>>

ER→Relational Mapping

- ER to Relational Mapping
- Relational Model vs ER Model
- Mapping Strong Entities
- Mapping Weak Entities
- Mapping N:M Relationships
- Mapping 1:N Relationships
- Mapping 1:1 Relationships
- Mapping n-way Relationships
- Mapping Composite Attributes
- Mapping Multi-valued Attributes (MVAs)
- Mapping Subclasses

^ >>

❖ ER to Relational Mapping

Reminder: a useful strategy for database design:

- perform initial data modelling using ER
  (conceptual-level modelling)
- transform conceptual design into relational model
  (implementation-level modelling)

A formal mapping exists for ER model → Relational model.

This maps "structures"; but additional info is needed, e.g.

- concrete domains for attributes and other constraints

<< ^ >>

❖ Relational Model vs ER Model

Correspondences between relational and ER data models:

entity means an instance of an entity

- attribute(ER) ≅ attribute(Rel), entity(ER) ≅ tuple(Rel)
- entity set(ER) ≅ relation(Rel), relationship(ER) ≅ relation(Rel)

entity set means what i think of when i say entity (remember how he talked about notation abuse)

Differences between relational and ER models:

- Rel uses relations to model entities *and* relationships
- Rel has no composite or multi-valued attributes (only atomic)
- Rel has no object-oriented notions (e.g. subclasses, inheritance)

Note that ...

- not all aspects of ER cab be represented exactly in a relational schema
- some aspects of relational schemas (e.g. domains) do not appear in ER

Notes: the ER model naming conventions are a little bonkers. When I say entity, I think of the generic form of an entity, not a specific instance. In reality, the generic form is supposed to be called an Entity Set and Entity does mean a specific instance e.g. Car(make, model, cc) would be the Entity Set, and Car(2012, Mitsubishi Mirage, 1200) would be an Entity

The Relation model also has some weird names. It calls rows tuples.

Note that in ER, Entity sets and Relationships are separate things (square vs diamond), but in the relational model, they are both just relations (tables).

<< ^ >>

❖ Mapping Strong Entities

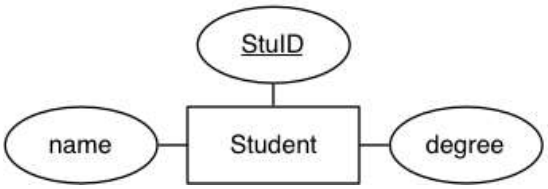
An entity set *E* with atomic attributes $a_1, a_2, \ldots a_n$ i.e not composite attributes aka "tree" attributes

maps to

A relation *R* with attributes (columns) $a_1, a_2, \ldots a_n$

Example:

