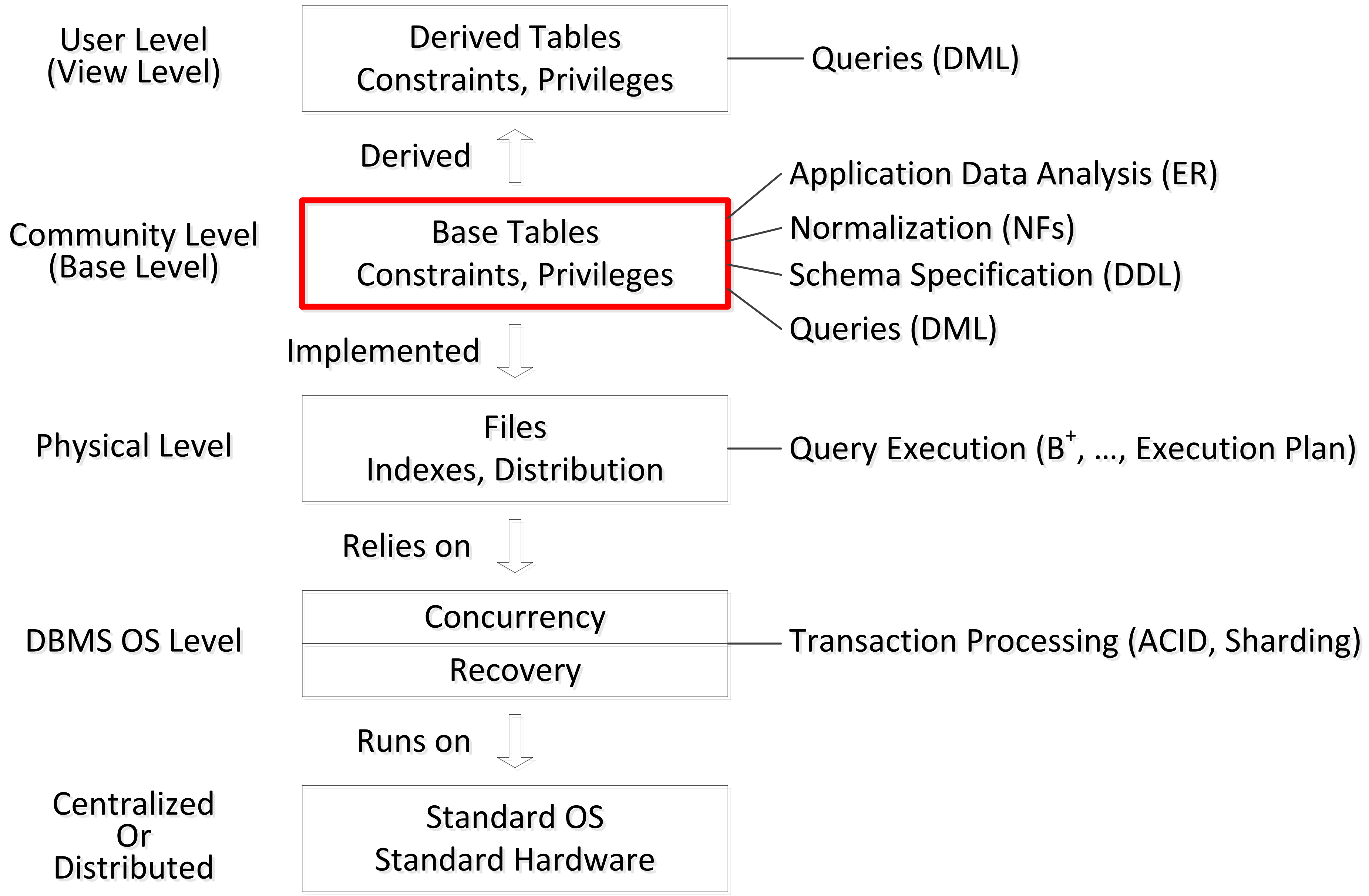
Unit 2
Modeling the Information of an Enterprise Using Chen’s Entity/Relationship Model and Diagrams

ER Diagrams in Context



Introduction

Purpose of ER Model and Basic Concepts

- ***Entity/relationship (ER) model*** provides a common, informal, and convenient method for communication between application end users (customers) and the database designers in order to model the information’s structure
- This is ***conceptual design***, though some call this ***logical design***
- The ER model, typically employs ***ER diagrams***, which are pictorial descriptions to visualize information’s structure

Purpose of ER Model and Basic Concepts

- There are three basic concepts appearing in the original ER model, which has since been extended
  - We will present the model from more simple to more complex concepts, with examples on the way
- We will go beyond the original ER model, and cover most of the ***Enhanced ER*** model
- While the ER model's ***concepts are standard***, there are several ***varieties of pictorial representations*** of ER diagrams
  - We will focus on one of them: ***Chen's*** notation
  - We will also cover ***Crow's foot*** notation, which is particularly suited for relational database design
  - Others are simple variations, so if we understand the above, we can easily understand all of them
- You can look at some examples at: http://en.wikipedia.org/wiki/Entity-relationship_model

Basic Concepts

- The three basic concepts are (elaborated on very soon):
- ***Entity***. This is an "object." Cannot be defined even close to a formal way. Examples:
  - Bob
  - Boston
  - The country whose capital is Paris

    There is only one such country so it is completely specified
- ***Relationship***. Entities participate in relationships with each other. Examples:
  - Alice and Boston are in relationship Likes (Alice likes Boston)
  - Bob and Atlanta are not in this relationship
- ***Attribute*** (property). Examples:
  - Age is an attribute of Alice
  - Size is an attribute of Boston

Entity

Entity and Entity Set

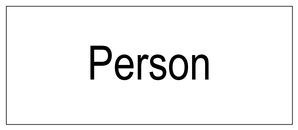
- ***Entity*** is a “thing” that is distinguished from others in our application
  - Example: Alice
- All entities of the same “type” form an ***entity set***; we use the term “type” informally for entity sets
  - Example: Person (actually a set of persons). Alice is an entity in this entity set
- What type is a little tricky sometimes

Entity and Entity Set

- Example. When we say “the set of all Boeing airplanes,” is this
  - The set of all models appearing in Boeing’s catalog (abstract objects), or
  - The set of airplanes that Boeing manufactured (concrete objects)?
- We may be interested in both and have two entity sets that are appropriately related
- We will frequently use the term “entity” while actually referring to entity sets, unless this causes confusion

Entity and Entity Set

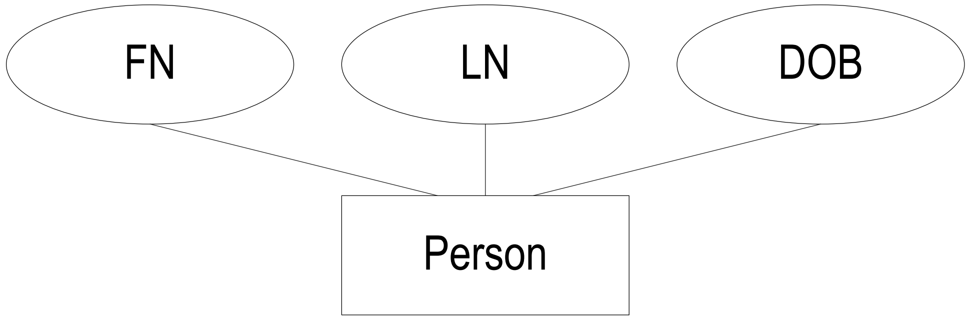
- Pictorially, an entity set is denoted by a ***rectangle*** with its type written inside
- By convention, singular noun,
- By convention capitalized or all capitals if acronym



Attribute

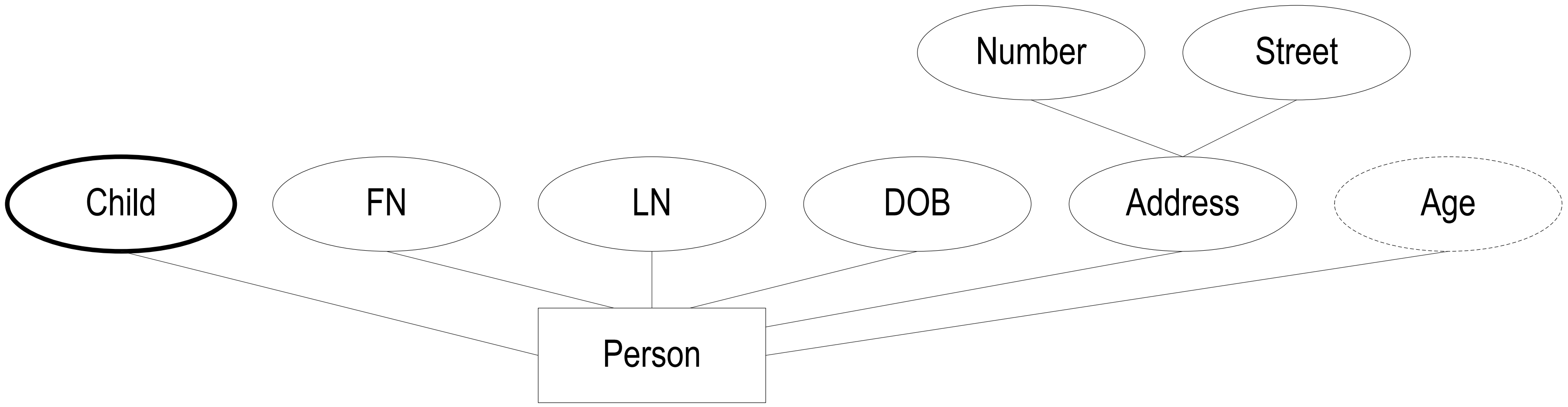
Attribute

- An entity may have (and in general has) a set of zero or more ***attributes***, which are some properties
- Each attribute is drawn from some domain (such as integers) possibly augmented by **NULL** (think about meanings for null)
- All the entities in an entity set have the same set of attributes, though not generally with the same values
- Attributes of an entity are written in ellipses (for now solid lines) connected to the entity
  - Example: FN: “First Name.” LN: “Last Name.” DOB: “Date of Birth.”



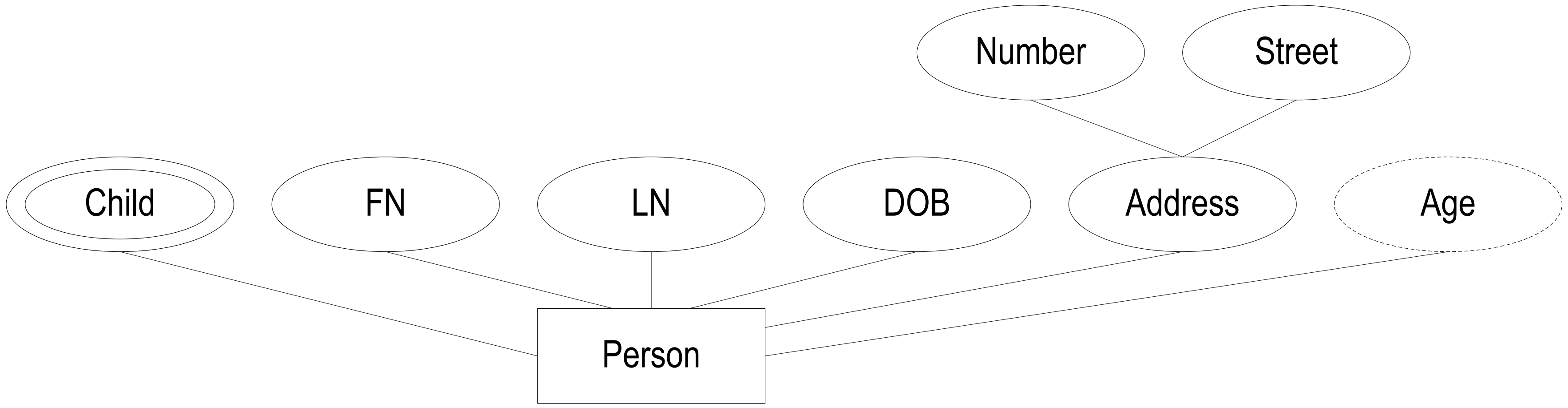
Attribute

- Attributes can be
  - ***Base*** (such as DOB) or ***derived*** denoted by dashed ellipses (such as Age, derived from DOB and the current date)
  - ***Simple*** (such as DOB) or ***composite*** having their component attributes attached to them (such as Address, when we think of it explicitly as consisting of street and number and restricting ourselves to one city only)
  - ***Singlevalued*** (such as DOB) or ***multivalued*** with unspecified in advance number of values denoted by thick-lined ellipses (such as Child; a person may have any number of children; we do not consider children as persons in this example, this means that they are not elements of the entity set Person, just attributes of the entities in this set)



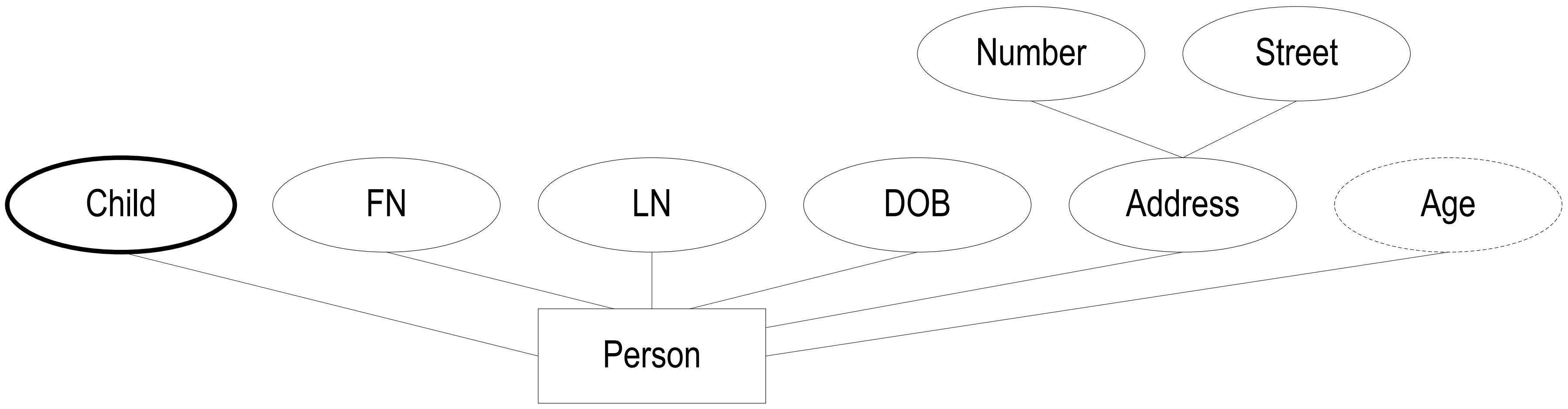
Thick Lines or Double Lines

- Some people (not we) use the convention of using double lines whenever we use thick lines
- We will use thick lines
- Using our example, the ER diagram using double lines would look like below



Attribute

- To have a simple example of a person with attributes
  - Child: Bob
  - Child: Carol
  - FN: Alice
  - LN: Xie
  - DOB: 1980-01-01
  - Address.Number: 100
  - Address.Street: Mercer
  - Age: Current Date minus DOB specified in years (rounded down)



Keys

Sets, Subsets, and Supersets

- Relations **subset** and **superset** are defined among sets $\subseteq$
- This is analogous to $\leq$
- Let us review by an example of three sets
  - A = {2,5,6}
  - B = {1,2,5,6,8}
  - C = {2,5,6}
- Then we have
  - $A \subseteq B$ and A is a subset of B

    and A is a **proper subset**, as it is not all of B; $A \subset B$
  - $A \subseteq C$ and A is a subset of C

    and A is not a proper subset, as it is equal to C; A = C

Keys

- Most of the time, some subset (proper or not) of the attributes of an entity has the property that two different entities in an entity set must differ on the values of these attributes
- This must hold for all conceivable entities in our database
- Such a set of attributes is called a ***superkey*** ("weak" superset of a key: either proper superset or equal)
- A minimal superkey is called a ***key***
  - This means that no proper subset of it is itself a superkey

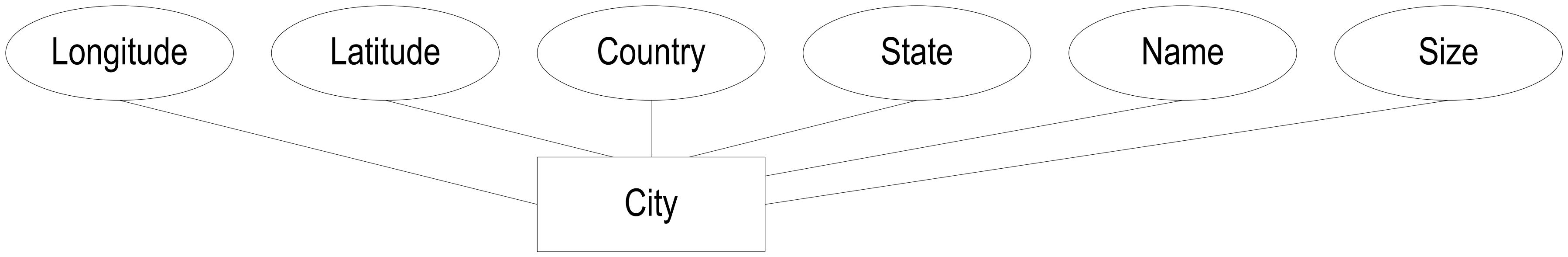
Keys

- Informally: superkey values can identify an individual entity but there may be unnecessary attributes
- Informally: key value can identify an individual entity but there are no unnecessary attributes

- Example: Social Security Number + Last Name form a superkey, which is not a key as Social Security Number is enough to identify a person

Keys

- In example below:
  - Longitude and Latitude (their values) identify (at most) one City, but only Longitude or only Latitude do not
  - (Longitude, Latitude) form a superkey, which is also a key
  - (Longitude, Latitude, Size, Name) form a superkey, which is not a key, because Size and Name are superfluous
- For simplicity, we assume that every country is divided into states and within a state the city name is unique



Relationship

Relationship

- Several entity sets (one or more) can participate in a ***relationship***
- Relationships are denoted by diamonds, to which the participating entities are "attached"
- A relationship could be binary, ternary, or more
- By convention, a capitalized verb in third person singular (e.g., Likes), though we may not adhere to this convention if not adhering to it makes things clearer

Relationship

- We will have some examples of relationships
- We will use three entity sets, with entities (and their attributes) in those entity sets listed below

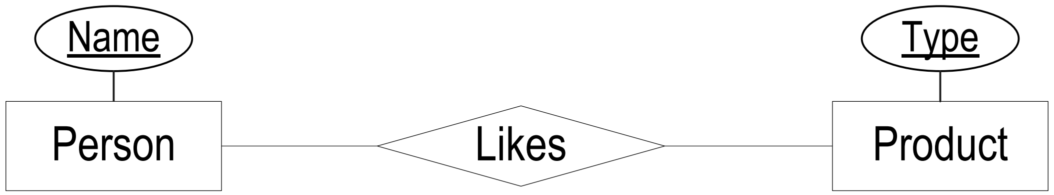
Person	Name
	Chee
	Lakshmi
	Marsha
	Michael
	Jinyang

Vendor	Company
	IBM
	Apple
	Dell
	HP

Product	Type
	computer
	monitor
	printer

Binary Relationship

- Let's look at Likes, listing all pairs of (*x*,*y*) where person *x* Likes product *y*, and the associated ER diagram
- First listing the relationship informally (we omit article "a"):
  - Chee likes computer
  - Chee likes monitor
  - Lakshmi likes computer
  - Marsha likes computer
- Note
  - Not every person has to Like a product
  - Not every product has to have a person who Likes it (informally, be Liked)
  - A person can Like many products
  - A product can have many person each of whom Likes it



Relationships

- Formally we say that R is a ***relationship*** among (not necessarily distinct) entity sets E_1, E_2, …, E_n if and only if R is a subset of $E_1 \times E_2 \times \ldots \times E_n$ (Cartesian product)
- In our example above:
  - $n = 2$
  - E_1 = {Chee, Lakshmi, Marsha, Michael, Jinyang}
  - E_2 = {computer, monitor, printer}
  - $E_1 \times E_2$ = { (Chee,computer), (Chee,monitor), (Chee,printer), (Lakshmi,computer), (Lakshmi,monitor), (Lakshmi,printer), (Marsha,computer), (Marsha,monitor), (Marsha,printer), (Michael,computer), (Michael,monitor), (Michael,printer), (Jinyang,computer), (Jinyang,monitor), (Jinyang,printer) }
  - R = { (Chee,computer), (Chee,monitor), (Lakshmi,computer), (Marsha,monitor) }
- R is a set (unordered, as every set) of ordered tuples, or sequences (here of length two, that is pairs)

Relationships

- Let us elaborate
- $E_1 \times E_2$ was the "universe"
  - It listed all possible pairs of a person liking a product
- At every instance of time, in general only some of these pairs corresponded to the "actual state of the universe"; *R* was the set of such pairs

