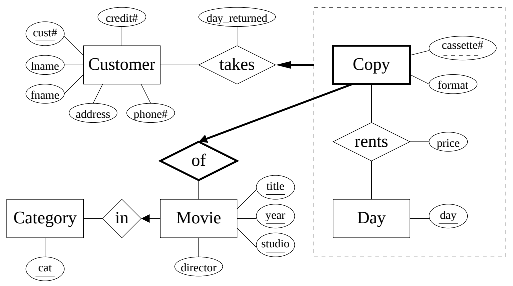
### Better keys



Handles both problems now!

(Can an entity have more than one key?)

## An Aggregation Solution



- Again, do we want **Day** as an entity?
  (**Foreshadowing**: However, note that in our conversion to a relational design, we won't really have to create a table **Day**.)
- Doesn't seem to score well on **simplicity**. But it does say what we want (**fidelity**).