ER Model



Relational Version

Student

StuID	name	address

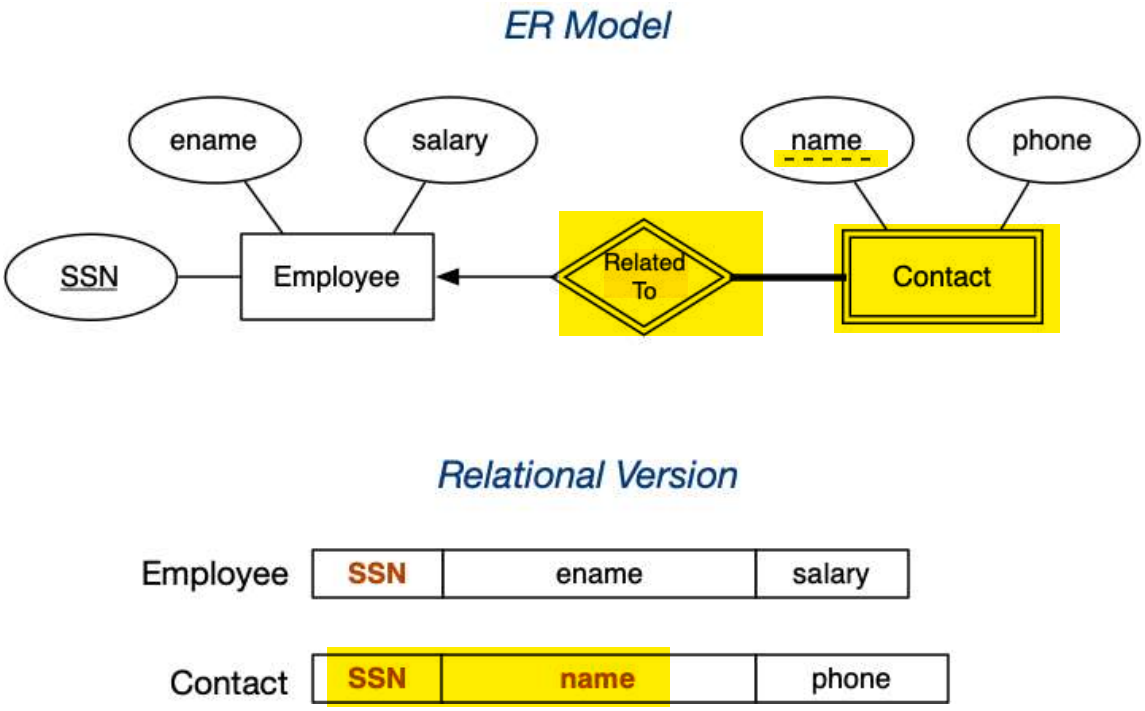
this should say degree, not address

Note: the key is preserved in the mapping.

<< ^ >>

❖ Mapping Weak Entities

Example:

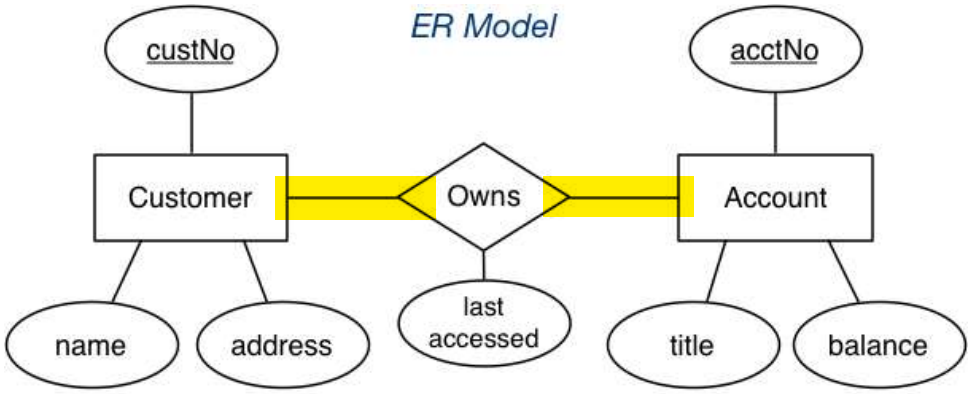


The important thing to note here is that weak entities become their own relation (table), which references the strong entity they are associated with via using the strong entities' pkey as a fkey. This fkey (SSN in this example), is combined with the weak entities discriminator (name in this example), to form a pkey for the weak entities relation (table).

<< ^ >>

❖ Mapping N:M Relationships

Example:



Relational Version

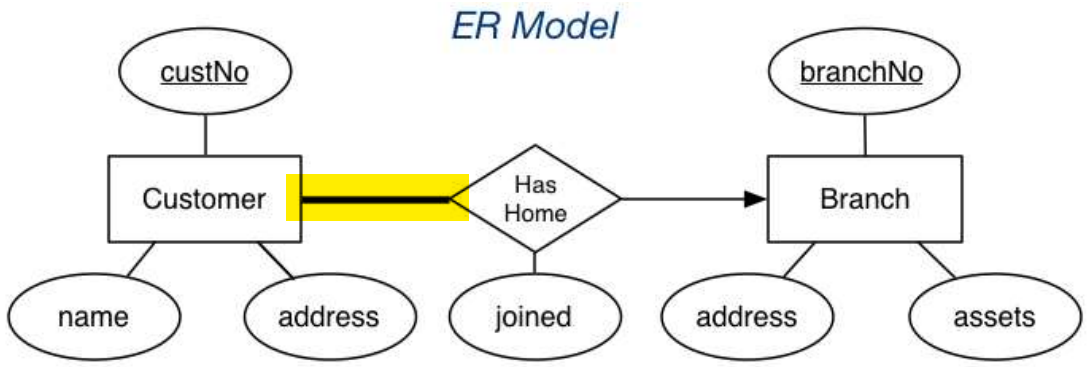
Customer	**custNo**	name	address
Account	**acctNo**	title	balance
Owns	**acctNo**	**custNo**	lastAccessed