Representing Relationships

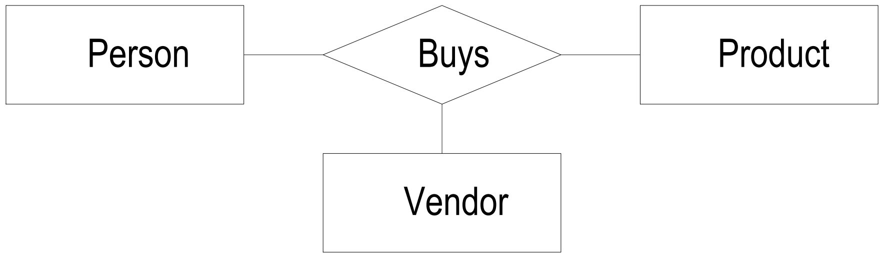
- Ultimately, we will store (most) relationships as tables
- So, our example for Likes could be

Likes	Name	Type
	Chee	Computer
	Chee	Monitor
	Lakshmi	Computer
	Marsha	Monitor

- And we identify the “participating” entities using their primary keys

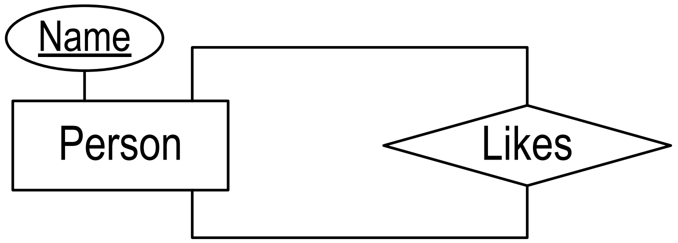
Ternary Relationship

- Let’s look at Buys listing all tuples of (*x*,*y*,*z*) where person *x* Buys product *y* from vendor *z*
- Let us just state it informally:
  - Chee buys computer from IBM
  - Chee buys computer from Dell
  - Lakshmi buys computer from Dell
  - Lakshmi buys monitor from Apple
  - Chee buys monitor from IBM
  - Marsha buys computer from IBM
  - Marsha buys monitor from Dell



Relationship With Nondistinct Entity Sets

- Let's look at Likes, listing all pairs of (*x*,*y*) where person *x* Likes person *y*
- Relationships in which a set participates more than once are sometimes called ***recursive***
- Let us just state it informally
  - Chee likes Lakshmi
  - Chee likes Marsha
  - Lakshmi likes Marsha
  - Lakshmi likes Michael
  - Lakshmi likes Lakshmi
  - Marsha likes Lakshmi
- Note that pairs must be ordered to properly specify the relationship, Chee likes Lakshmi, but Lakshmi may not like Chee



Relationship With Nondistinct Entity Sets

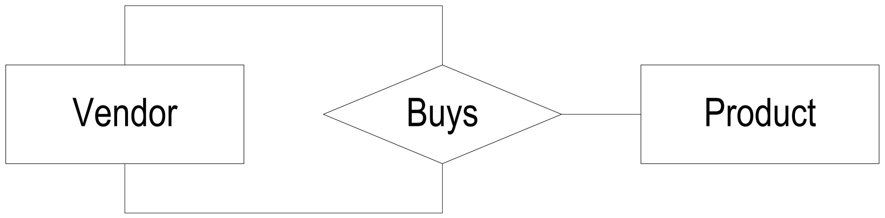
- Again:
  - Chee likes Lakshmi
  - Chee likes Marsha
  - Lakshmi likes Marsha
  - Lakshmi likes Michael
  - Lakshmi likes Lakshmi
  - Marsha likes Lakshmi

- Formally Likes is a subset of the Cartesian product Person × Person, which is the set of all ordered pairs of the form (person,person)
- Likes is the set { (Chee,Lakshmi), (Chee,Marsha), (Lakshmi,Marsha), (Lakshmi,Michael), (Lakshmi,Lakshmi), (Marsha,Lakshmi) }
- Likes is an arbitrary directed graph in which persons serve as vertices and arcs specify who likes whom

Relationship With Nondistinct Entity Sets

- Consider Buys, listing all triples of the form (*x*,*y*,*z*) where vendor *x* Buys product *y* from vendor *z*

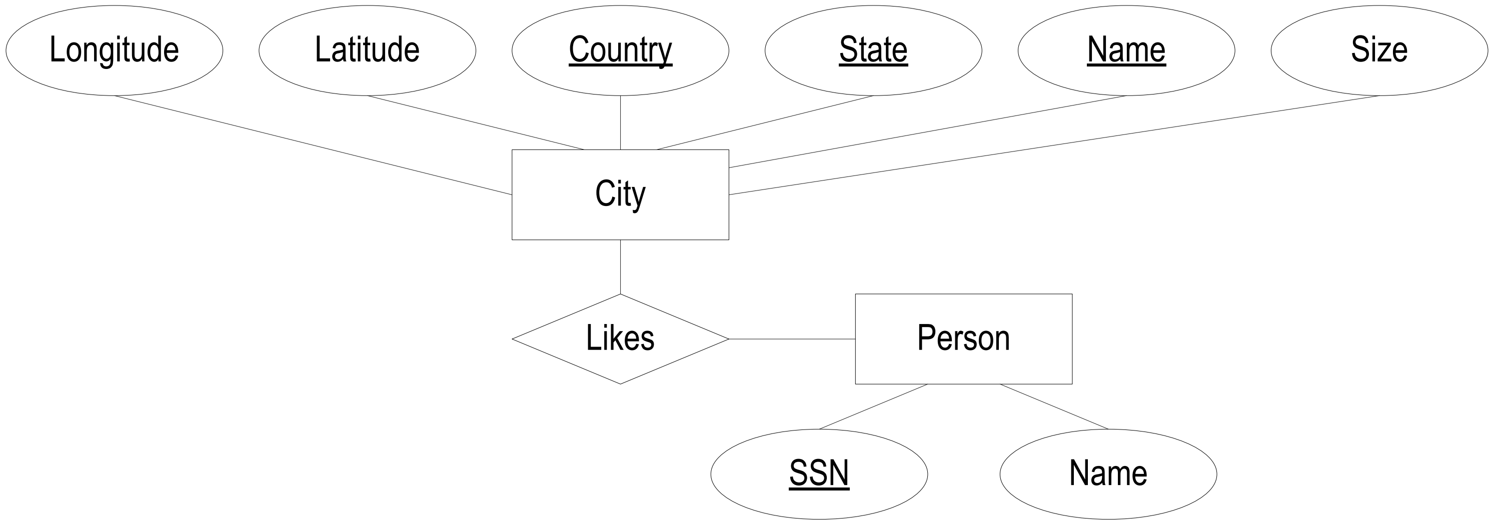
- A typical tuple might be (Dell,printer,HP), meaning that Dell buys a printer from HP



ER Diagrams

ER Diagrams

- To show which entities participate in which relationships, and which attributes participate in which entities, we draw line segments between:
  - Entities and relationships they participate in
  - Attributes and entities they belong to
- We also underline the attributes of the primary key for each entity that has a primary key
- Below is a simple ER diagram (with a simpler Person than we had before):



Further Refinements to the ER Model

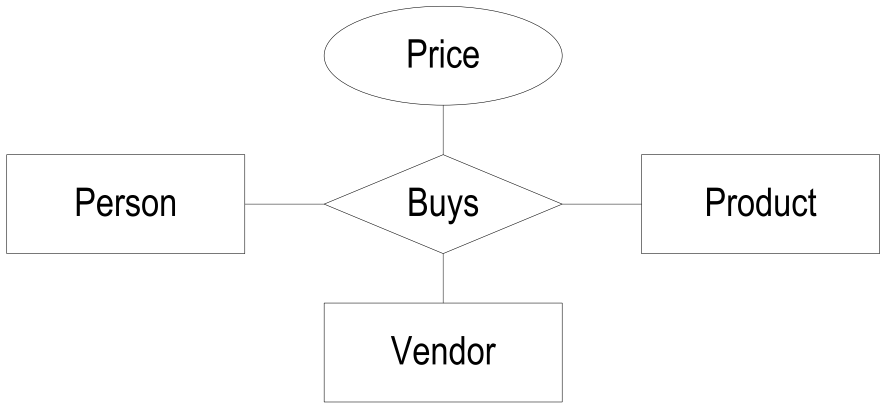
- We will present, in steps, further refinements to the model and associated diagrams
- The previous modeling concepts and the ones that follow are needed for producing a data base design that models a given application well
- We will then put it together in a larger comprehensive example

Relationship With Attributes

- Consider relationship Buys among Person, Vendor, and Product
- We want to specify that a person Buys a product from a vendor at a specific price
- Price
  - is not a property of a vendor, because different products may be sold by the same vendor at different prices
  - is not a property of a product, because different vendors may sell the same product at different prices
  - is not a property of a person, because different products may be bought by the same person at different prices

Relationship With Attributes

- So Price is really an attribute of the relationship Buys
- For each tuple (person, product, vendor) there is a value of price



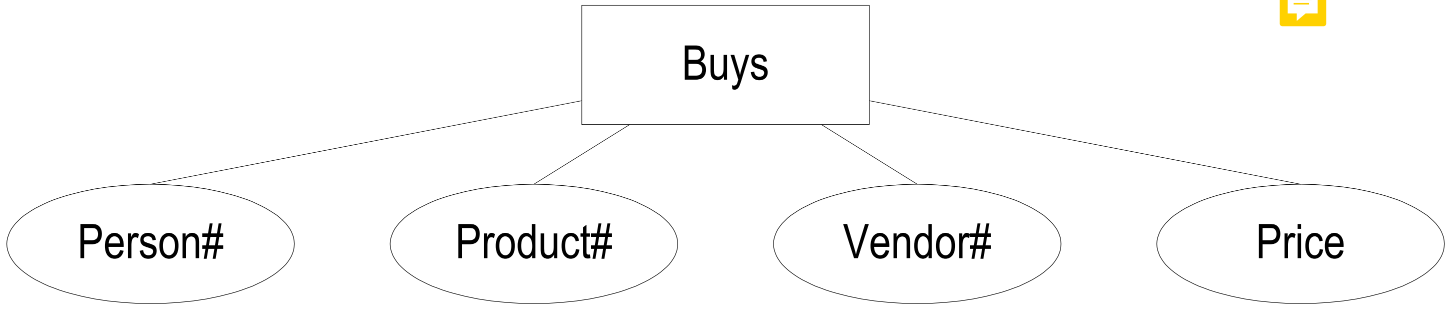
Entity Versus Attribute

- Entities can model situations that attributes cannot model naturally
- Entities can
  - Participate in relationships
  - Have attributes
- Attributes cannot do any of these (it is true that a composite attribute an attribute may have attributes, but that is a composition relationship)

- Let us look at a “fleshed out example” for possible alternative modeling of Buys

Other Choices for Modeling Buys

- What do you think of this design?



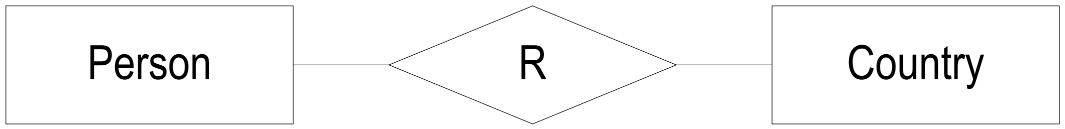
- Not if we want to model something about a person, such as the date of birth of a person or whom a person likes
- These require a person to have an attribute (date of birth) and enter into a relationship (with other persons)
- And we cannot model this situation if person is an attribute of Buy
- Similarly, for product and vendor

Functionality (As in Mathematical Functions)

Binary Relationships and Their Functionality

- "***Functionality***" means here that sometimes a relation is a ***partial function***, that is
  - A function
  - But maybe not necessarily defined on the whole domain

    For example, if *x* varies over all real numbers, the function *y* = 1 / *x* is not defined for *x* = 0
- Consider a relationship R between two entity sets A, B.
- We will look at examples where A is the set of persons and B is the set of all countries



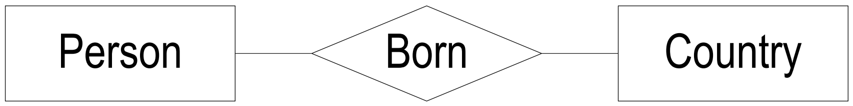
- We will be making some simple assumptions about persons and countries, which we list when relevant

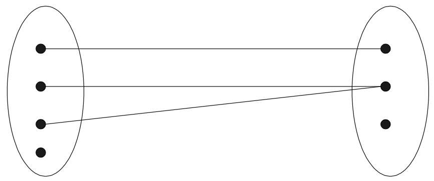
Binary Relationships and Their Functionality

- Relationship R is called ***many to one*** from A to B if and only if for each element of A there exists at most one element of B related to it
  - Example: R is Born (in)

    Each person was born in at most one country (maybe not in a country but on a ship in the middle of an ocean), so there is a ***partial function*** from set Person to set Country

    Maybe nobody was born in some country because the country never existed until now





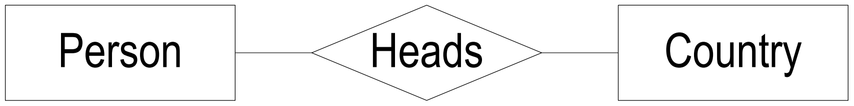
- The picture on the right describes the universe of four persons and three countries, with lines indicating which person was born in which country
  - We will have similar diagrams for other examples

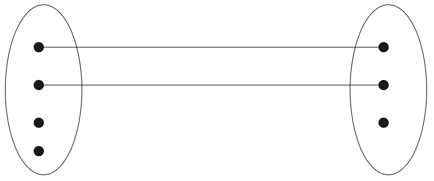
Binary Relationships and Their Functionality

- The relationship R is called ***one to one*** between A and B if and only if for each element of A there exists at most one element of B related to it and for each element of B there exists at most one element of A related to it
  - Example: R is Heads

    Each Person is a Head (President, Queen, etc.) of at most one country (not true in reality)

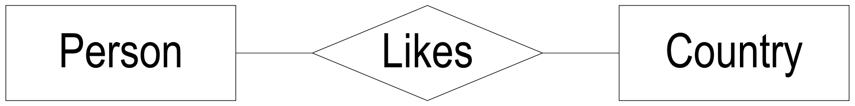
    Each country has at most one head (maybe the queen died and it is not clear who will be the monarch next)
- Another example of 1 to 1 (in New York state): legally_married_to

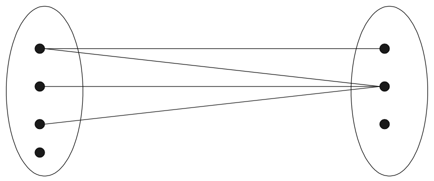




Binary Relationships and Their Functionality

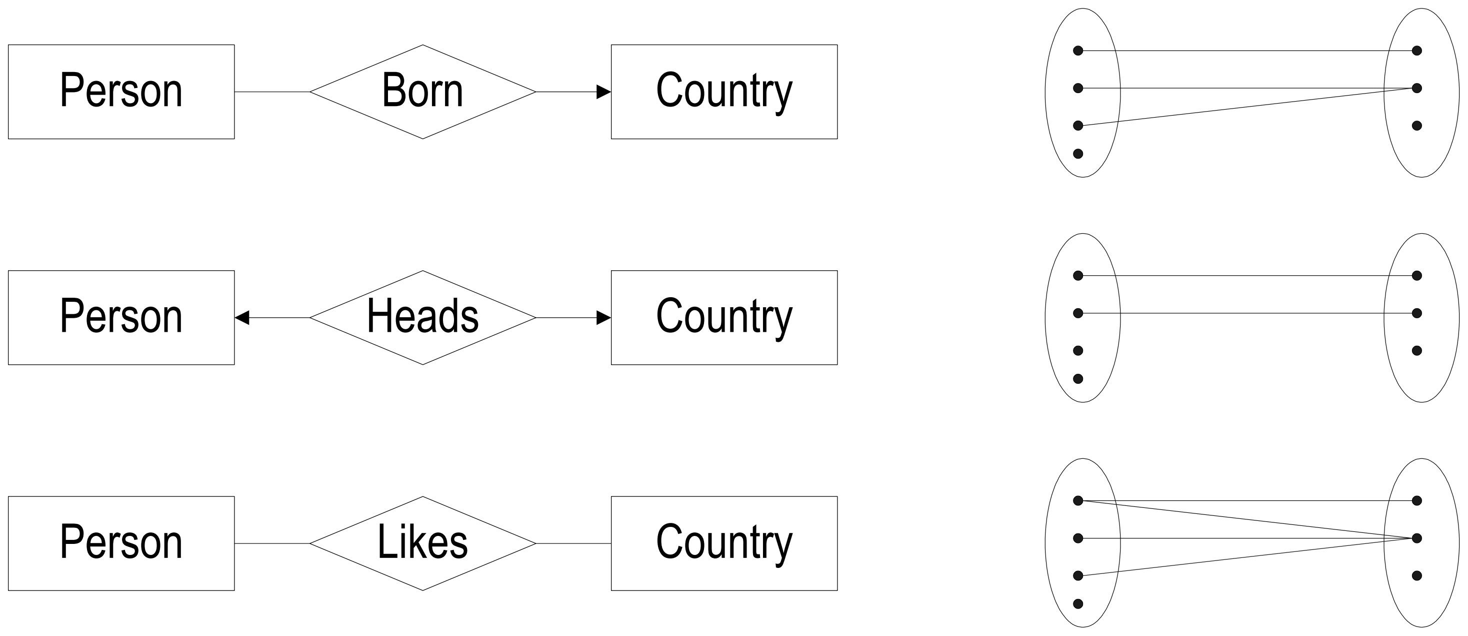
- The relationship is called ***many to many*** between A and B, if an element of A can be linked to many in B and vice versa.
  - Example: R is “Likes”





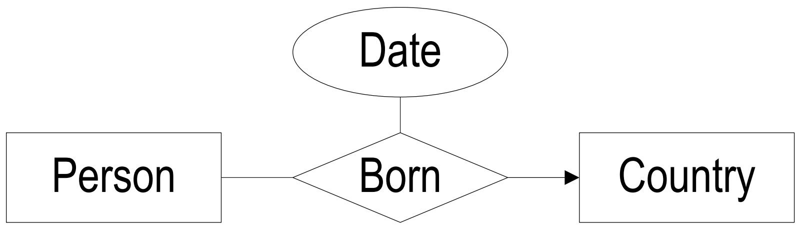
Binary Relationships and Their Functionality

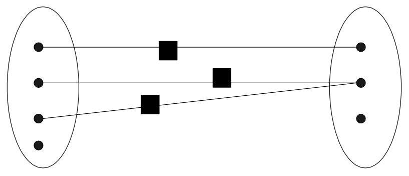
- We have in effect considered the concepts of partial functions of one variable.
  - The first two examples were ***partial functions*** (to remind, "partial" means that the function does not have to be defined on all elements of its domain)
  - The last example was not a function
- Pictorially, functionality for binary relationships can be shown by drawing an arc head in the direction to the "one"

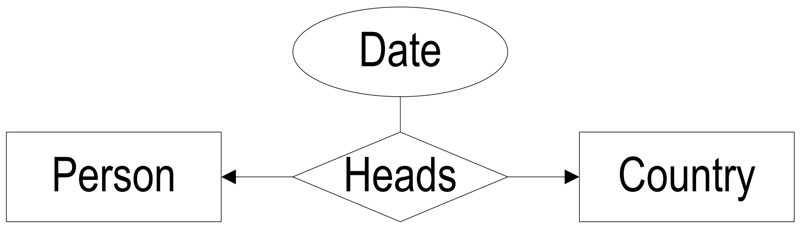


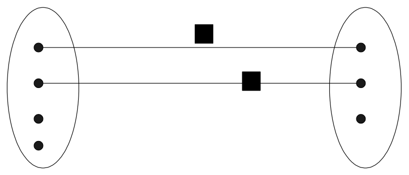
Binary Relationships and Their Functionality

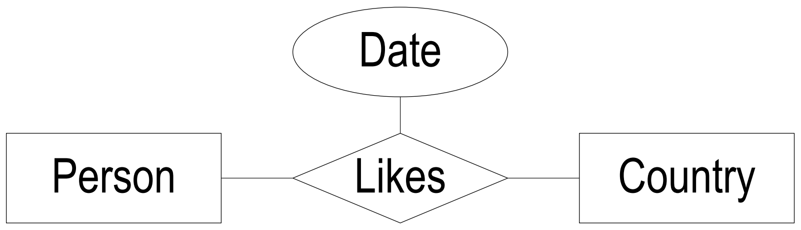
- How about the properties of a relationship?
- Date: when a person and a country in a relationship first entered into the relationship (marked also with black square)

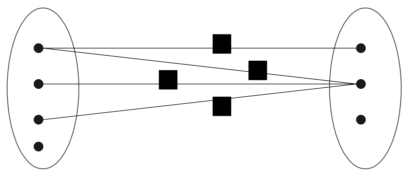






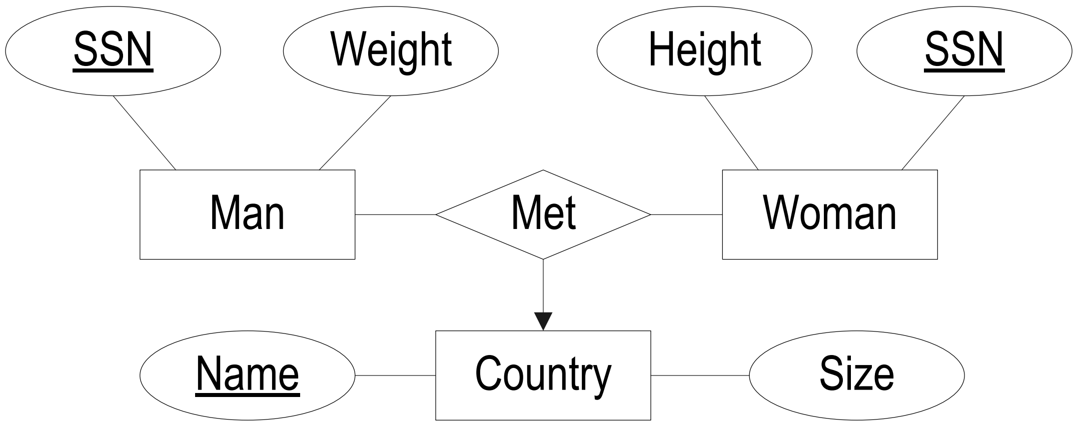






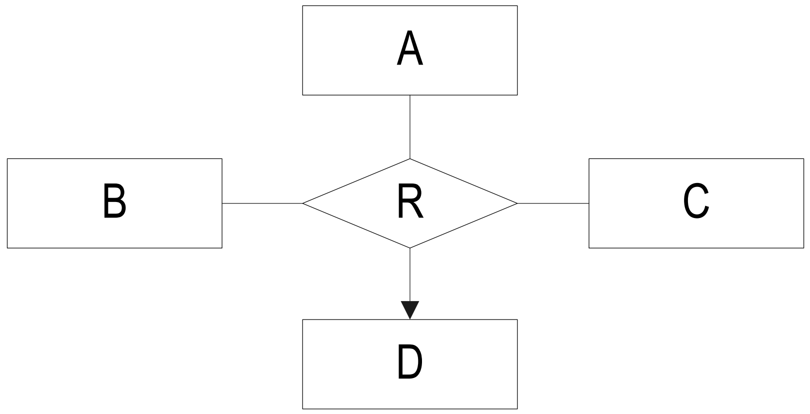
Non-Binary Many-To-One Relationships

- There could also be relationships that are not binary but still many-to-one
- In the example below, a man and a woman could have met in at most one country
- This is indicated by an arrow



Non-Binary Many-To-One Relationships

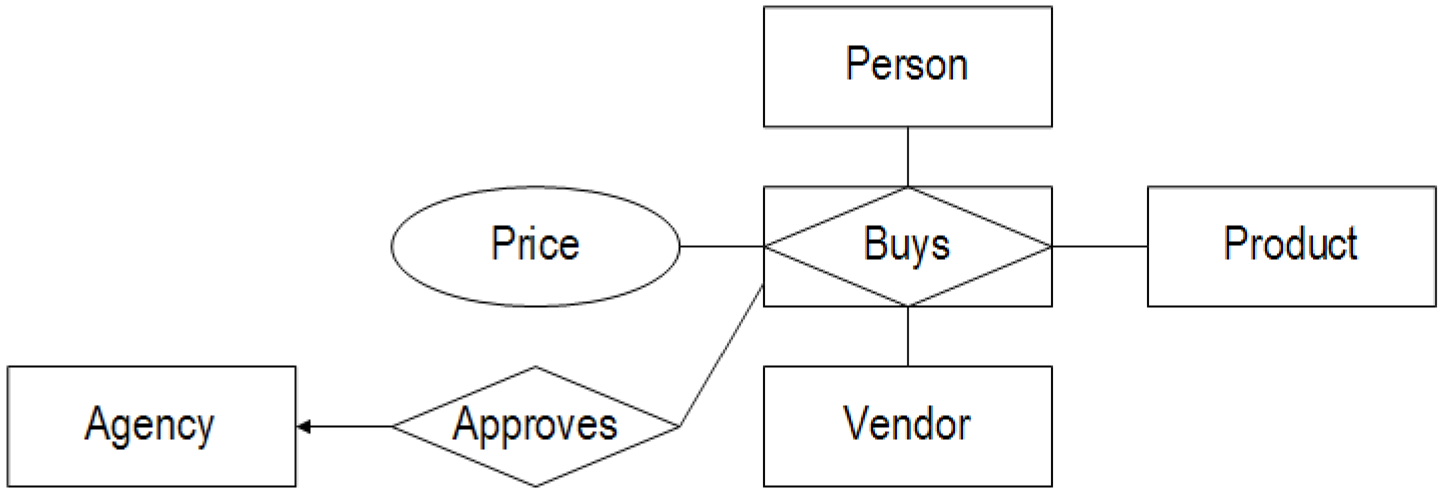
- Below an example of modeling when for every triple of entities (a,b,c) from A, B, and C, there is at most one entity d in D



Aggregation

Aggregation: Relationships as Entities

- It is sometimes natural to consider relationships as if they were entities.
- This will allow us to let relationships participate in other “higher order” relationships
- Here each instance of Buys needs to be approved by (at most) one agency
- Relationship is “made into” an entity by putting it into a ***rectangle***; note that the edge between Buys and Approves touches the Buys rectangle but not the Buys diamond, to make sure that we are not confused



Weak Entities

Strong and Weak Entities

- We have two entity sets
  - Man
  - Woman
- Woman has a single attribute, SSN
- Let us defer discussion of attributes of Man
- A woman has 5 sons, among them Aarav and Richard, neither of the two is her eldest son and she writes the following in her will:

My SSN is 123-45-6789 and I leave $100 to my eldest son and $200 to my son Aarav and $300 to my son Wei …

- How do we identify these 3 men from among all the men in the universe?

Strong and Weak Entities

- A ***strong entity*** (set): Its elements can be identified by the values of their attributes, that is, it has a (primary) key made of its attributes

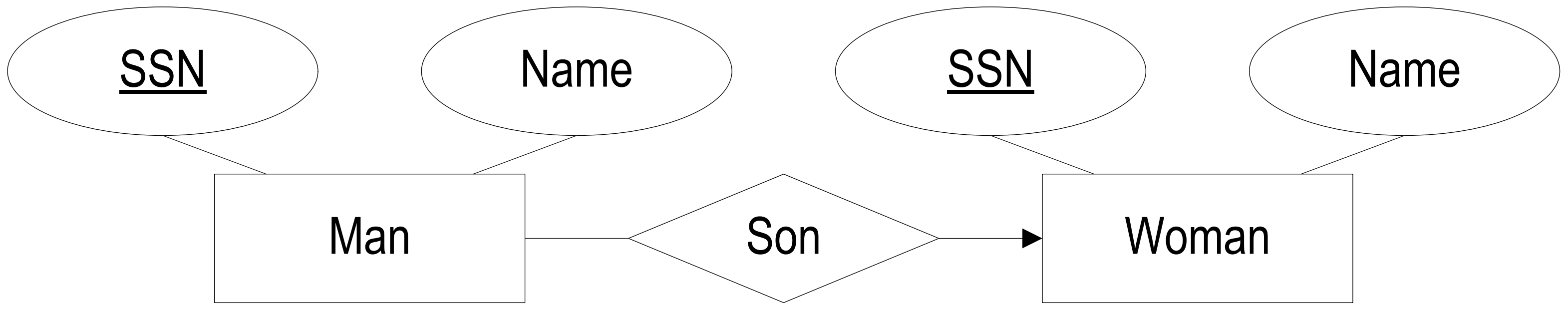
  Tacitly, we assumed only such entities so far

- ***A weak entity*** (set): Its elements cannot be identified by the values of their attributes: no primary key can be made from its own attributes

  Such entities can be identified by a combination of their attributes and the relationship they have with an entity in another entity set

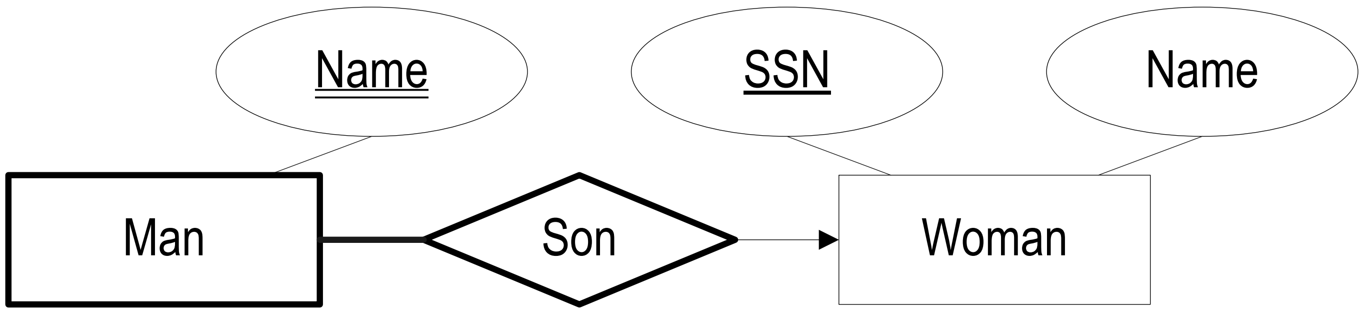
Man as a Strong Entity

- Most entities are ***strong***: a specific entity can be distinguished from other entities based on the values of its attributes
- We assume that every person has his/her own SSN
- Woman is a strong entity as we can identify a specific woman based on her attributes. She has a primary key: her own SSN
- Man is a strong entity as we can identify a specific man based on his attributes. He has a primary key: his own SSN



Man as a Weak Entity

- We assume that women are given SSNs
- Men are not given SSNs; they have first names only, but for each we know who the mother is (that is, we know the SSN of the man’s mother)
- Man is a ***weak*** entity as we cannot identify a specific man based on his own attributes and **this is indicated by thick lines around it**
- Many women could have a son named Bob, so there are many men named Bob
- However, if a woman never gives the same name to more than one of her sons, a man can be identified by his name (note double underline) and by his mother’s SSN



Man as a Weak Entity

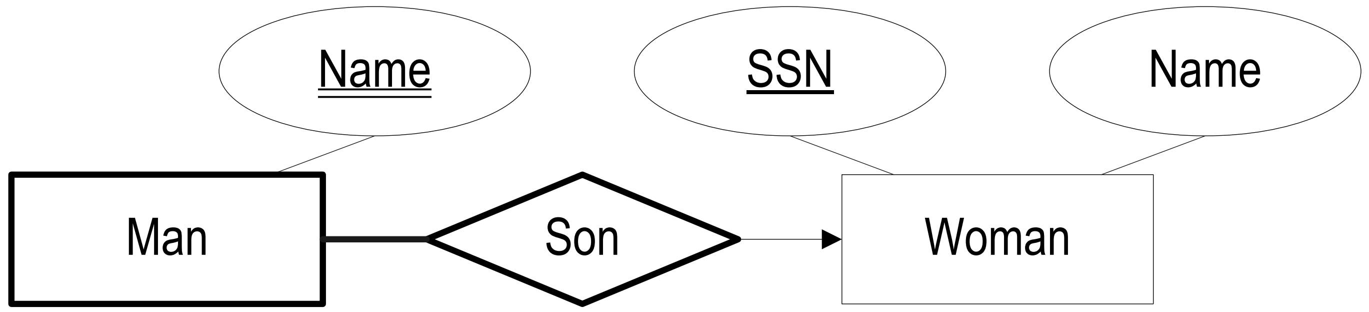
- We could have the following situation of two mothers: one with two sons, and one with three sons, when we gave people also heights in inches (just to have additional attributes that are not necessary for identification)

- SSN: 070-43-1234, height: 65
  - Name: Bob, height 35
  - Name: Dao-I, height 35

- SSN: 056-35-4321, height 68
  - Name: Bob, height 35
  - Name: Mohammed, height 45
  - Name: Vijay, height 84

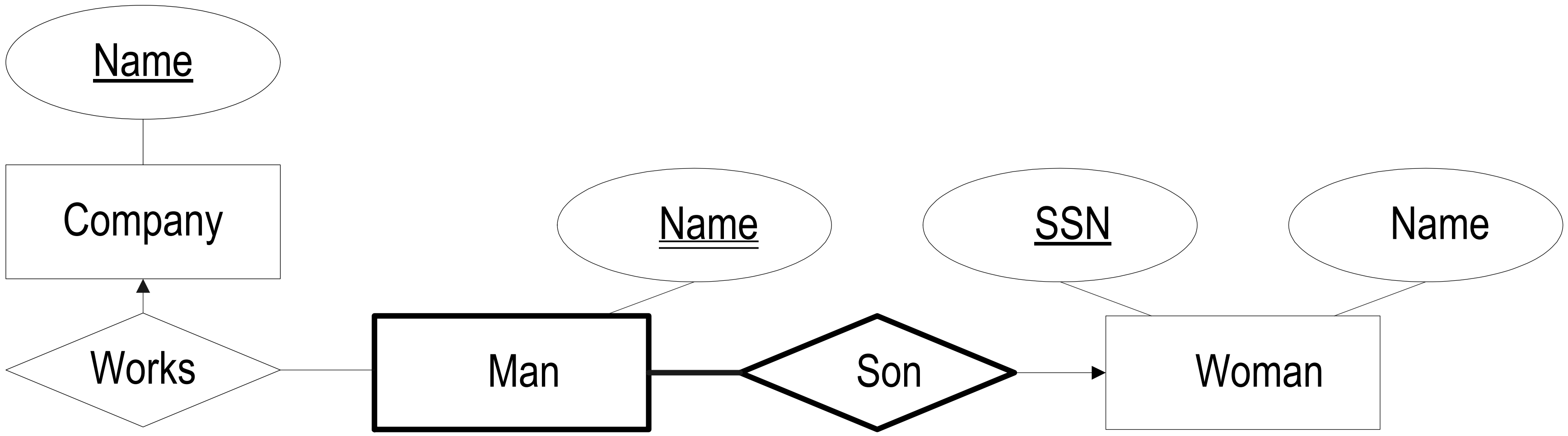
Man as a Weak Entity

- Assuming that a woman does not have more than one son with a specific first name
- Name becomes a ***discriminant***
- Man can be identified by the combination of:
  - The Woman to whom he is related under the Son relation. This is indicated by thick lines around Son (it is weak). Thick line connecting Man to Son indicates the relationship is total on Man (every Man participates) and used for identification
  - We add an arrowhead for consistency, although we know that semantically the relationship must be many-to-one
  - His Name. His Name is now a ***discriminant***; **this is indicated by double underline**



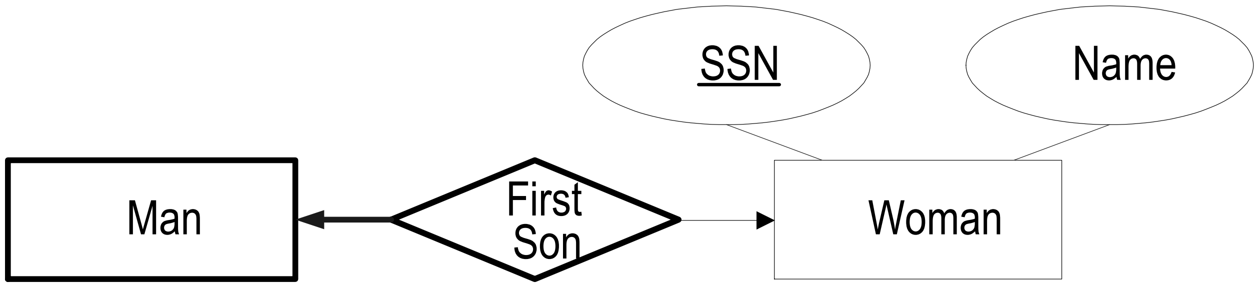
Man as a Weak Entity

- We need to specify for a weak entity **through which relationship it is identified; this is done by using thick lines**
- Otherwise we do not know whether Man is identified through Son or through Works



Man as a Weak Entity

- Sometimes a discriminant is not needed
- We are only interested in men who happen to be first sons of women
- Every Woman has at most one First Son
- So we do not need to have Name for Man (if we do not want to store it, but if we do store it, it is not a discriminant)



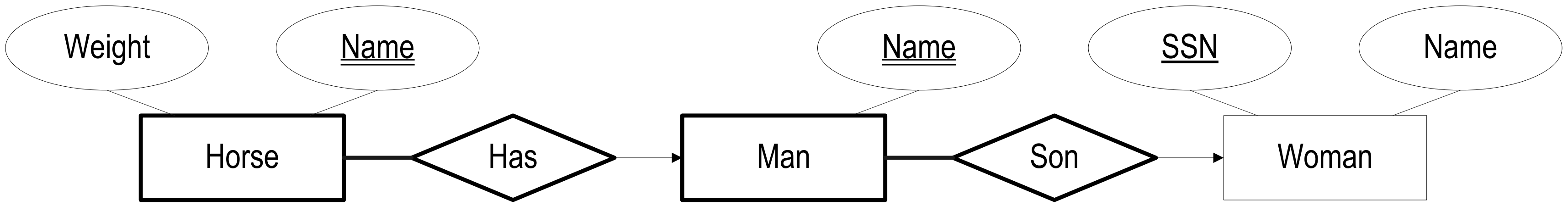
- Note an arrowhead to the left: each Woman has at most one First Son

Man as a Weak Entity

- In general, more than one attribute may be needed as a discriminant
- For example, let us say that man has both first name and middle name
- A mother may give two sons the same first name or the same middle name
- A mother will never give two sons the same first name and the same middle name
- The pair (first name, middle name) together form a discriminant

From Weaker to Stronger

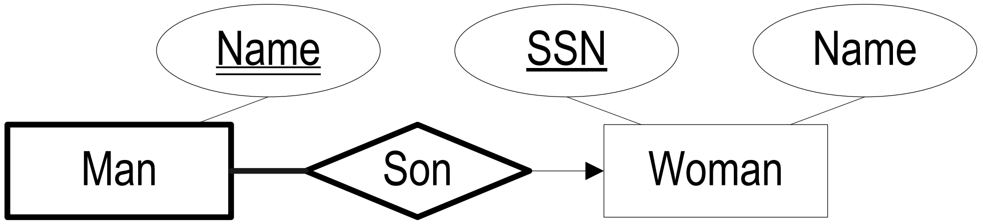
- There can be several levels of “weakness”
- Here we can say that a horse named “Speedy” belongs to Bob, whose mother is a woman with SSN 072-45-9867



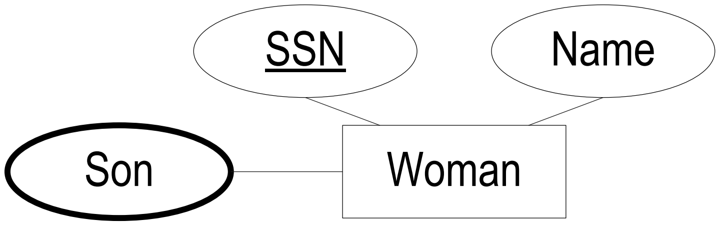
- A woman can have several sons, each of whom can have several horses

Weak Entity or Multivalued Attribute?