What uniquely identifies a relationship? It is the entities involved in that relationship, and because entities are identified by pkeys, we need to store those pkey's as fkey's on the relation as its composite pkey. For example, the Owns relationship above is uniquely identified by which customer owns which acount, and because customers are identified by custNo and accounts are identified by acctNo, the Owns table must store acctNo and custNo as fkeys on it. They combine to make the pkey of Owns.
The relationship table then gets any other needed attributes (e.g. lastAccessed).

<< ^ >>

❖ Mapping 1:N Relationships

Example:



Relational Version

Customer	custNo	name	address	branchNo	joined

Branch	branchNo	address	assets

A customer MUST have ONE branch as their home.
A branch MAY be home to ONE OR MORE customers.

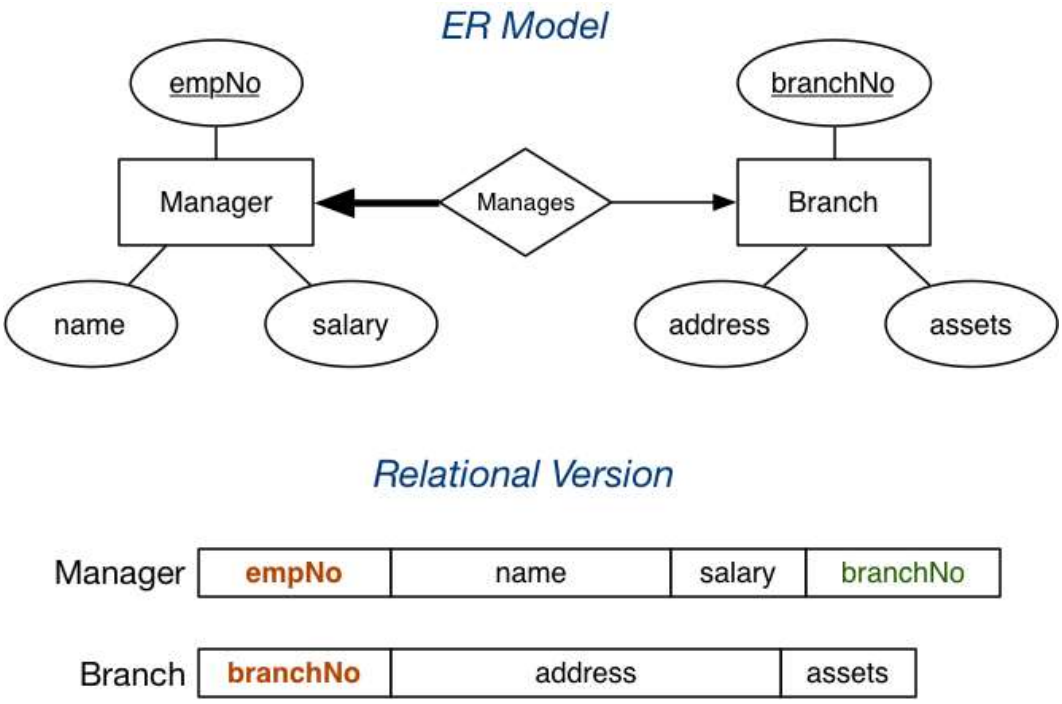
The customer is the one who must participate, so we can safely store the relationship details on them. For example, we give the customer their regular attributes (custNo, name, address), but then we also give them an fkey for the branch and any attributes the relationship has (in this case, the joined attribute).

In general, put the attributes of the relationship on the entity who must participate, and also the primary key of the other entity as an fkey.
This makes complete sense! Why would you store it on the entity who isn't always required to participate when you know for every instance of the relation there is at least one guaranteed entity that you can put it on.