- We had



- If nothing in the description of the application leads us to believe that Man could have its own attributes or could participate in relationships (other than with his mother), it might be more natural to implement the fragment of the database using a multivalued attribute
  - That’s what you should do in all your relevant assignments



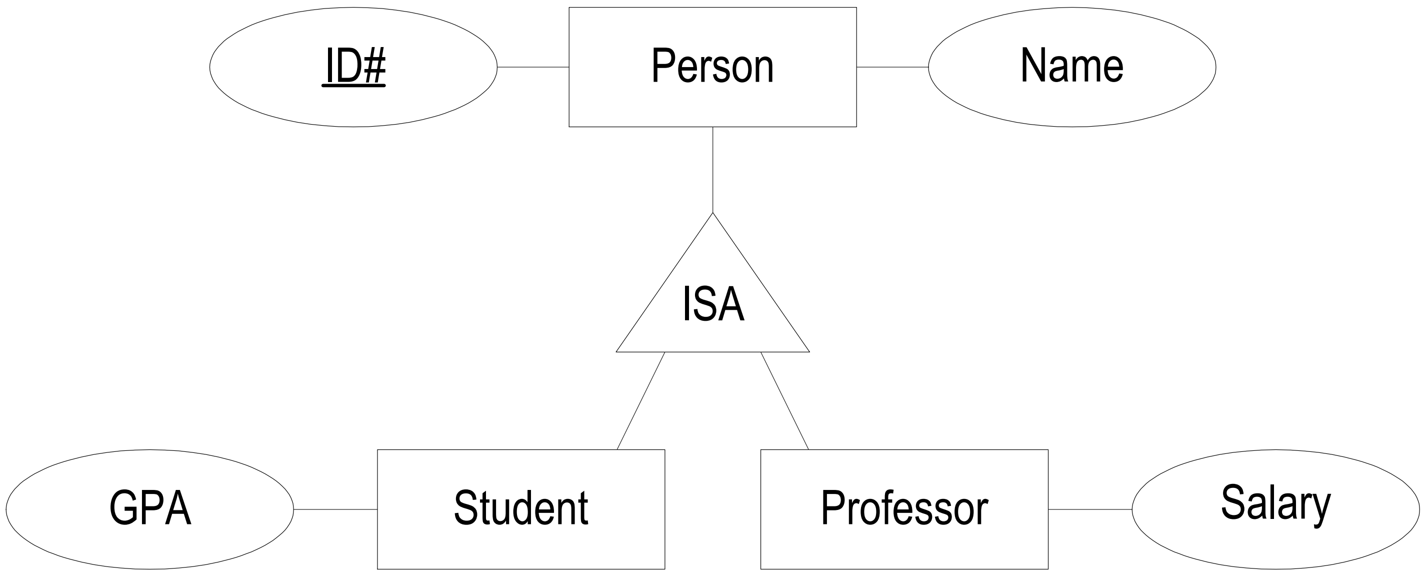
ISA

The ISA Relationship

- For certain purposes, we consider subsets of an entity set
- The subset relationship between the set and its subset is called ***ISA***, meaning "***is a***"
- Elements of the subset, of course, have all the attributes and relationships as the elements of the set: they are in the "original" entity set
- In addition, they may participate in relationships and have attributes that make sense for them
  - But those relationships and attributes do not make sense for every entity in the "original" entity set
- ISA is indicated by a triangle
- The elements of the subset are weak entities, as we will note next

The ISA Relationship

- Example: A subset that has an attribute that the original set does not have
- We look at all the persons associated with a university
- Some of the persons happen to be professors and some of the persons happen to be students



The ISA Relationship

- Professor is a weak entity because it cannot be identified by its own attributes (here: Salary)
- Student is a weak entity because it cannot be identified by its own attributes (here: GPA)
- They do not have discriminants, they are identified by the primary key of the strong entity (Person)
- We do not need to put Student and Professor in a rectangle with thick lines as being “under” ISA automatically tells us that they are weak entities
- There may be an attribute of Person (called ***discriminator***) that can be used to determine who is a Student and who is a Professor
  - For a fake example: Professor names start with a A–M and Student names start with N–Z (but note advisement in our department for a real example)
- The set and the subsets are sometimes referred to as ***class*** and ***subclasses***

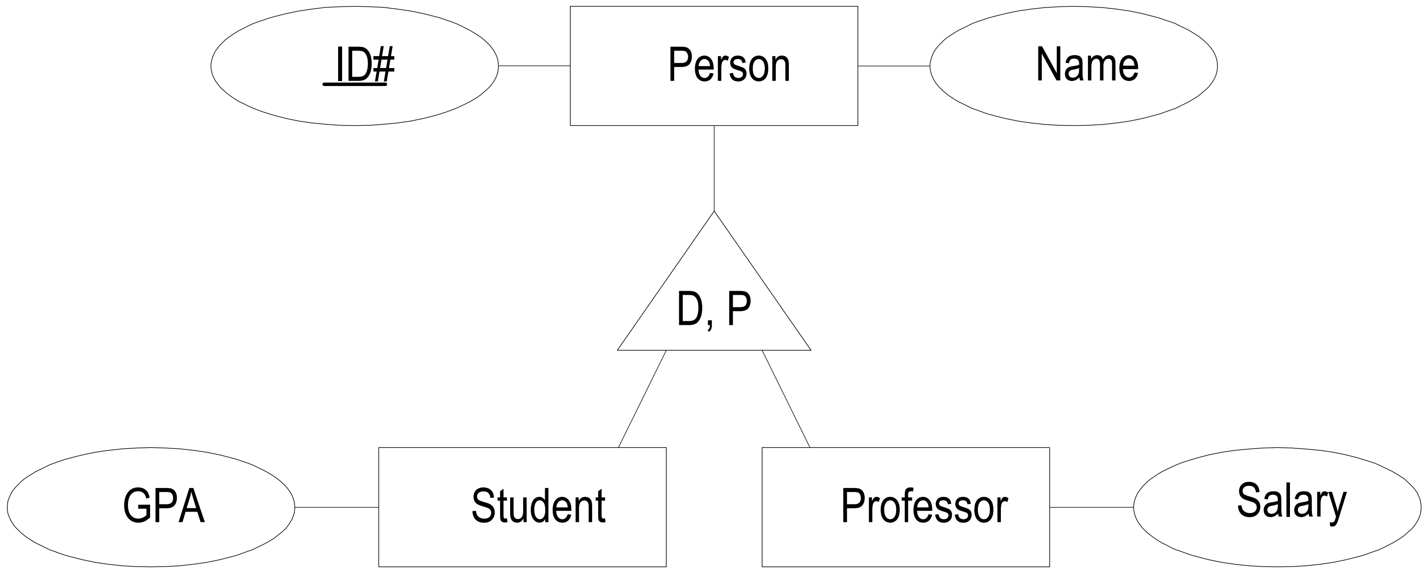
The ISA Relationship

- A person associated with the university (and therefore in our database) can be in general
  - Only a professor
  - Only a student
  - Both a professor and a student
  - Neither a professor nor a student
- A specific ISA could be
  - ***Disjoint***: no entity could be in more than one subclass
  - ***Overlapping***: an entity could be in more than one subclass
  - ***Total*** (also called ***Complete***): every entity has to be in at least one subclass
  - ***Partial***: an entity does not have to be in any subclass

- This could be specified by replacing “ISA” in the diagram by an appropriate letter
- If nothing stated, then no restriction, so effectively O,P

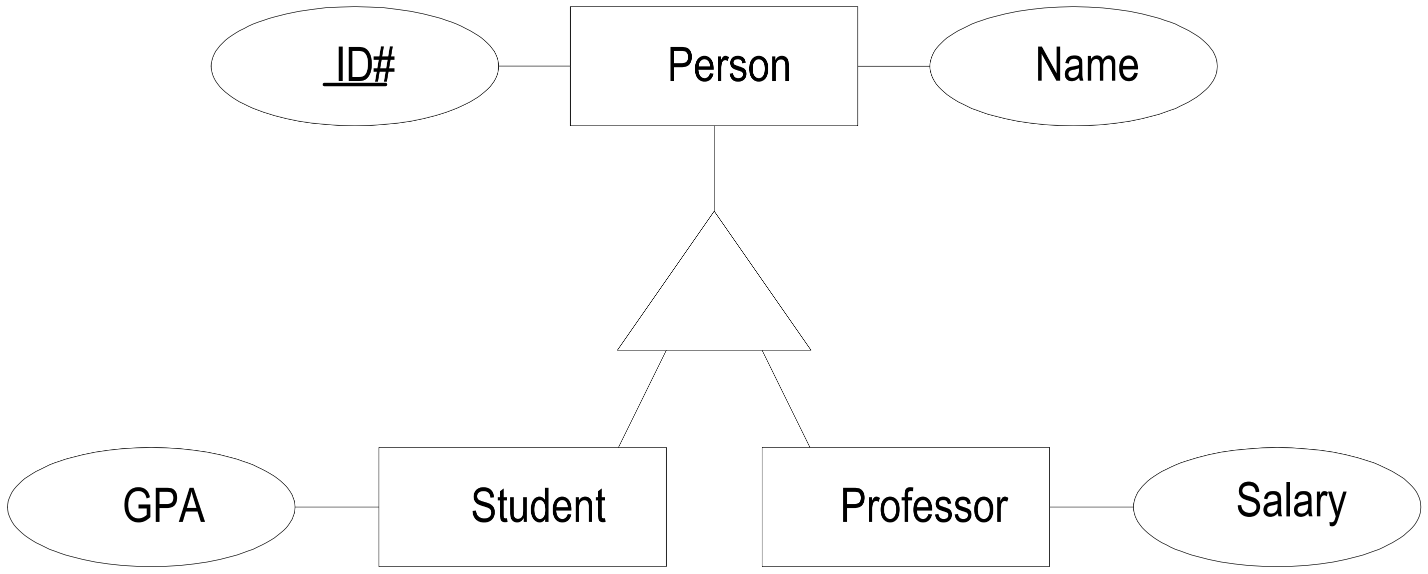
The ISA Relationship (Example)

- Some Persons are Professors
- Some Persons are Students
- Some Persons are neither Professors nor Students
- No Person can be both a Professor and a Student



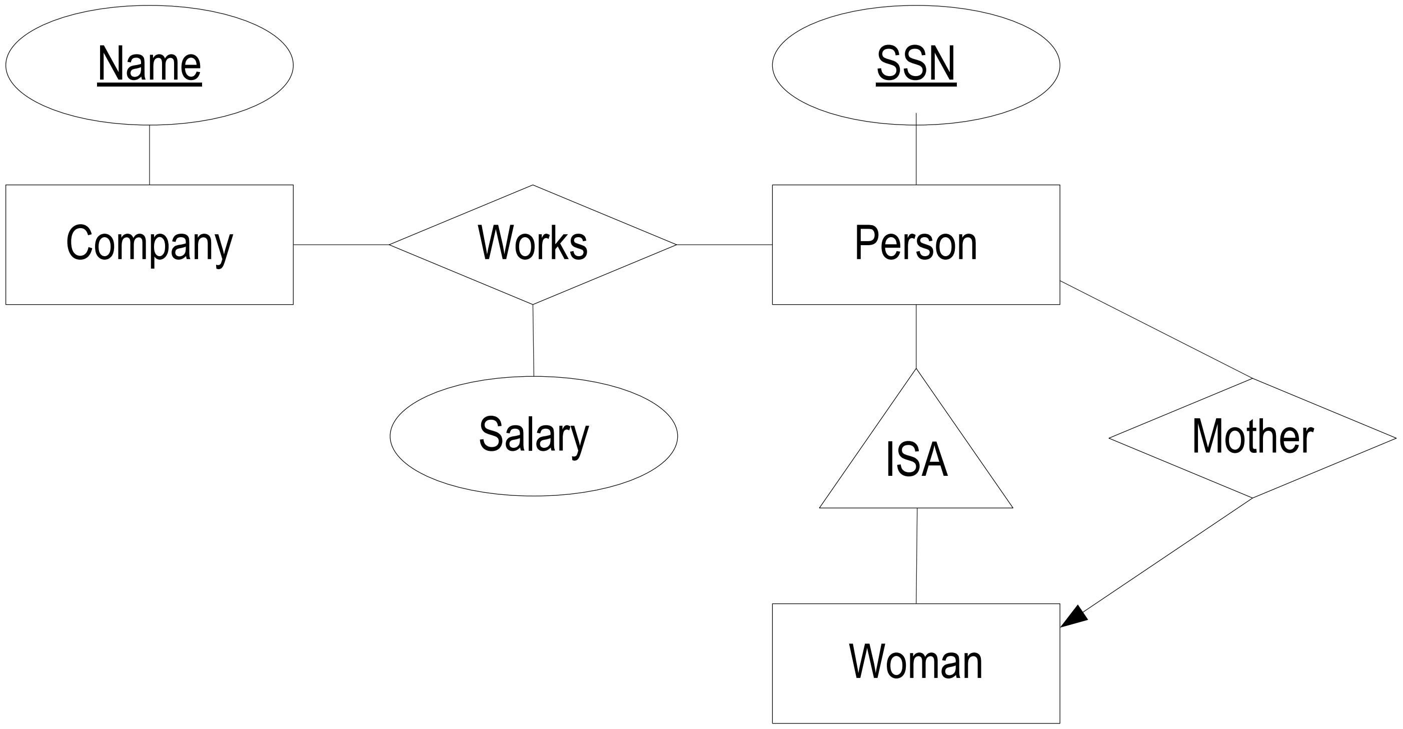
The ISA Relationship (Example)

- If there are no constraints on the ISA relationship, the word “ISA” may be omitted.



The ISA Relationship

- Example: subsets participating in relationships modeling the assumed semantics more clearly (every person has at most one woman who is the biological mother) for people in the database



The ISA Relationship

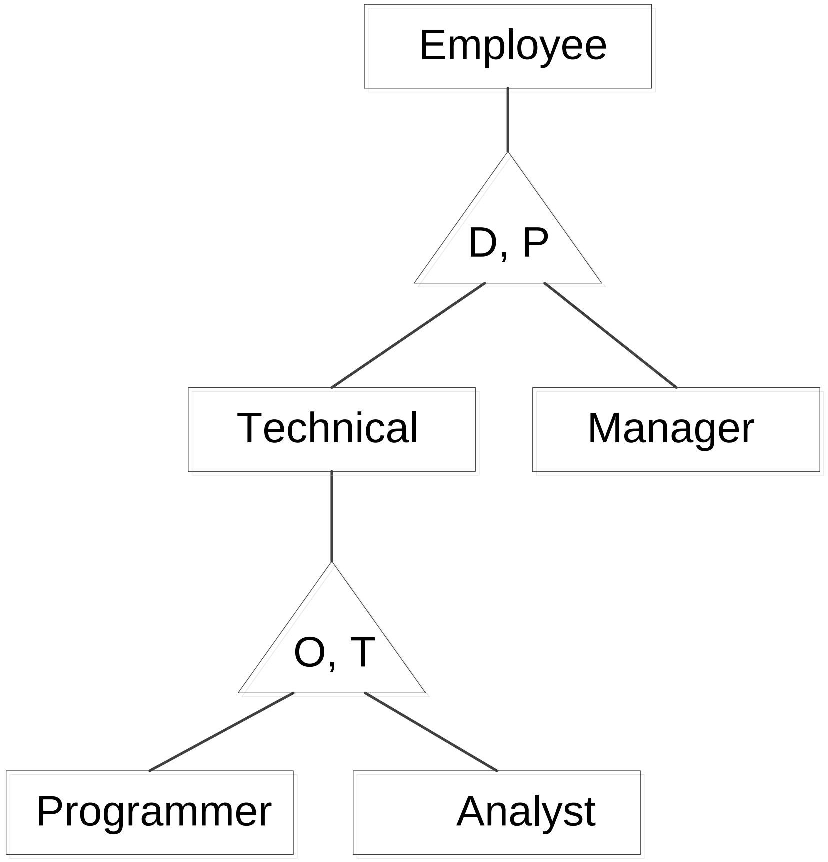
- ISA is really a superclass/subclass relationship
- ISA could be ***specialization***: subsets are made out of the "more basic" set
- ISA could be ***generalization***: a superset is made of "more basic" sets
- Again, the diagram could be annotated to indicate this.

A More Complex Example

- We have several types of employees
  - Managers
  - Programmers
  - Analysts
  - Other
- An employee can be one of the following
  - Manager
  - Programmer and/or Analyst
  - Other
- The 3 sets are disjoint, that is
  - Manager cannot be Programmer or Analyst, or Other
  - Other cannot be Manager, Programmer, or Analyst
- All Employees have some share properties
- It is convenient to group Programmers and Analysts together as they have some shared properties

A Sketch of an ER Diagram With Hierarchical ISAs

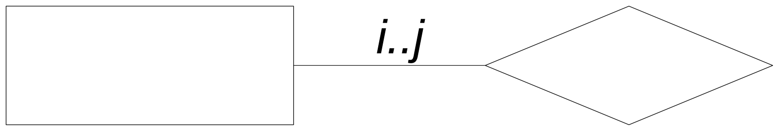
- Mutual exclusion (disjointness) might be enforced by triggers: “e.g. if try to insert in manager, make sure the person is not technical”



Cardinality Constraints

Cardinality Constraints

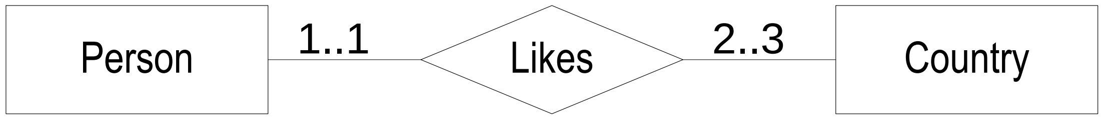
- We can specify how many times each entity from some entity set can participate in some relationship, in every instance of the database
- This generalizes the concept of functionality
- In general we can say that
  - This number is in the interval [*i*,*j*], 0 ≤ *i* ≤ *j*, with *i* and *j* integers, denoted by i..j; or
  - This number is in the interval [*i*, ∞), denoted by i..*
- 0..* means no constraint and can be omitted
- No constraint can also be indicated by not writing out anything

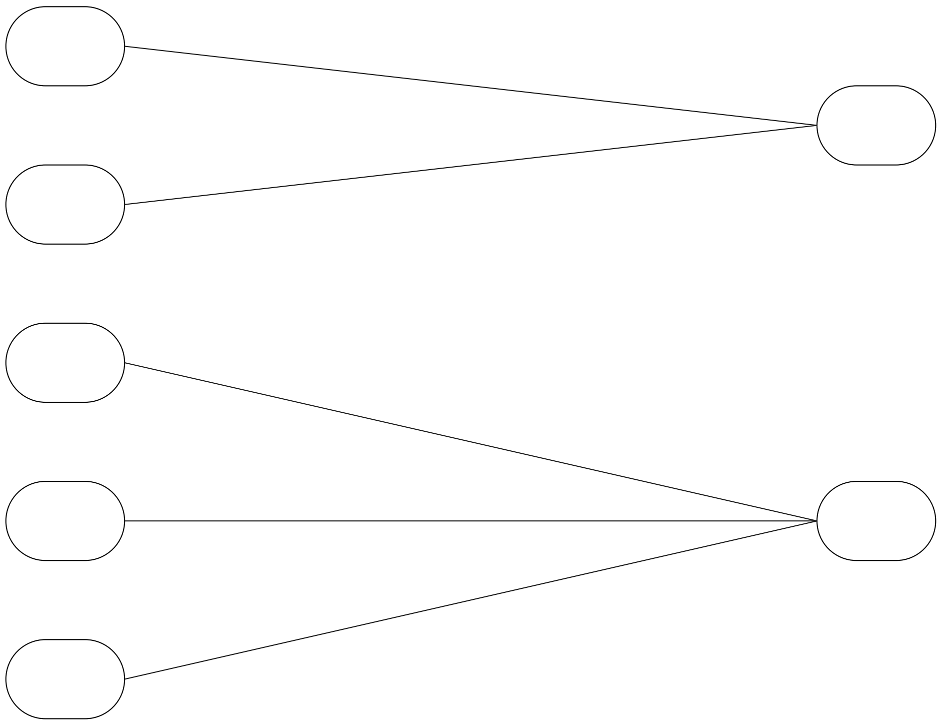


- ***Note the specific convention we will be using, others may use other conventions for cardinality constraints***

Cardinality Constraints

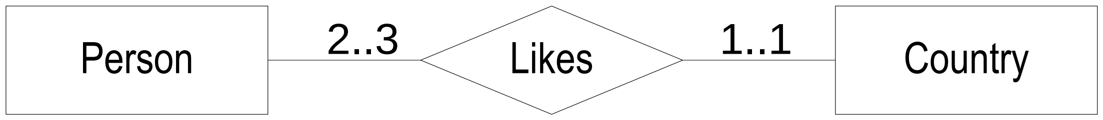
- Every person likes exactly 1 country
- Every country is liked by 2 or 3 persons

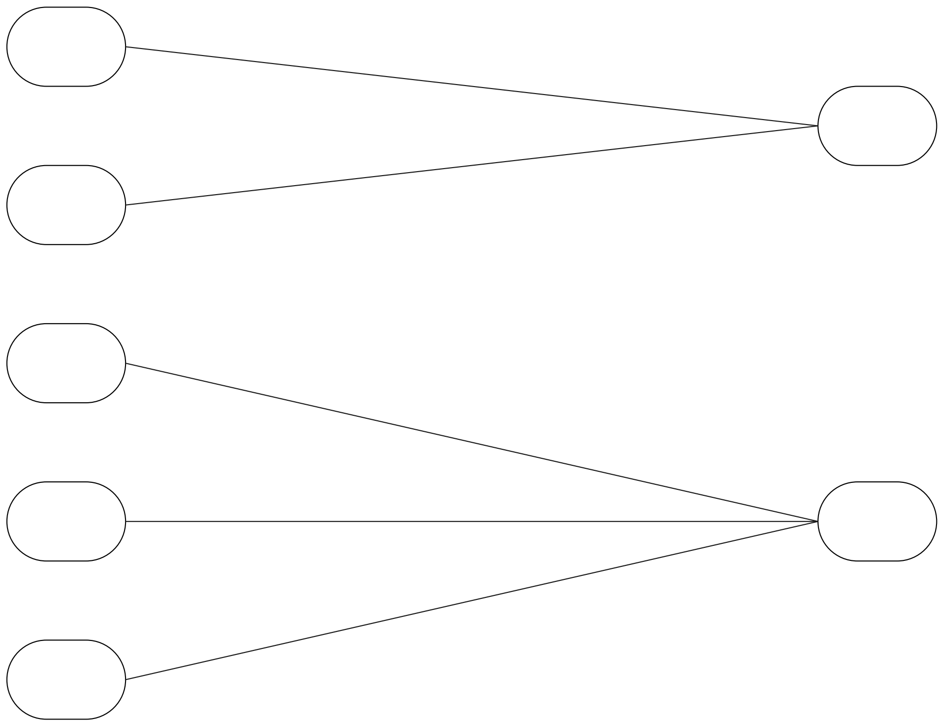




Note on Cardinality Constraints

- Every person likes exactly 1 country
- Every country is liked by 2 or 3 persons
- **Sometimes the opposite convention is used: note where cardinality constraints are written; do not use that**





Cardinality Constraints

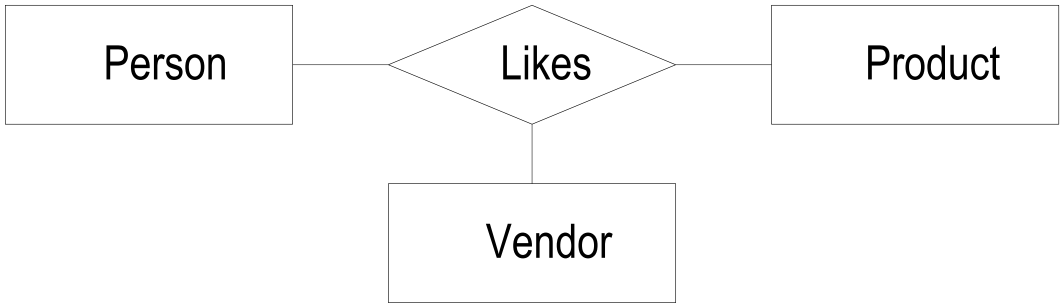
- Returning to an old example without specifying which entities actually exist

Person	Name

Vendor	Company

Product	Type

- We have a relationship: Likes
- A typical “participation” in a relationship would be that Chee, IBM, Computer participate in it



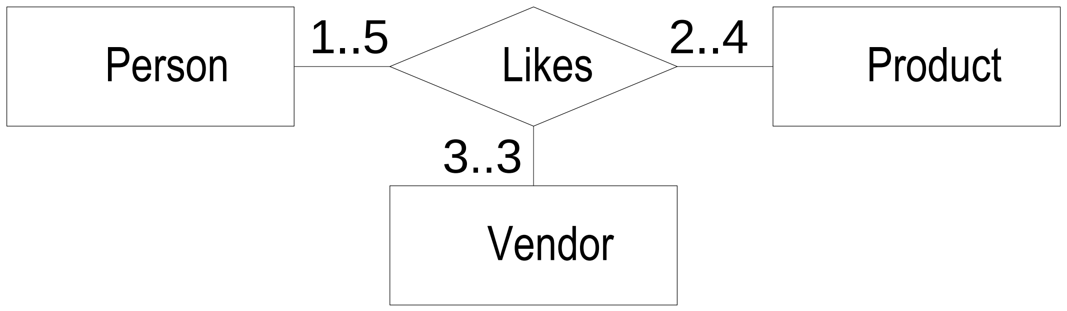
Cardinality Constraints

Person	Name

Vendor	Company

Product	Type

- We want to specify cardinality constraints that ***every instance of the database*** (that is the ***schema***) needs to satisfy
  - Each person participates in between 1 and 5 relationships
  - Each vendor participates in between 3 and 3 (that is exactly 3) relationships
  - Each product participates in between 2 and 4 relationships
- This is indicated as follows:



Cardinality Constraints

- A specific instance of the database

Person	Name
	Chee
	Lakshmi
	Marsha

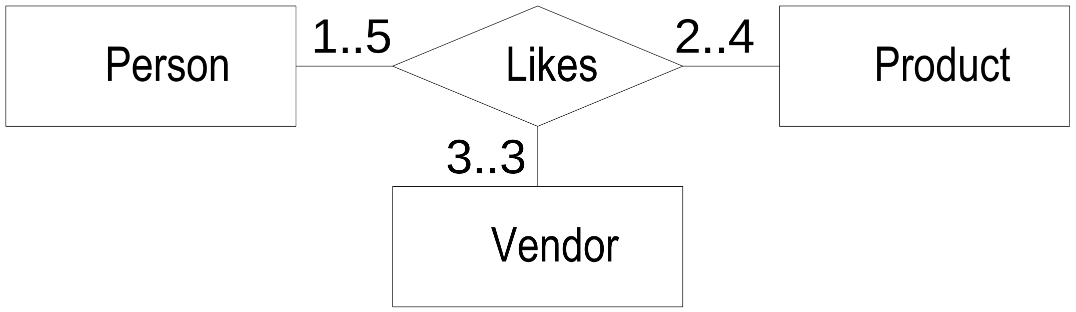
Vendor	Company
	IBM
	Apple

Product	Type
	computer
	monitor

- If we have the following tuples in the relationship

  Chee IBM computer
  Lakshmi Apple monitor
  Marsha Apple computer
  Marsha IBM monitor
  Marsha IBM computer
  Lakshmi Apple computer

- Then, it is true that:

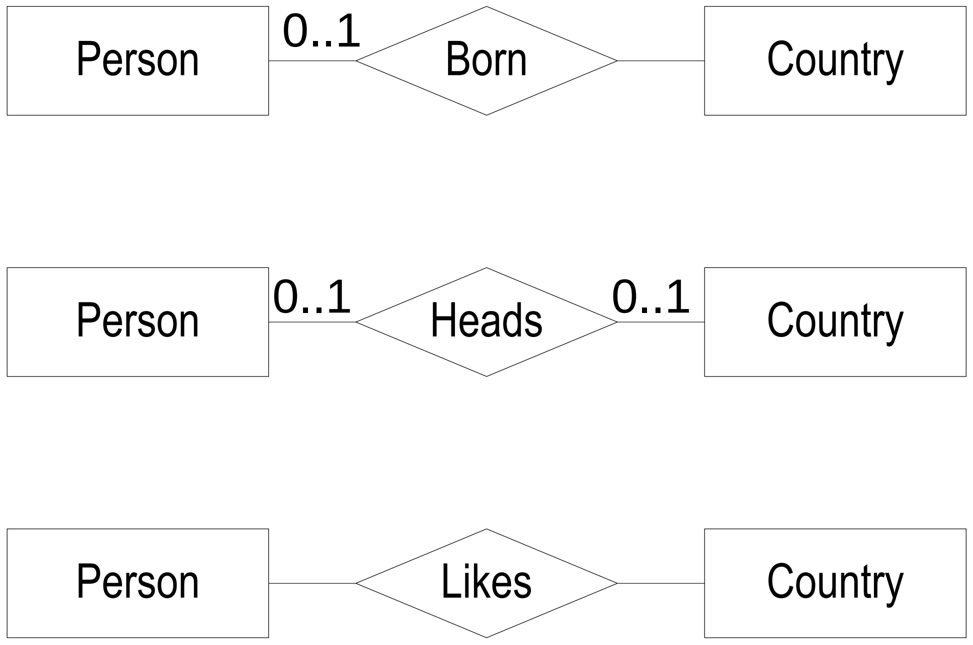


Cardinality Constraints

- Let us confirm that our instance of Likes satisfies the required cardinality constraints
- Person: required between 1 and 5
  - Chee in 1
  - Lakshmi in 2
  - Marsha in 3
- Product: required between 2 and 4
  - Monitor in 2
  - Computer in 4
- Vendor between 3 and 3
  - Apple in 3
  - IBM in 3
- Note that we do not have to have an entity for every possible permitted cardinality value
  - For example, there is no person participating in 4 or 5 tuples

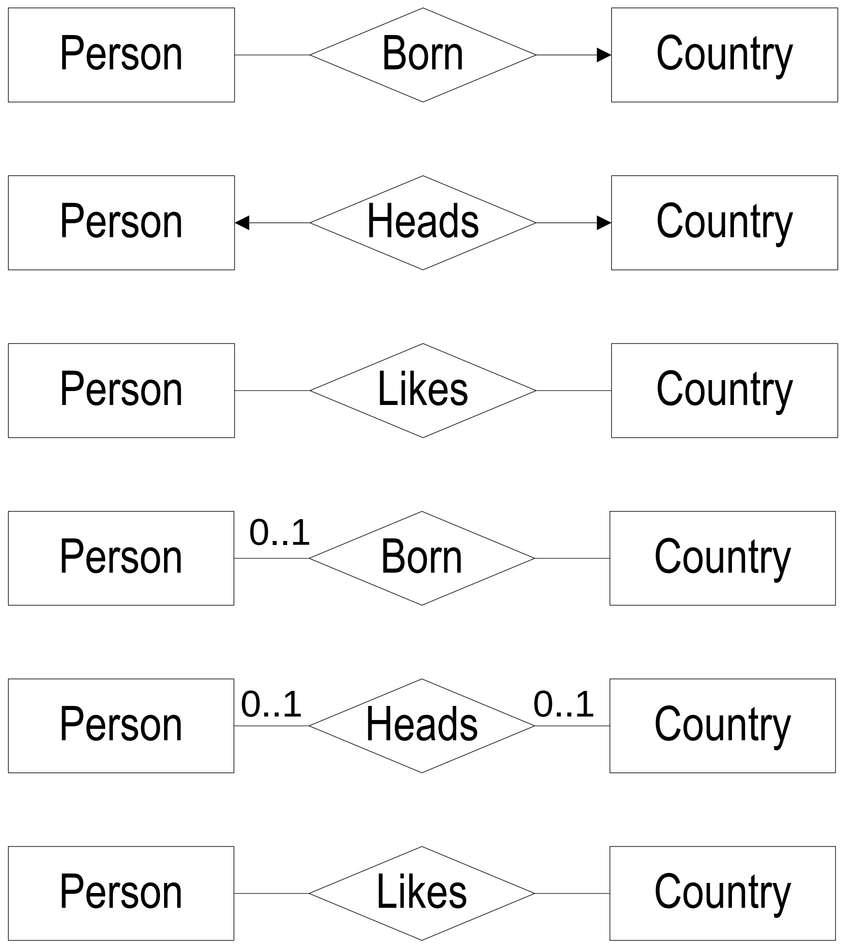
Cardinality Constraints

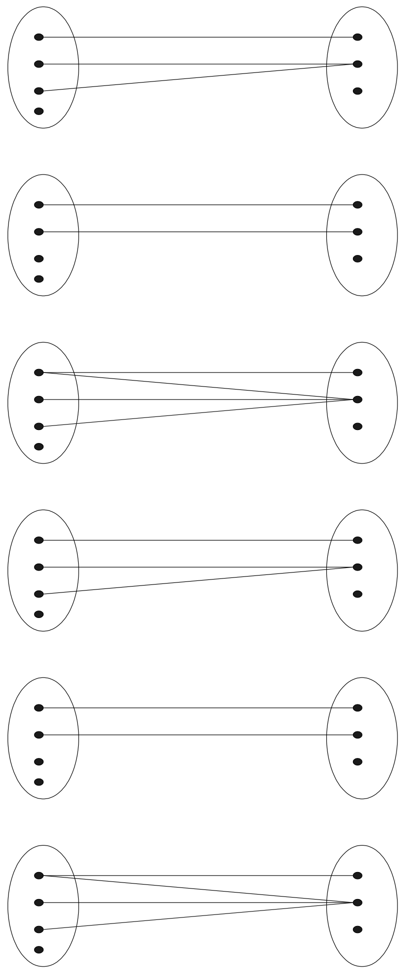
- So we can also have, expressing exactly what we had before



Cardinality Constraints

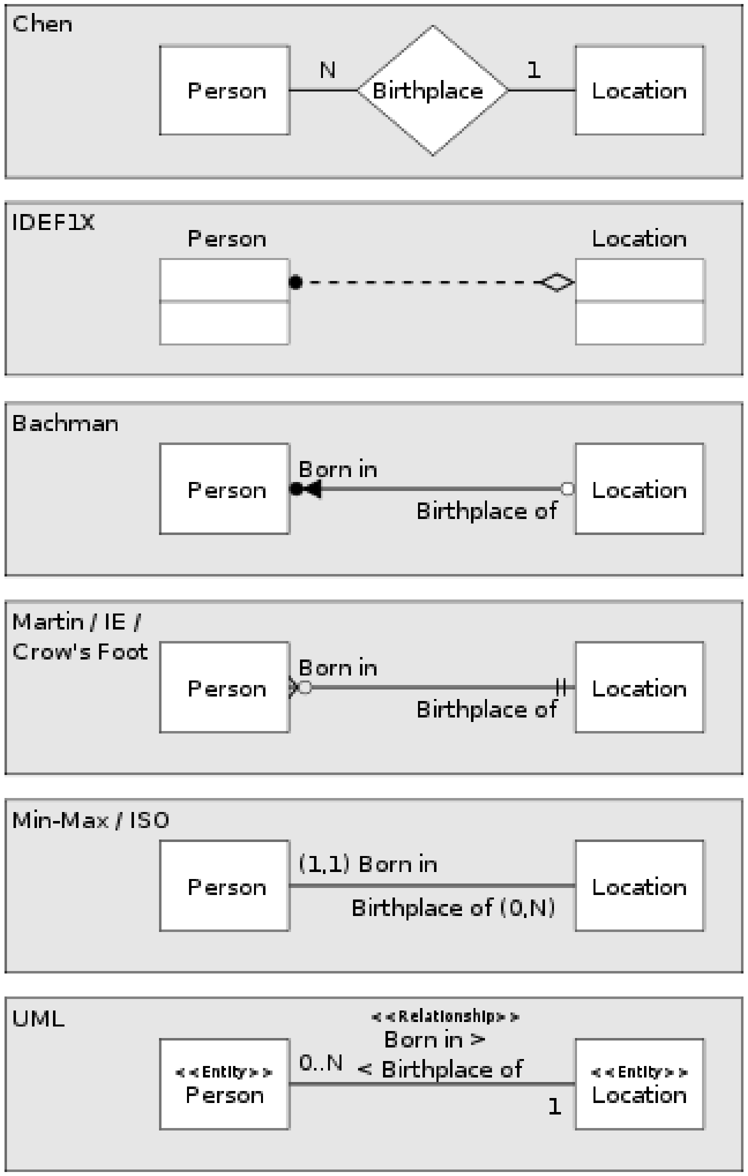
- Compare to the previous notation





Many "Standards"
Just For Binary Many-to-One

- https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model



Software for Modeling

Drawing Software

- Any software that can produced the shapes we need and connect them appropriately will be fine
- I prefer Visio 2010 but it is no longer available for download from MSDNAA
- You can use Visio 2016 after downloading from MSDNAA servers but it only runs on Windows (or in a Windows virtual machine)
- You can use the drawing shapes in PowerPoint
- You can use https://www.draw.io/
    1. Create New Diagram
    2. Create Blank Diagram (do not use any templates, use the Basic option)

Case Study

A Case Study

- Next, we will go through a relatively large example to make sure we know how to use ER diagrams
- We have a largish fragment of a large application to make sure we understand all the points
- The fragment has been constructed so it exhibits interesting and important capabilities of modeling
- It will also review the concepts we have studied earlier

- It is chosen based on its suitability to practice modeling using the power of ER diagrams
- It will also discuss various points, to be discussed later on how to design good relational databases

- We will use the cardinality notation by specifying numbers on the lines

Our Application

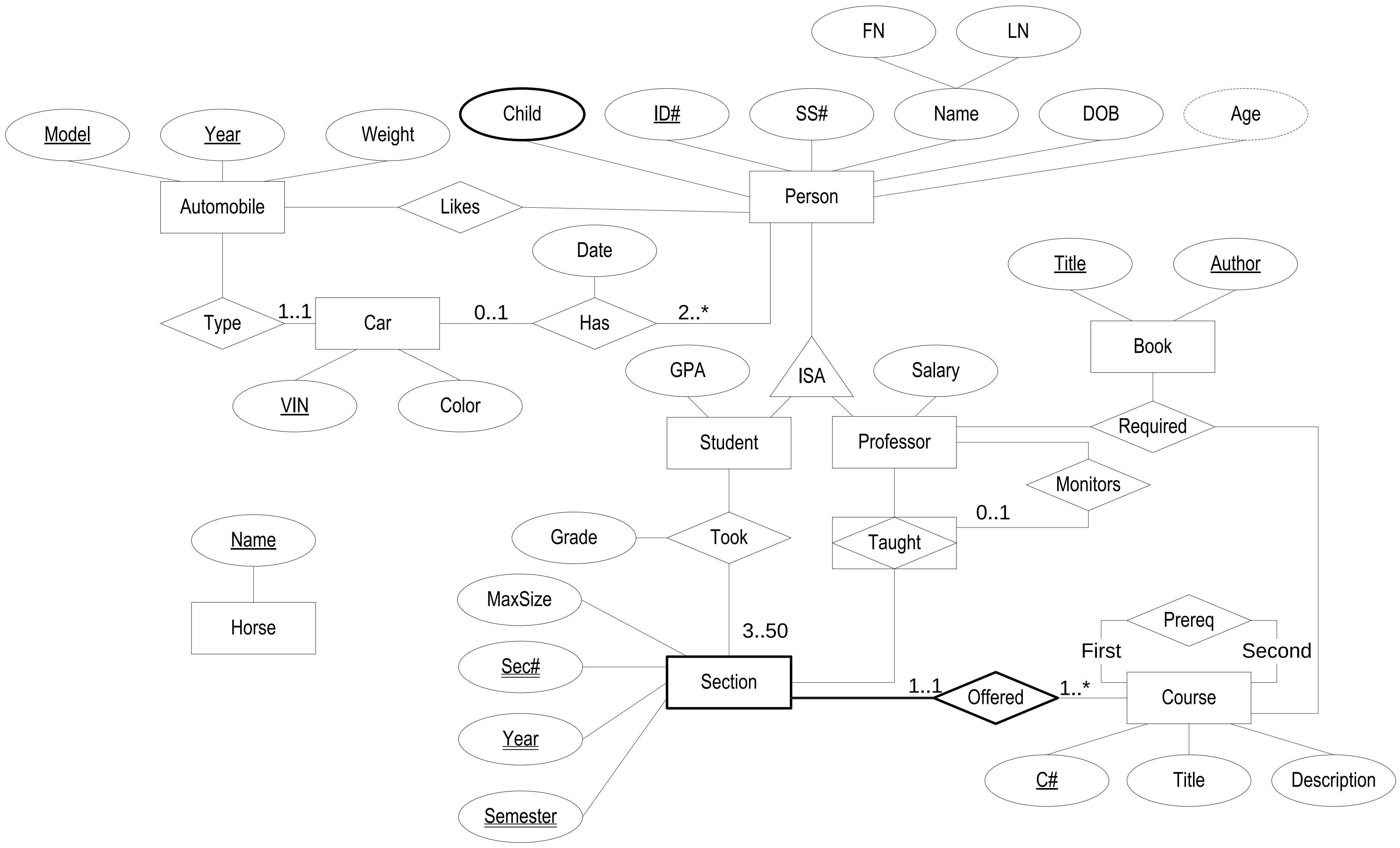
- We are supposed to design a database for a university
- We will look at a small fragment of the application and will model it as an entity relationship diagram annotated with comments, as needed to express additional features
- But it is still a reasonable "small" database
- In fact, larger than what is typically discussed in database courses, but more realistic for modeling real applications

Our Application

- Our understanding of the application will be described in a narrative form
- While we do this, we construct the ER diagram
- For ease of exposition (technical reasons only: limitations of the projection equipment) we look at the resulting ER diagram and construct it in pieces
- We will pick some syntax for annotations, as there is no commonly accepted standard
- One may try and write the annotations on the diagram itself using appropriate phrasing, but this will make our example too cluttered

Building the ER Diagram

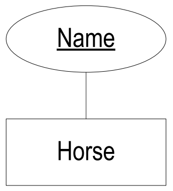
- We describe the application in stages, ultimately getting:



Horse

- ***Horse***; entity set
- Attributes:
  - ***Name; always known***
- Constraints
  - Primary Key: Name

Our ER Diagram



Horse

- We should specify what is the domain of each attribute, in this case, for Name only
- We will generally not do it in our example, as there is nothing interesting in it
  - We could say that Name is an alphabetic string of at most 100 characters, for example

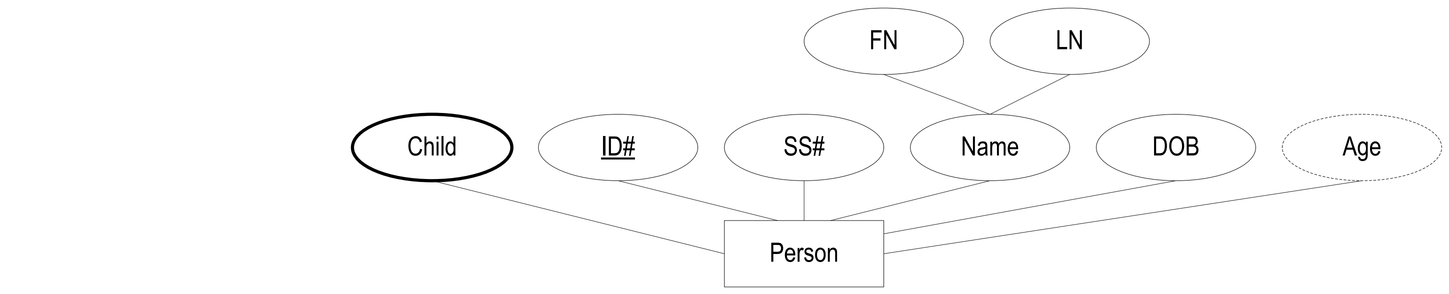
Person

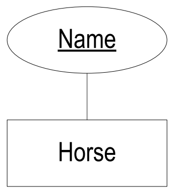
- ***Person***; entity set
- Attributes:
  - ***Child****; a multivalued attribute* (thick ellipse, although it may be difficult to see on the screen)
  - ***ID#; always known***
  - ***SS#; always known***
  - ***Name****; composite attribute, consisting of*
    - ***FN***
    - ***LN; always known***
  - ***DOB; always known***
  - ***Age****; derived attribute* (we should state how it is computed)