<< ^ >>

❖ Mapping 1:1 Relationships

Example:



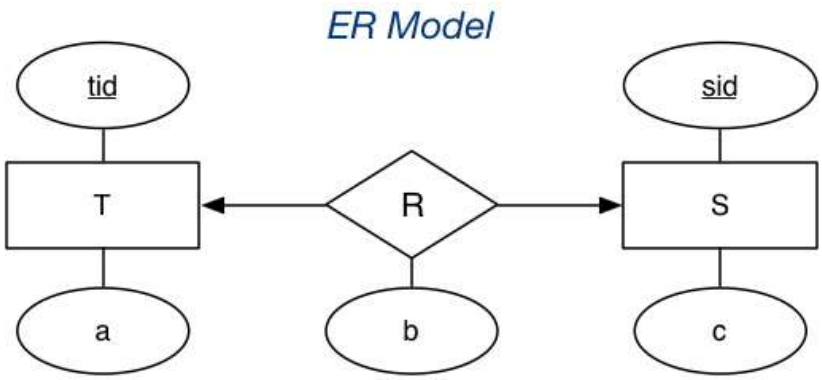
A manager MUST manage EXACTLY ONE branch.
A branch MAY be managed by EXACTLY ONE branch.

Again, put the details on the entity who must participte, in this case, the Manager.

<< ^ >>

❖ Mapping 1:1 Relationships (cont)

If there is no reason to favour one side of the relationship ...



Relational Version #1

T | tid | a | sid | b

S | sid | c

Relational Version #2

T | tid | a

S | sid | c | tid | b

This answers the question of: What if neither side MUST participate? For example;

A T MAY R EXACTLY ONE S.
An S MAY be R'd by EXACTLY ONE T.

They both MAY be involved in the relationship i.e. neither MUST participate. In this case there is no reason to pick one over the other, so just pick one and do what was done before (copy over the pkey of the un-chosen one to the chosen one as an fkey, and also give all of the relationship attributes to the chosen one).

<< ^ >>

❖ Mapping n-way Relationships

Relationship mappings above assume binary relationship.

If multiple entities are involved:

- *n:m* generalises naturally to *n:m:p:q*
  - include foreign key for each participating entity
  - include any other attributes of the relationship
- other multiplicities (e.g. *1:n:m*) ...
  - need to be mapped the same as *n:m:p:q*
  - so not quite an accurate mapping of the ER

Some people advocate converting n-way relationships into:

- a new entity, and a set of *n* binary relationships

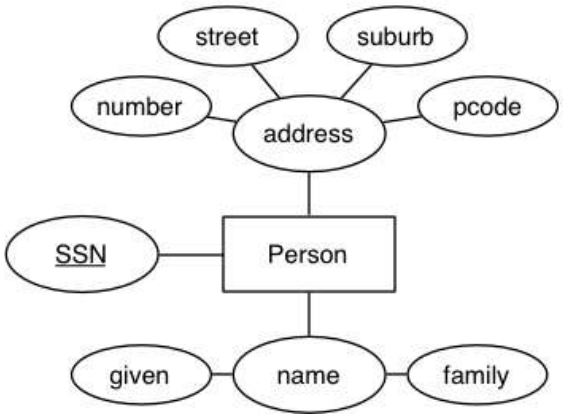
<< ^ >>

❖ Mapping Composite Attributes

Composite attributes are mapped by concatenation or flattening.

Example:

ER Model



Relational Version #1

Person

SSN	name	address

the parts become one

Relational Version #2

Person — store the parts concatenated

SSN	given	family	.....

.....	number	street	suburb	pcode

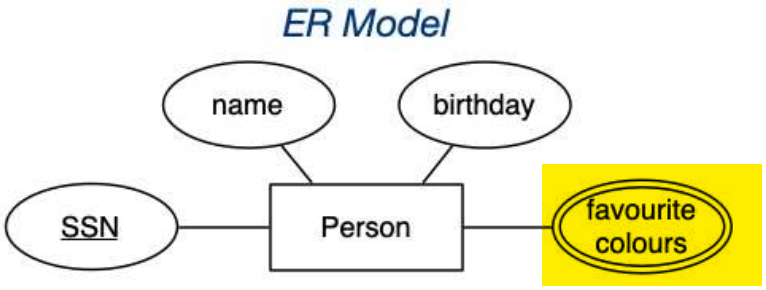
<< ^ >>

❖ Mapping Multi-valued Attributes (MVAs)

MVAs are mapped by a new table linking values to their entity.

Example:

OR by storing the MVA as an array



Relational Version

Person	**SSN**	name	birthday

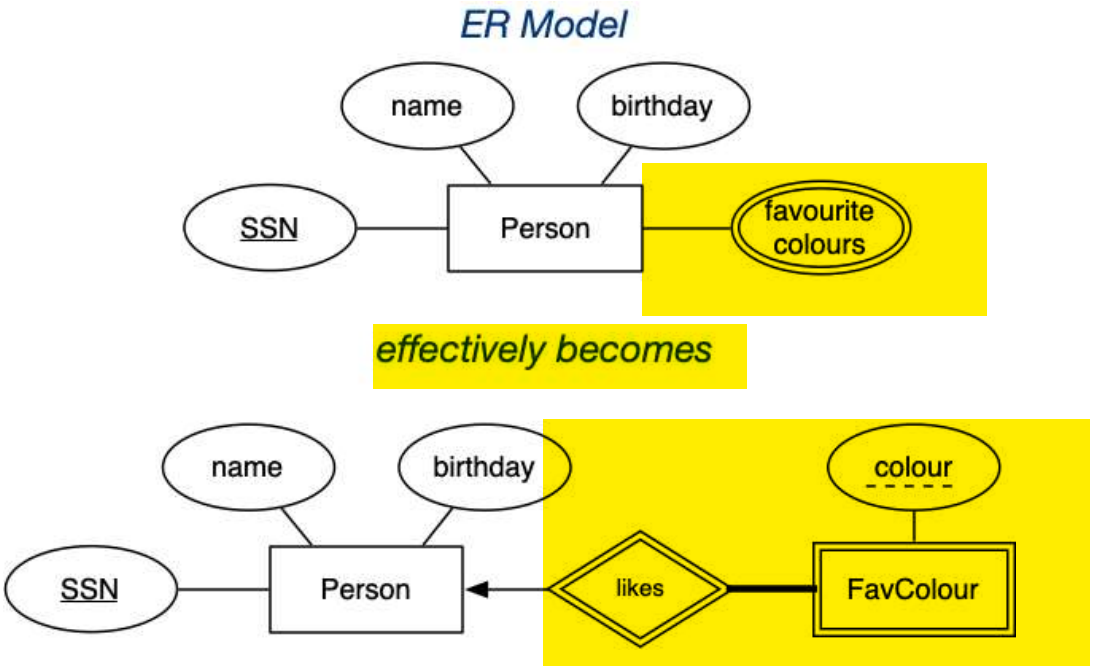
FavColour	**SSN**	colour

SSN + colour would be pkey here, SSN is fkey to Person

<< ^ >>

❖ Mapping Multi-valued Attributes (MVAs) (cont)

Analogy:



<< ^ >>

❖ Mapping Multi-valued Attributes (MVAs) (cont)

Example: the two entities

```
Person(12345, John, 12-feb-1990, [red,green,blue])
Person(54321, Jane, 25-dec-1990, [green,purple])
```

would be represented as

```
Person(12345, John, 12-feb-1990)
Person(54321, Jane, 25-dec-1990)
FavColour(12345, red)
FavColour(12345, green)
FavColour(12345, blue)
FavColour(54321, green)
FavColour(54321, purple)
```

<< ^ >>

❖ Mapping Subclasses

Three different approaches to mapping subclasses to tables:

- ER style
  - each entity becomes a separate table,
  - containing attributes of subclass + FK to superclass table

  n tables with only their specific attributes and fkeys to refer to their parent tables.
- object-oriented
  - each entity becomes a separate table,
  - inheriting all attributes from all superclasses

  n tables where each has its own attributes plus those of all its parents.
- single table with nulls
  - whole class hierarchy becomes one table,
  - containing all attributes of all subclasses (null, if unused)