Person

- Constraints
  - Primary Key: ID#
  - Unique: SS# (**Note that this must be stated in words as we do not have a way of marking the diagram directly; it is very important to state that**)
  - A child is “allocated” to only one parent
    - In other words, if some child is a child of two persons in our database it will appear as an attribute of only one of them
    - The reason for this is that we want to identify children through their parents and we choose one parent for that purpose

Our ER Diagram





Person

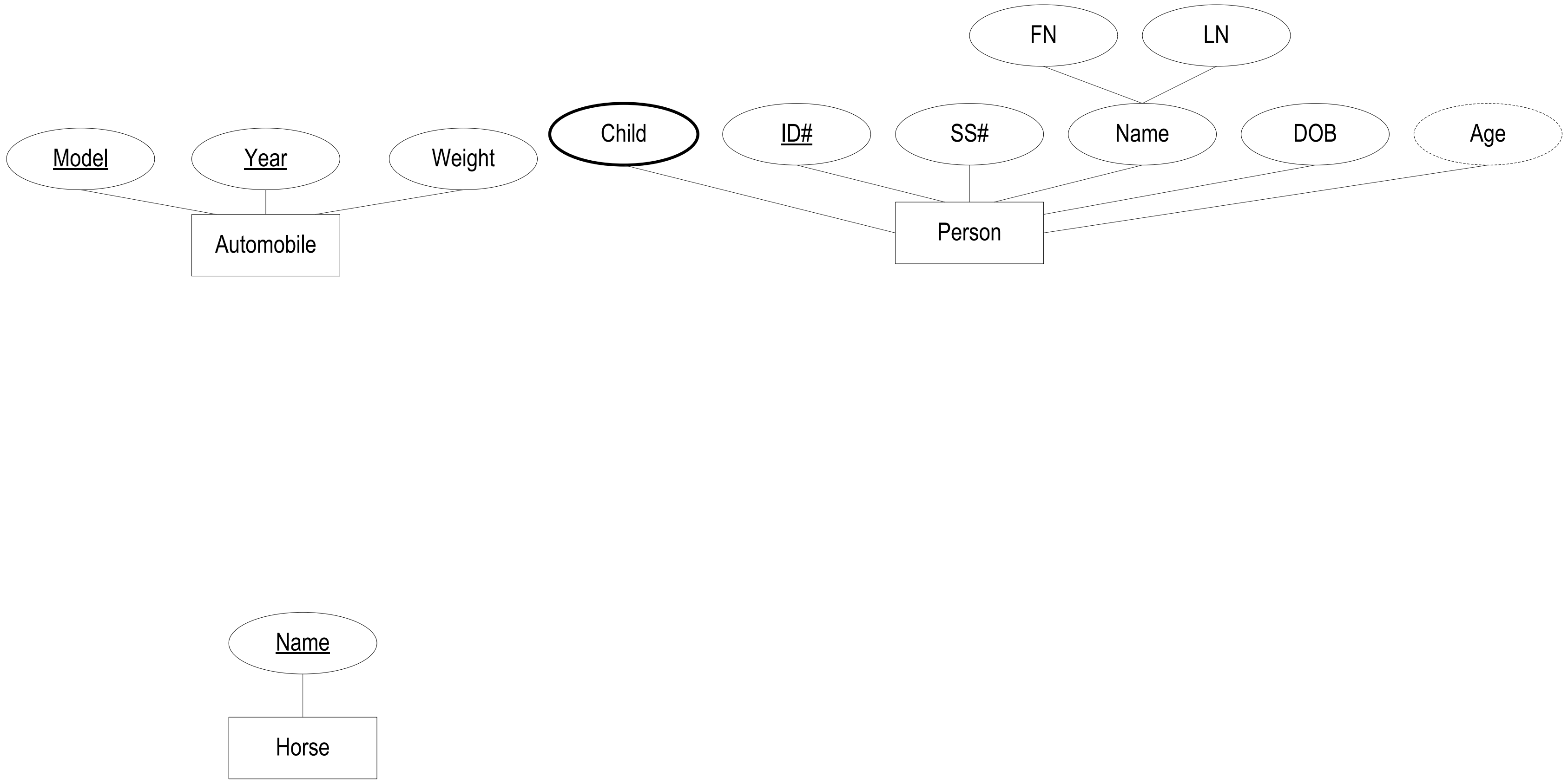
- Since ID# is the primary key (consisting here of one attribute), we will consistently identify a person using the value of this attribute (for later implementation as a relational database)
- Since SS# is unique, no two persons will have the same SS# (and we need to tell the database about that property, so it can be enforced)

Automobile

- ***Automobile***; entity set
- Attributes:
  - ***Model; always known***
  - ***Year; always known***
  - ***Weight; always known***
- Constraints
  - Primary Key: Model,Year

- Note: Automobile is a “catalog entry” in a catalog of cars that Amazon (if it sells cars) offers to you
  - It is not a specific “physical car”

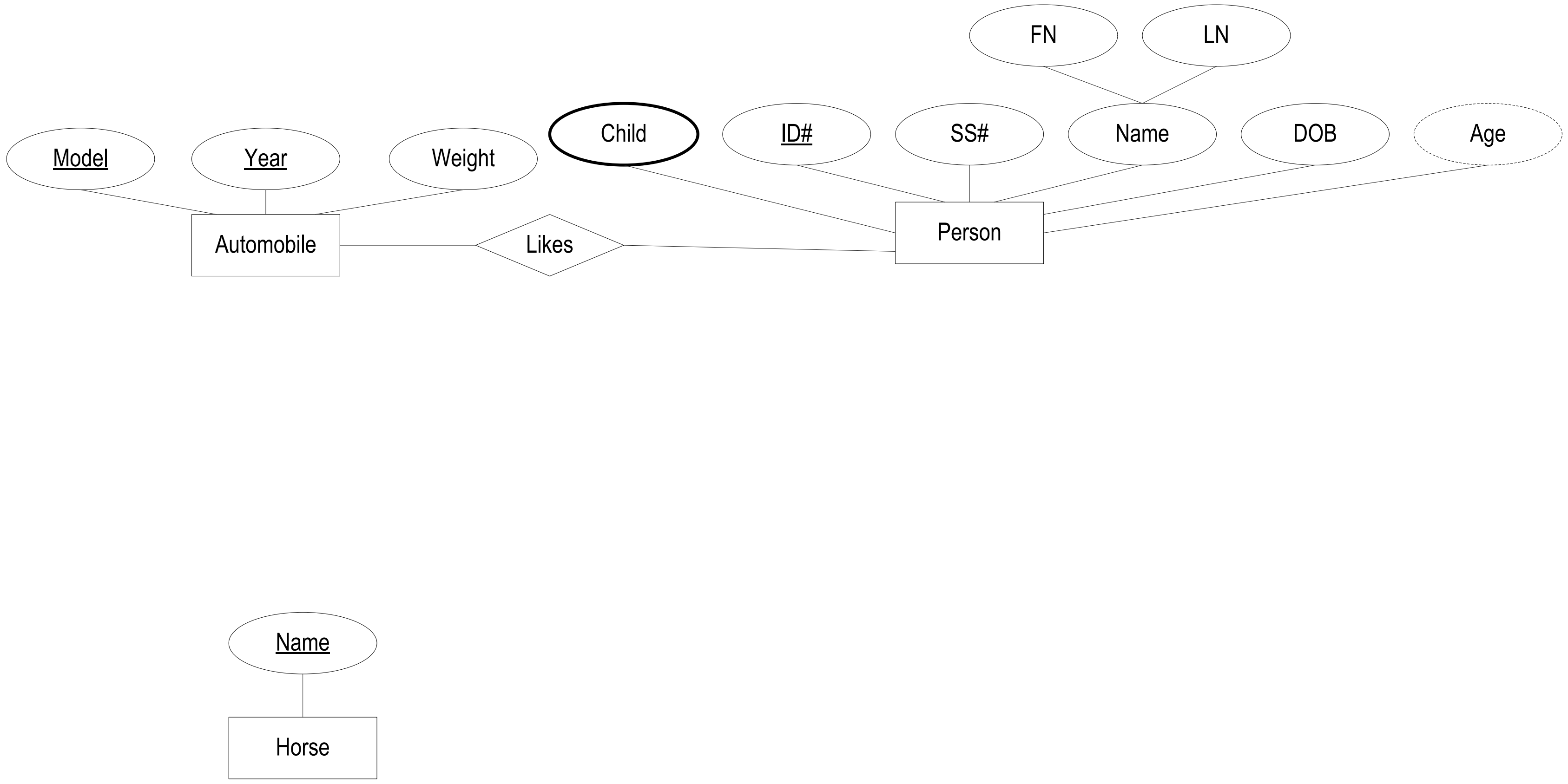
Our ER Diagram



Likes

- ***Likes***; relationship
- Relationship among/between:
  - ***Person***
  - ***Automobile***
- Attributes
- Constraints

Our ER Diagram



Likes

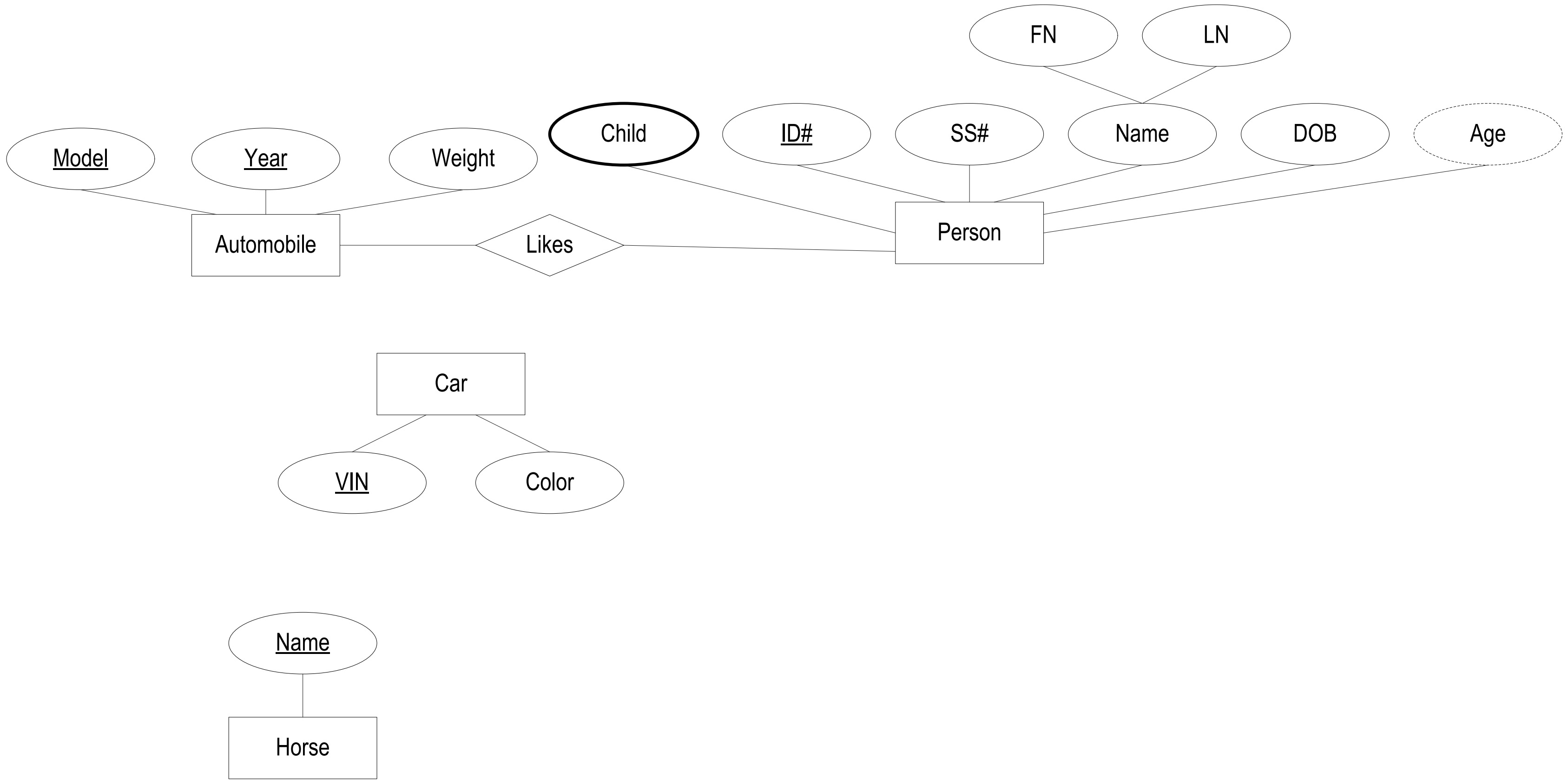
- This relationship has no attributes
- This relationship has no constraints
- This relationship is a general many-to-many relationship (as we have not said otherwise)
- This relationship does not have any cardinality constraints

Car

- ***Car***; entity set
- Attributes
  - ***VIN; always known***
  - ***Color***
- Constraints
  - Primary Key: VIN

- Note: Car is a “physical entity”: this is an entity you see driving on the street
  - VIN stands for “Vehicle Identification Number,” which is like a Social Security Number for cars

Our ER Diagram



Type

- ***Type***; relationship
- Relationship among/between:
  - ***Automobile***
  - ***Car***
- Attributes
- Constraints
  - Cardinality: 1..1 between Car and Type

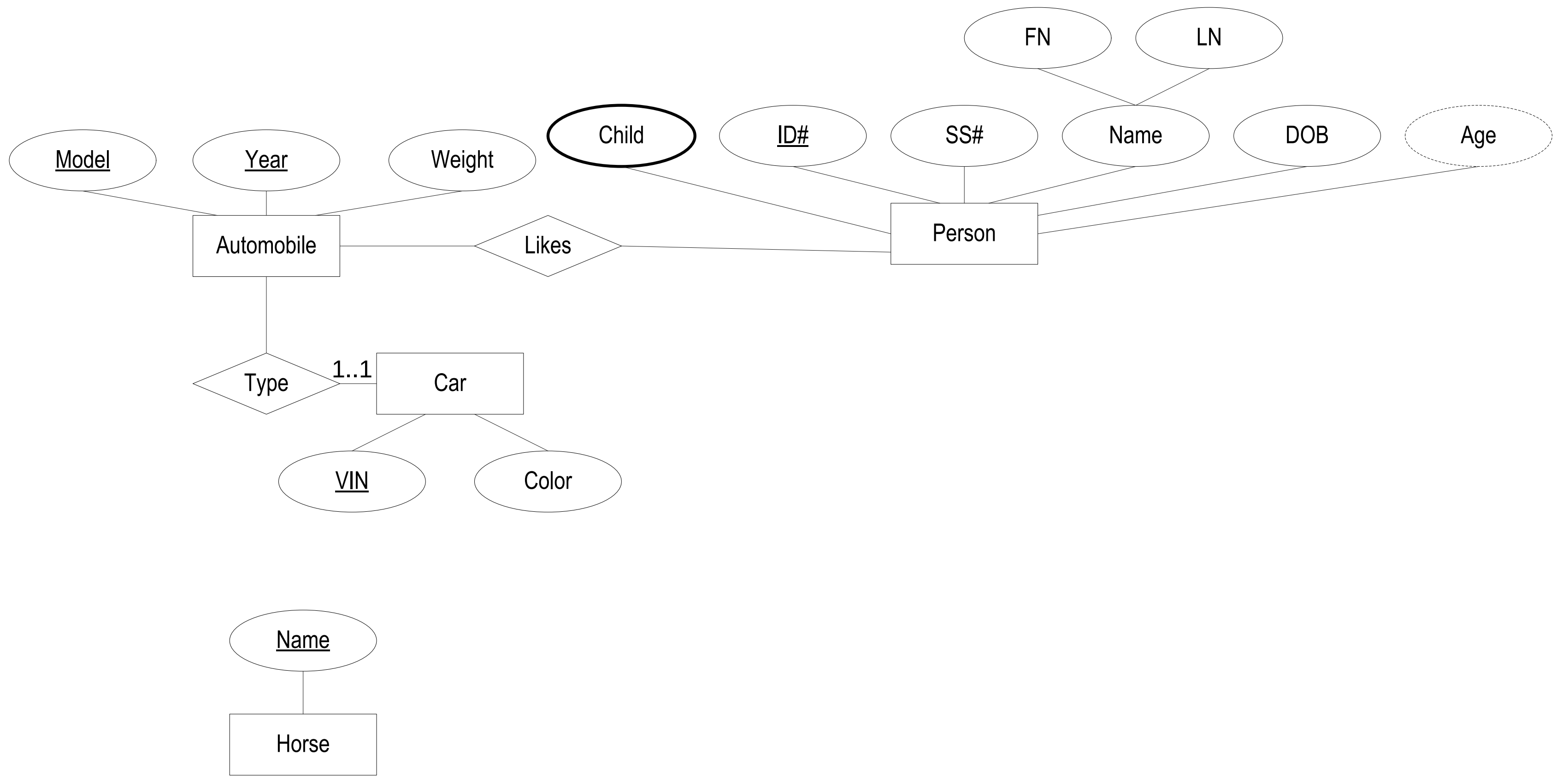
- This tells us for each physical car what is the automobile catalog entry of which it is an instantiation
  - Each car is an instantiation of an exactly one catalog entry

Type is Not ISA

- An entity in Automobile is a catalog entry, say a page in a book describing all automobiles
- An entity in Car is an object (in which people can ride) and it is on a road (or parking lot, etc.)
- So the entities in Car are not a subset of the entities in Automobile.
- That's why we do not have ISA here

- We will see ISA later

Our ER Diagram



Type

- We see that the relationship Type is:
- Many to one from Car to Automobile
- It is total not partial

  In other words, it is a total function from Car to Automobile

- Not every Automobile is in the range of the relationship

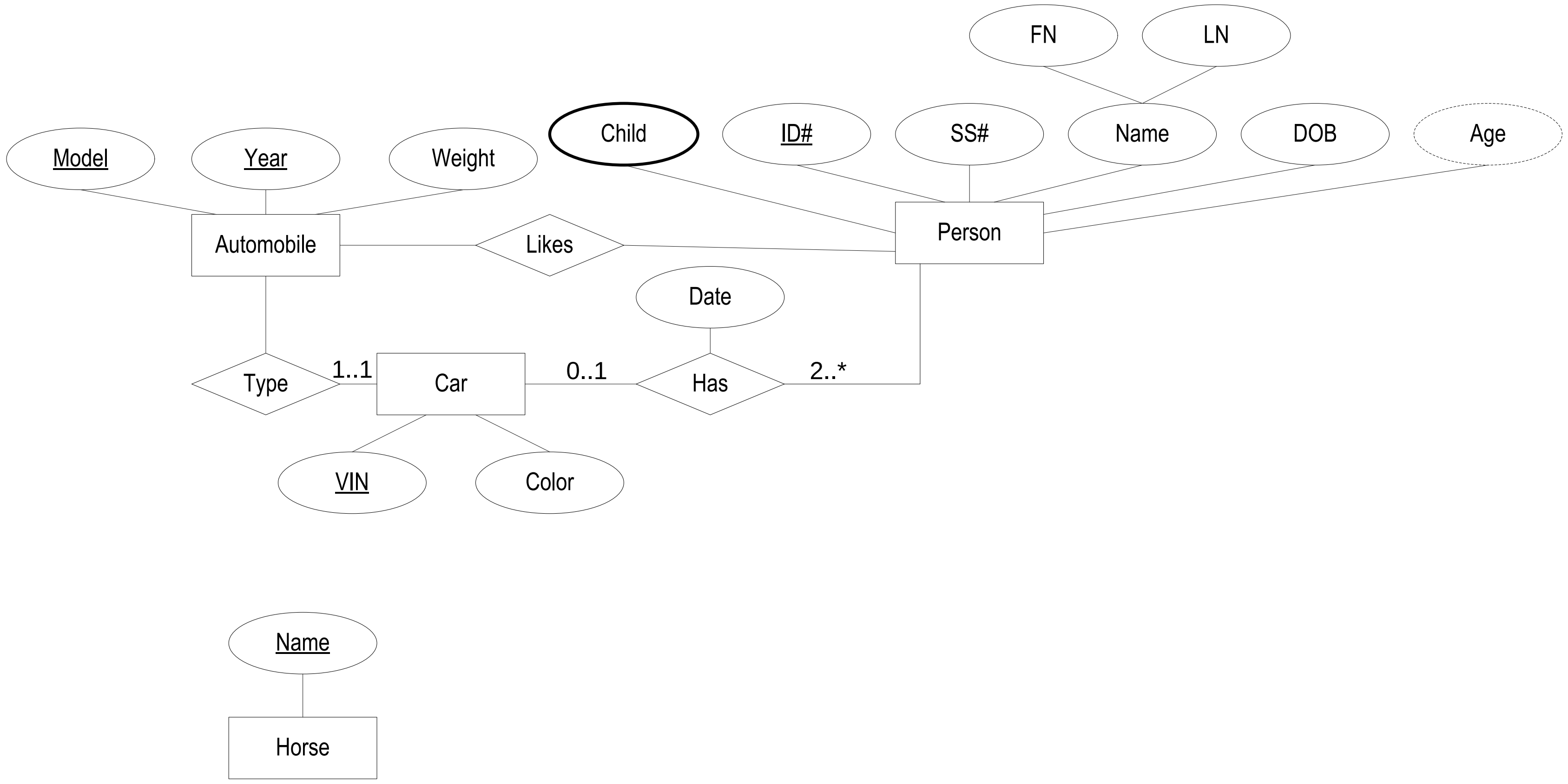
  There may be elements in Automobile for which no Car exists