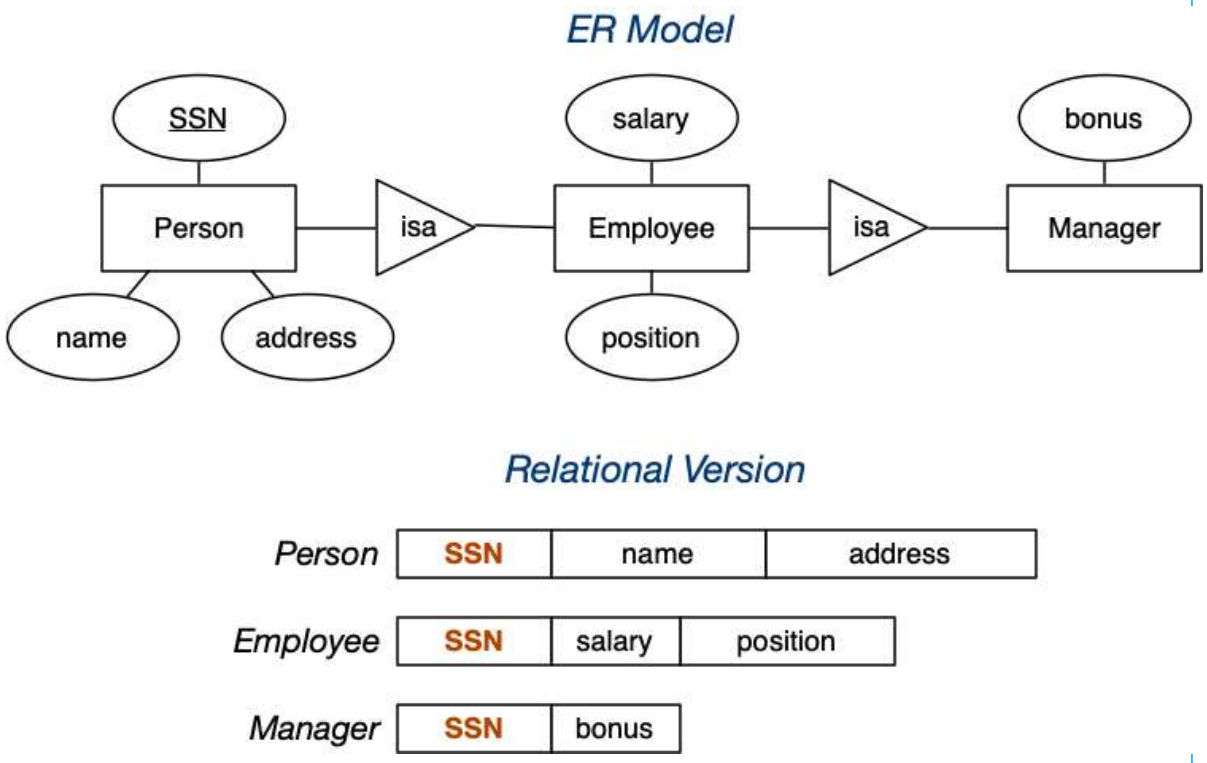
  1 mega-table that can be used to represent any of the n types.

Which mapping is best depends on how data is to be used.

<< ^ >>

❖ Mapping Subclasses (cont)

Example of ER-style mapping:



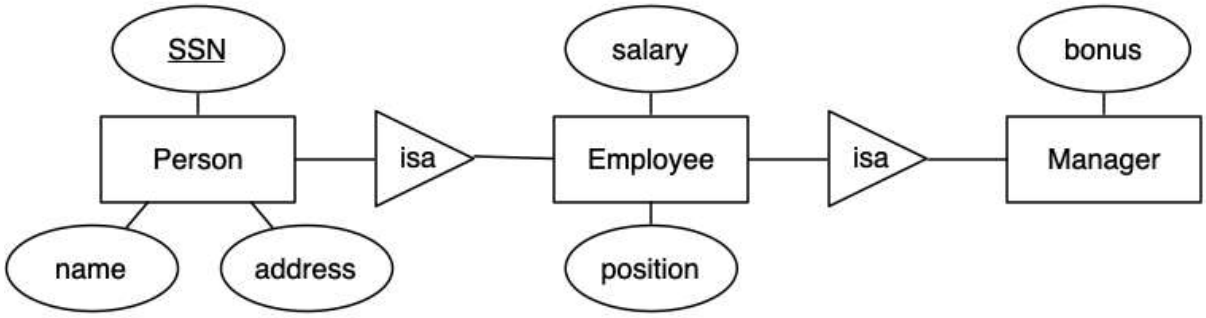
To get to the Employee from Manager, you query for all Employees with SSN = foo.

<< ^ >>

❖ Mapping Subclasses (cont)

Example of object-oriented mapping:

ER Model



Relational Version

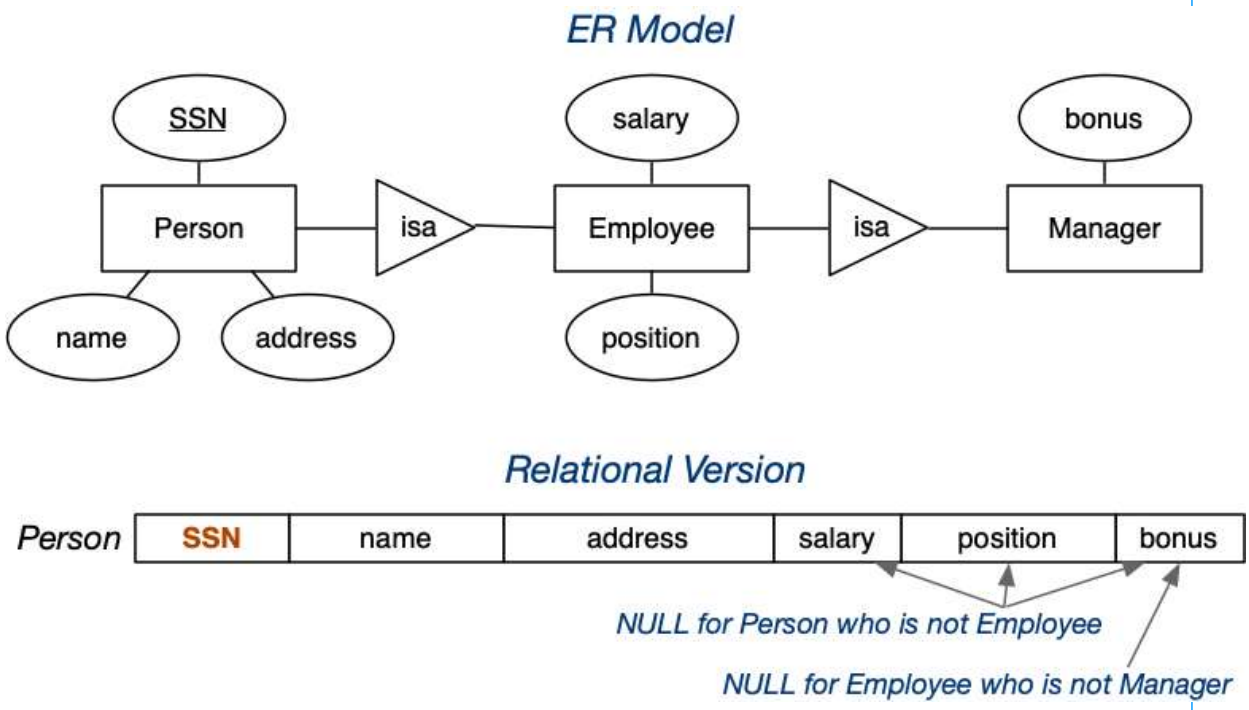
Person	SSN	name	address			
Employee	SSN	name	address	salary	position	
Manager	SSN	name	address	salary	position	bonus

attributes for each entity are all parents + own, hence you get bigger tables the lower in the heirarchy you go.

<< ^

❖ Mapping Subclasses (cont)

Example of single-table-with-nulls mapping:



salary onwards will ALWAYS be null for an instance of this table that is representing a Person, bonus will ALWAYS be NULL for any instances of Employee, Manager will have all fields.

Produced: 15 Sep 2020