Has

- ***Has***; relationship
- Relationship among/between
  - ***Person***
  - ***Car***
- Attributes
  - ***Date***
- Constraints
  - Cardinality: 2..* between Person and Has
  - Cardinality: 0..1 between Car and Has

- Date (if we know it) tells us when the person got the car
- Every person has at least two cars
- Every car can be had (owned) by at most one person
  - Some cars may have been abandoned and not owned by anybody

Our ER Diagram



Has

- We see that Has is a partial function from Car to Person

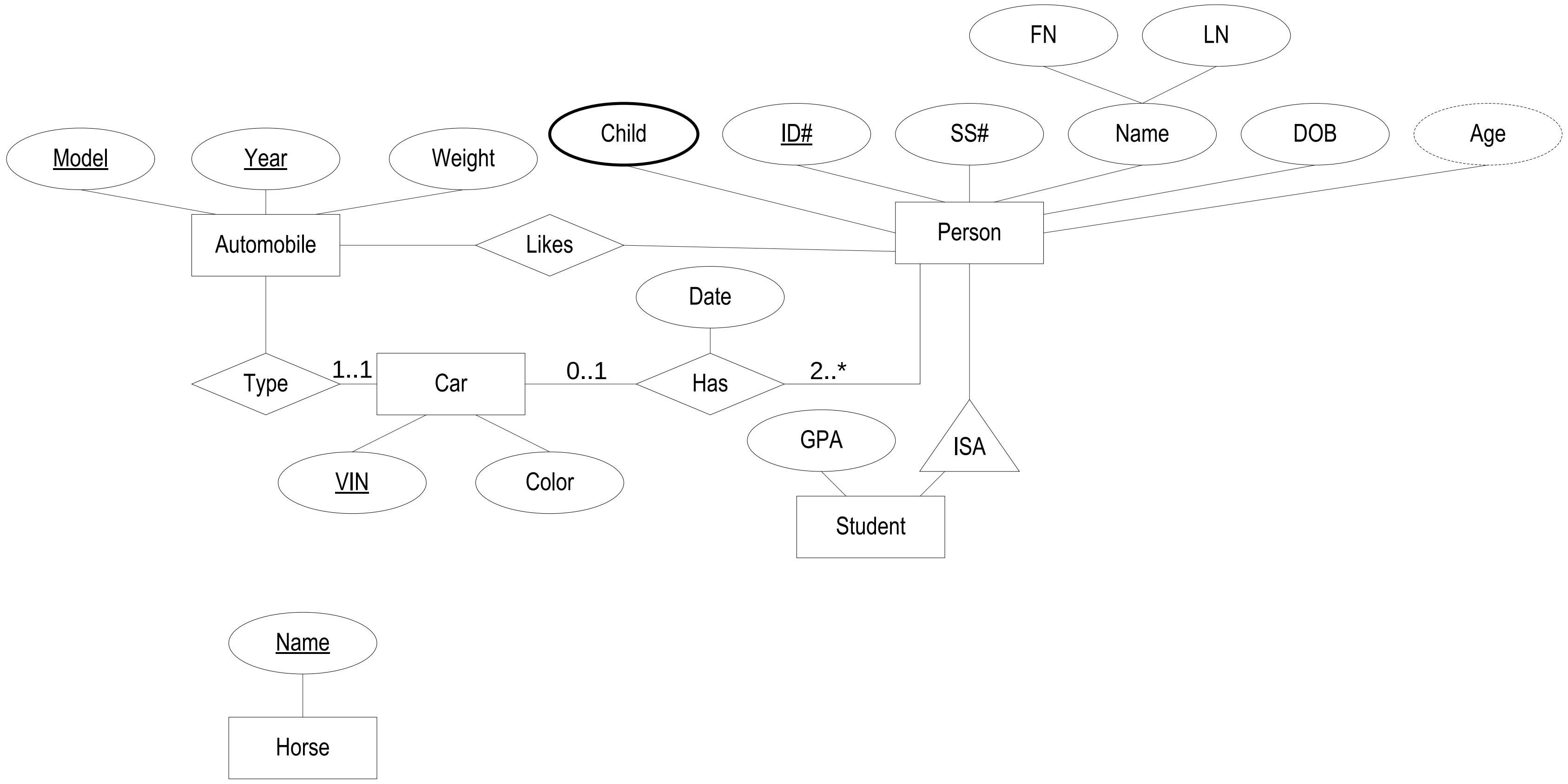
- Every Person is a “target” in this function (in fact at least twice)

Student

- ***Student***; entity set
- Subclass of Person
- Attributes
  - ***GPA***
- Constraints

- Note that Student is a weak entity
  - This is generally true for related subclass/class
  - It is identified through a Person
  - You may think of a student as being an “alias” for some person: “Split personality”
- Note that there is a binary one-to-one mapping between Student and Person and it is total from Student to Person
  - This is generally true for related subclass/class

Our ER Diagram



Student and ISA

- Every entity in Student is an entity in Person
- Therefore, the entities in Student are a subset of the entities in Person
- Therefore we have an ISA relationship
- Student inherits all the attributes of Person, as it is in some sense an alias for Person

Student and ISA

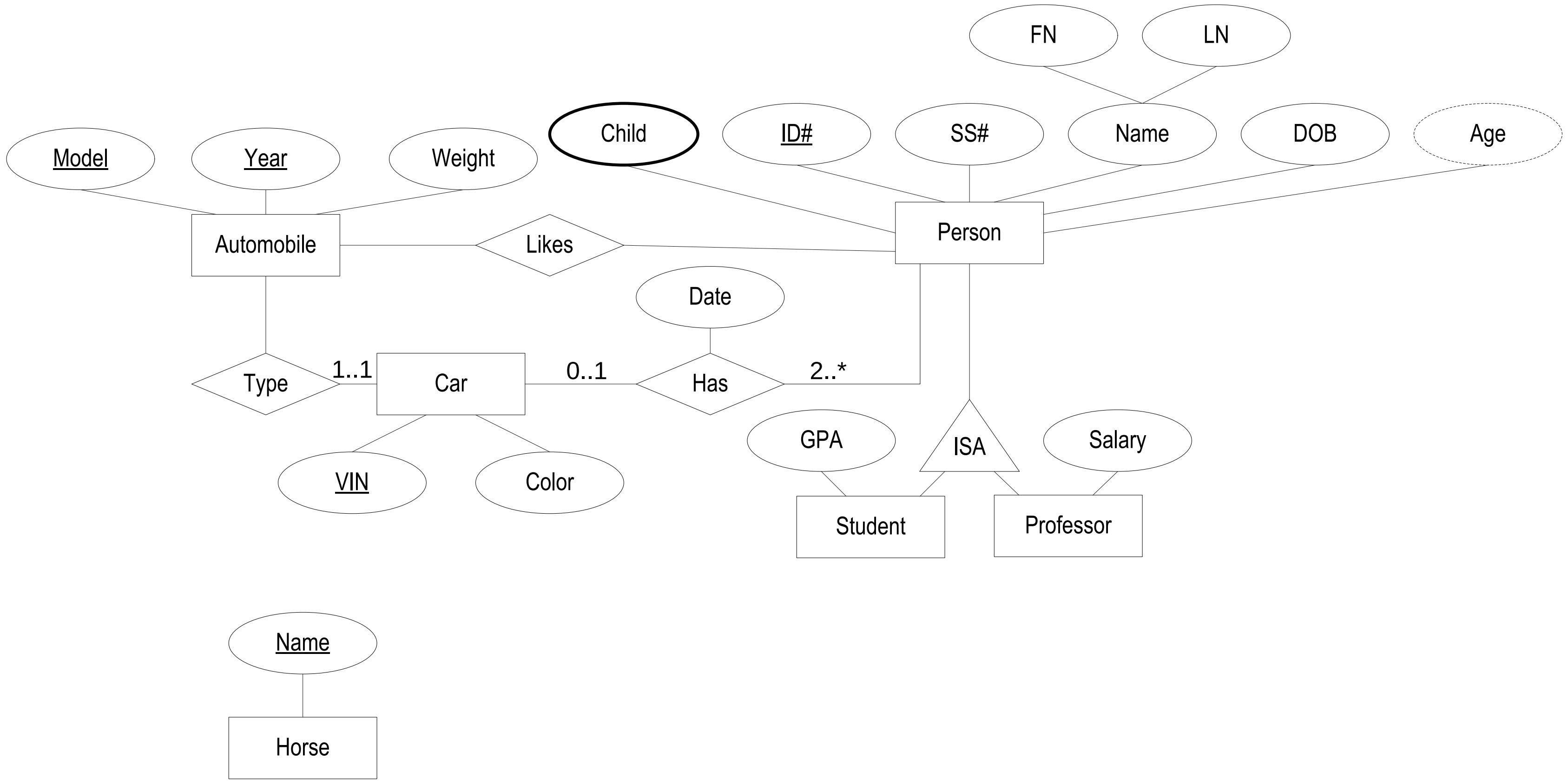
- ***Contrast*** with a non-ISA relationship between Automobile and Car
- Car did not inherit the attributes of Automobile
  - But since there is a total binary many-to-one relationship from Car to Automobile, for each entity in Car we can find out what the attributes of the corresponding unique entity in Automobile

Professor

- ***Professor***; entity set
- Subclass of Person
- Attributes
  - ***Salary; always known***
- Constraints

- Note that Professor is a weak entity
  - This is generally true for related subclass/class
  - It is identified through a Person
  - You may think of a student as being an “alias” for some person: “Split personality
- Note that there is a binary one-to-one mapping between Professor and Person and it is total from Professor to Person
  - This is generally true for related subclass/class

Our ER Diagram



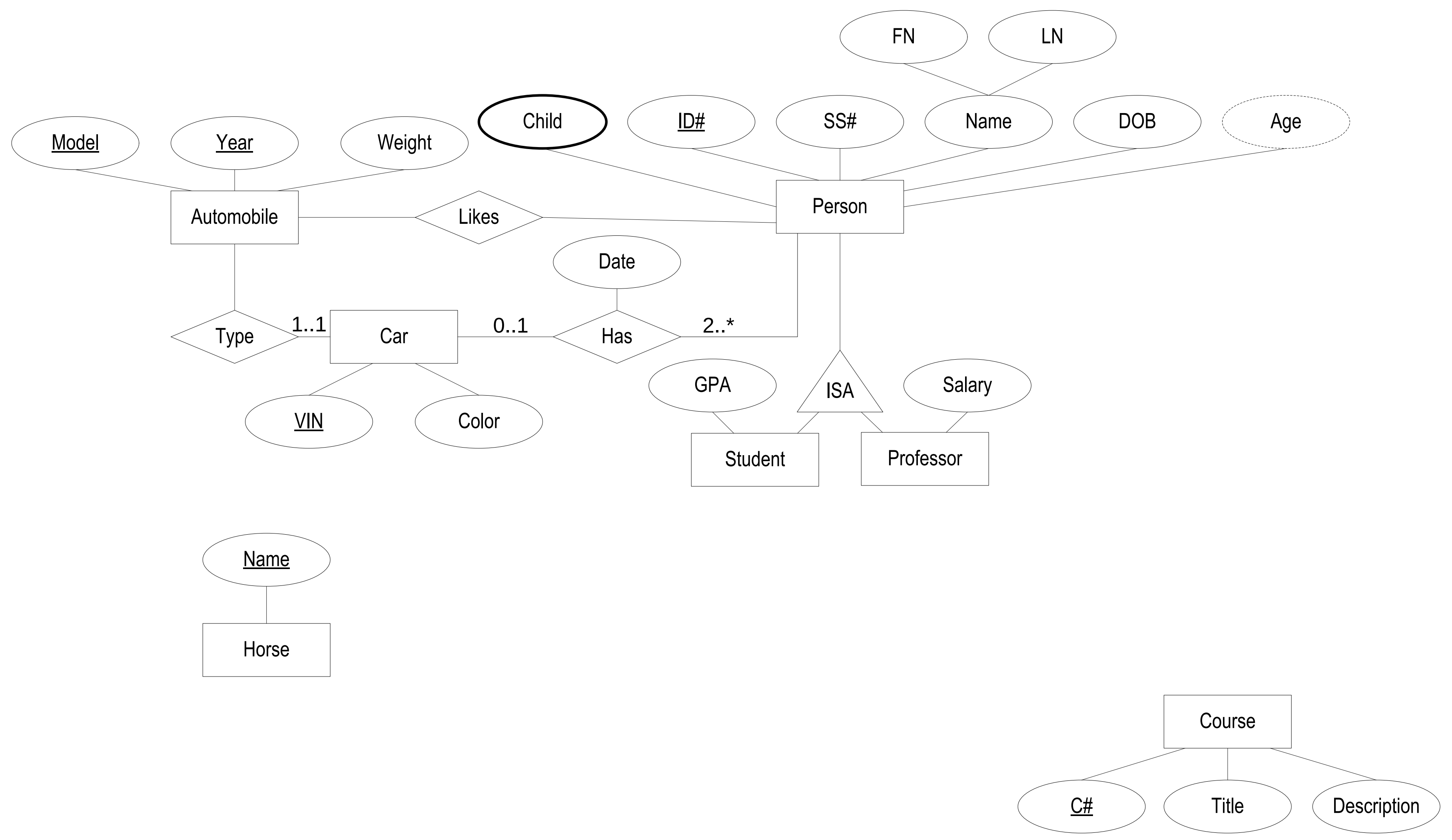
Course

- ***Course***; entity set
- Attributes:
  - ***C#; always known***
  - ***Title; always known***
  - ***Description***
- Constraints
  - Primary Key: C#

- Course is a catalog entry appearing in the bulletin
  - Not a particular offering of a course
  - Example: CSCI-GA.2433 (which is a C#), not necessarily Fall 2018 offering

- In practice, we may need an attribute specifying the number of points, but let’s assume that it is always 3, so we do not have the attribute

Our ER Diagram



Prereq

- ***Prereq***; relationship
- Relationship among/between:
  - ***Course***; role: First
  - ***Course***; role: Second
- Attributes
- Constraints

- We have a directed graph on courses, telling us prerequisites for each course, if any
  - To take “second” course every “first” course related to it must have been taken previously
  - We needed the roles first and second, to be clear
  - Note how we model well that prerequisites are not between offerings of a course but catalog entries of courses
  - Note however, that we cannot directly “diagram” that a course cannot be a prerequisite for itself, and similar, so these need to be annotated

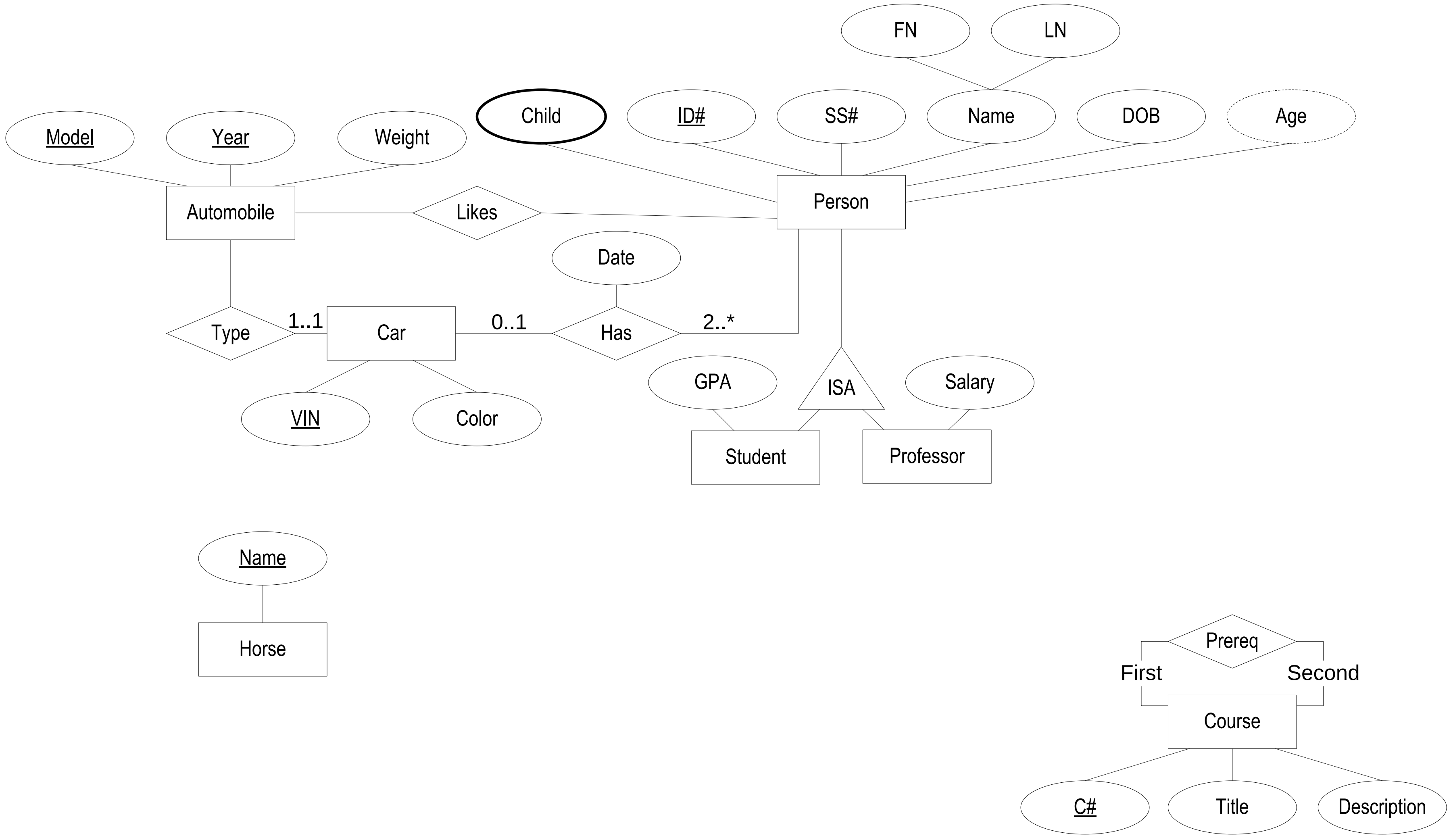
Important Digression

- Ultimately, we will store (most) relationships as tables
- So, comparing to our example for Likes, Prereq instance could be

Prereq	First	Second
	101	103
	101	104
	102	104
	104	105
	107	106
	107	108

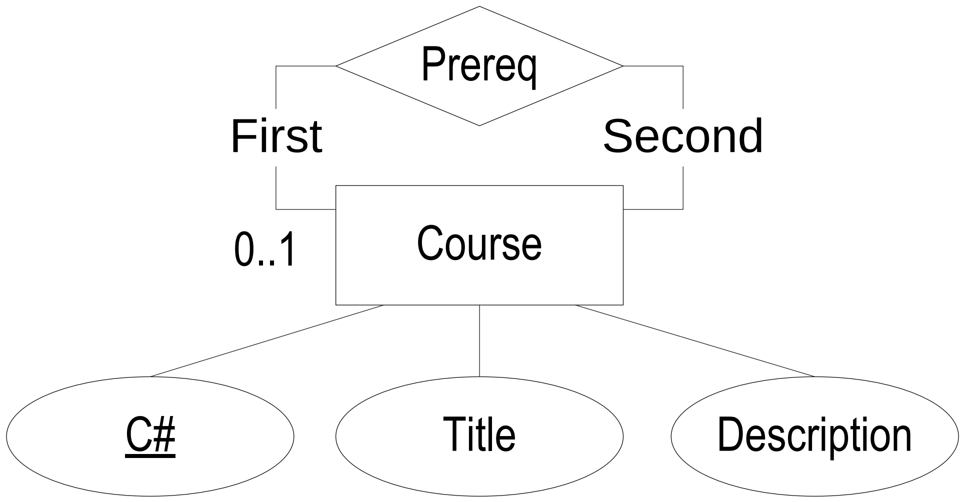
- Where we identify the “participating” entities using their primary keys, but renaming them using roles
- So looking at the table we see that 101 is a prerequisite for 104 but 104 is not a prerequisite for 101

Our ER Diagram



Assuming Prereq is Many-to-One

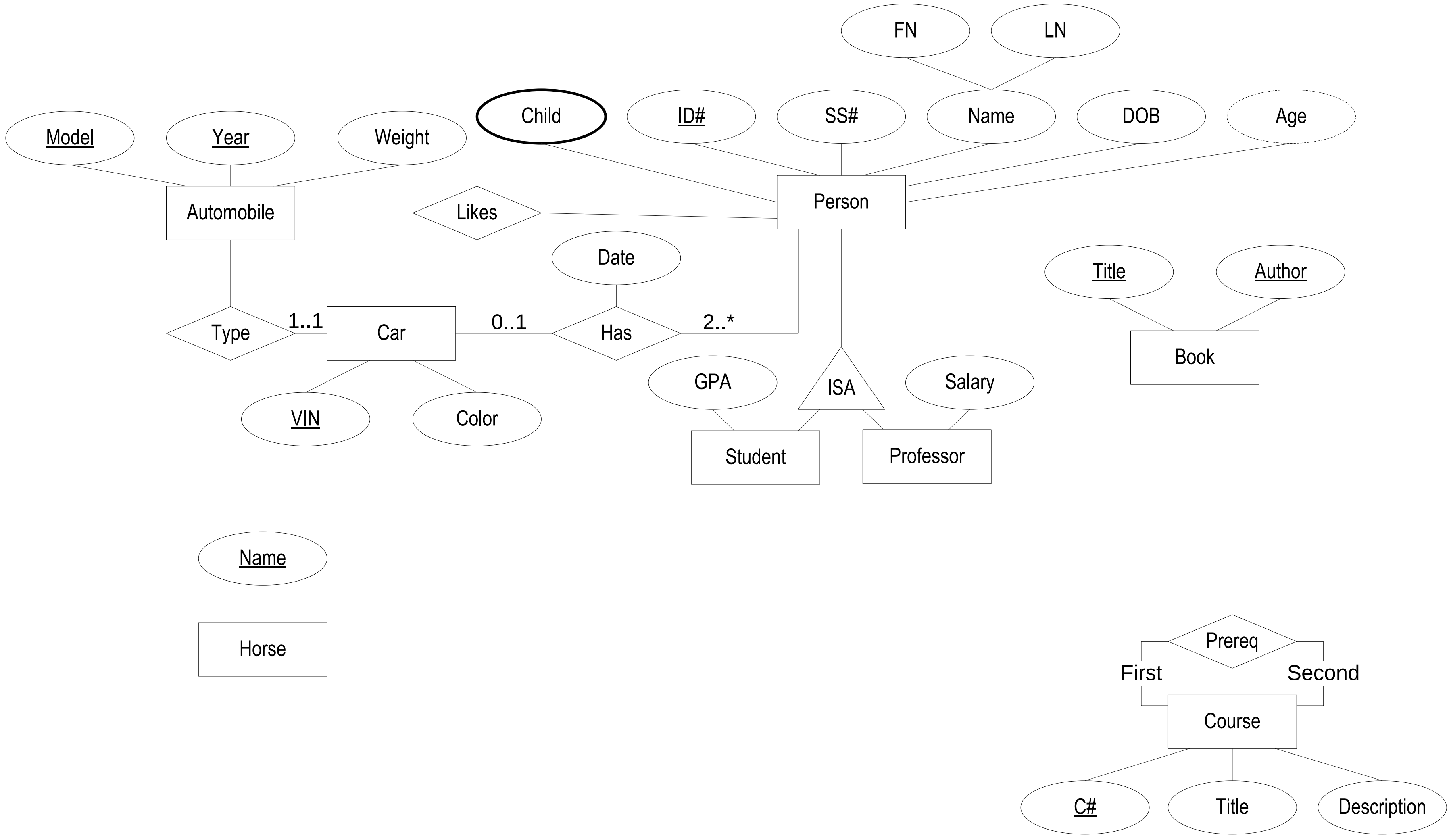
- We did not assume that Prereq is Many-to-One
- If it were many-to-one, for example is each Course is a Preq for at most one Course, we would add an a cardinality property



Book

- ***Book***; entity set
- Attributes:
    - ***Author; always known***
    - ***Title; always known***
- Constraints
    - Primary Key: Author,Title

Our ER Diagram

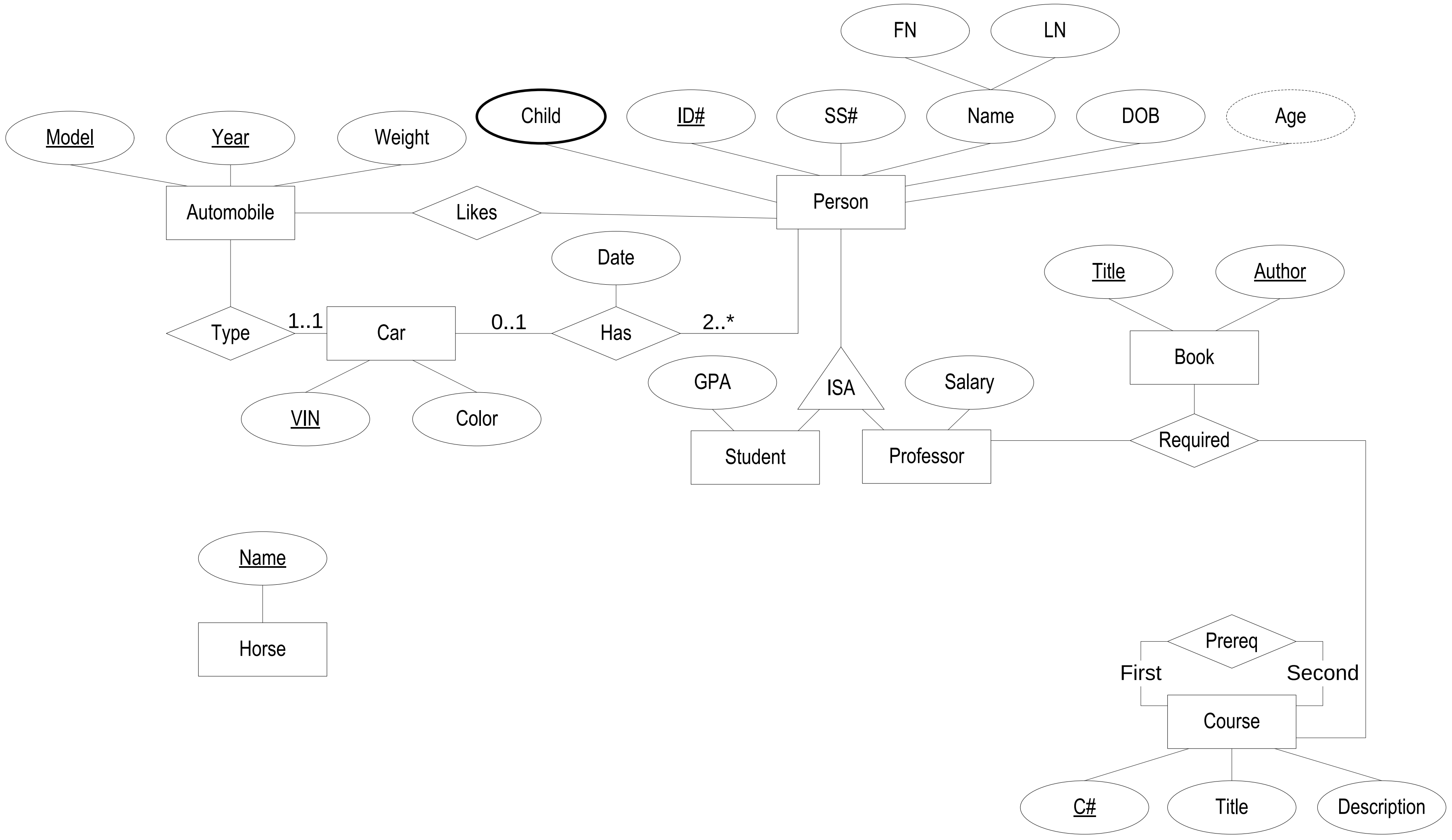


Required

- ***Required***; relationship
- Relationship among/between:
  - **Professor**
  - **Course**
  - **Book**
- Attributes
- Constraints

- A professor specifies that a book is required for a course

Our ER Diagram



Required

- Note that there are no cardinality or other restrictions
- Any professor can require any book for any course and a book can be specified by different professors for the same course
- A book does not have to required for any course

Section

- ***Section***; entity set
- Attributes:
  - ***Year; always known***
  - ***Semester; always known***
  - ***Sec#; always known***
  - ***MaxSize***
- Constraints
  - Discriminant: Year, Semester, Sec#
  - Identified through relationship Offered to Course
  - Each Course has to have at least one Section (we have a policy of not putting a course in a catalog unless it has been offered at least once)

Section

- Section is a weak entity
- It is related for the purpose of identification to a strong entity Course by a new relationship Offered
- It has a discriminant, so it is in fact identified by having the following specified

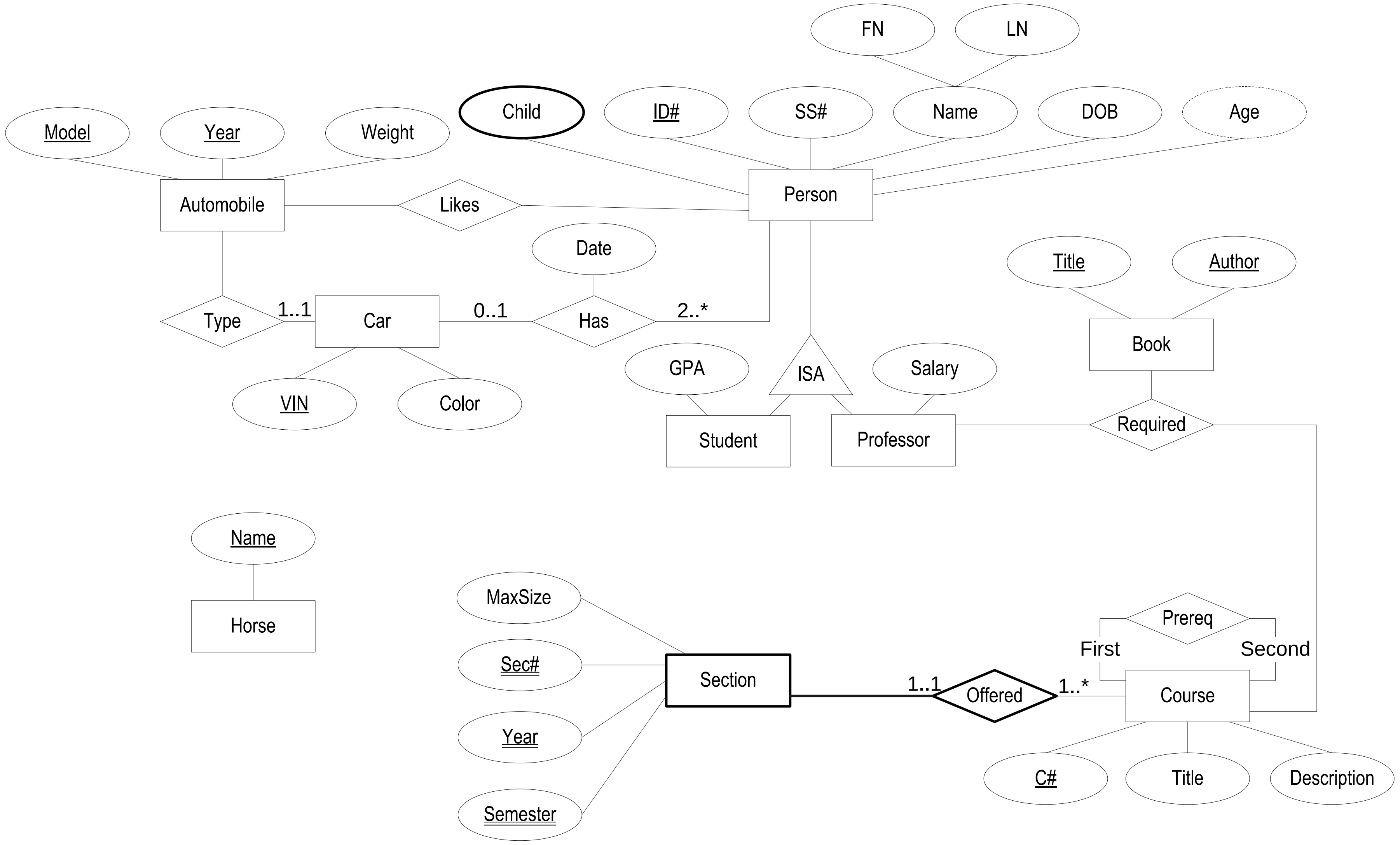
  C#, Year, Semester, Sec#
- Our current section is identified by:

  CSCI-GA.2433, 2017, Fall, 001

Offered

- ***Offered***; relationship
- Relationship among/between:
  - **Course**
  - **Section**
- Attributes
- Constraints
  - Section has to be related to exactly one course (this automatically follows from the fact that section is identified through exactly one course, so maybe we do not need to say this); so there is a binary many-to-one mapping between Section and Course
  - This is generally true for pairs of “related” weak/strong entities
  - Course has to be related to at least one section (see above); this a peculiarity of the policy of the university and not a general structural constraint
- Note: May be difficult to see, but Section and Offered are both drawn with thick lines

Our ER Diagram



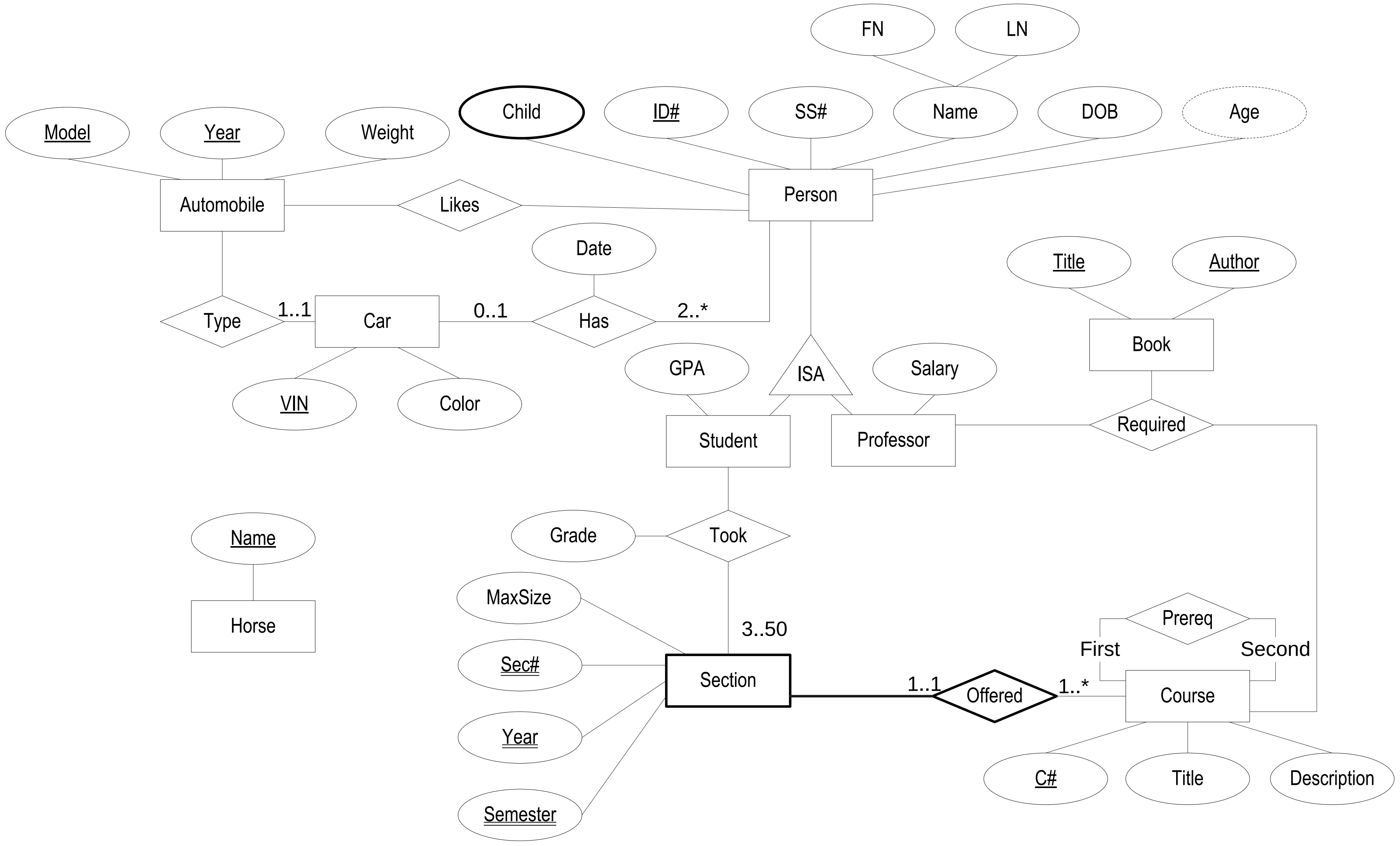
Took

- ***Took***; relationship
- Relationship among/between
  - ***Student***
  - ***Section***
- Attributes
  - ***Grade***
- Constraints
  - Cardinality: 3..50 between Section and Took (this means that a section has between 3 and 50 students)

- Students register for sections and not for courses

- For the time being, we did not pay attention that the user told us that Grade is a derived attribute computed in some specific way

Our ER Diagram

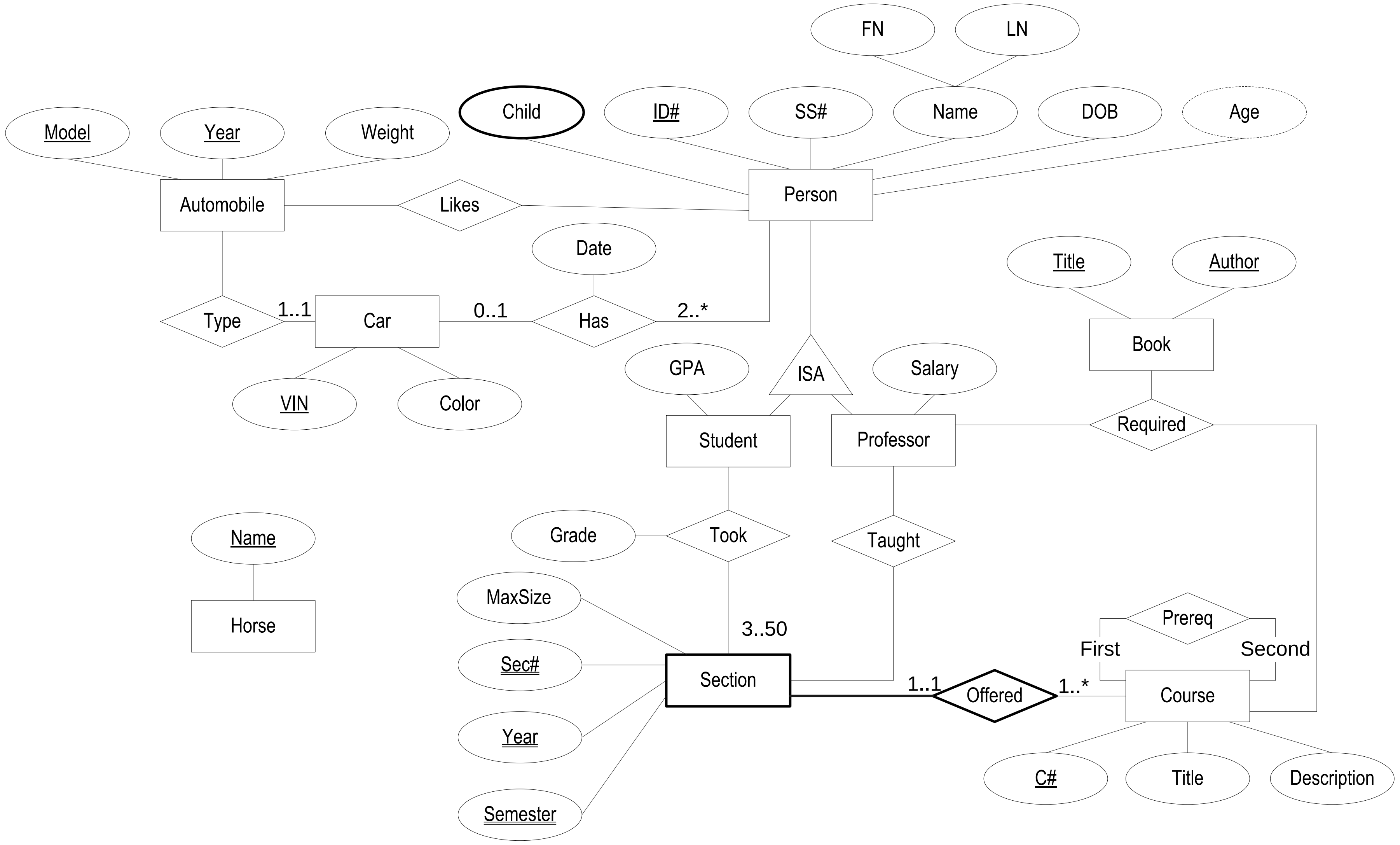


Taught

- ***Taught***; relationship
- Relationship among/between
  - ***Professor***
  - ***Section***
- Attributes

- This tells us which professor teach which sections
  - Note there is no cardinality constraint: any number of professors, including zero professors can teach a section (no professor yet assigned, or hypothetical situation)
  - If we wanted, we could have put 1..* between Section and Taught to specify that at least one professor has to be assigned to each section

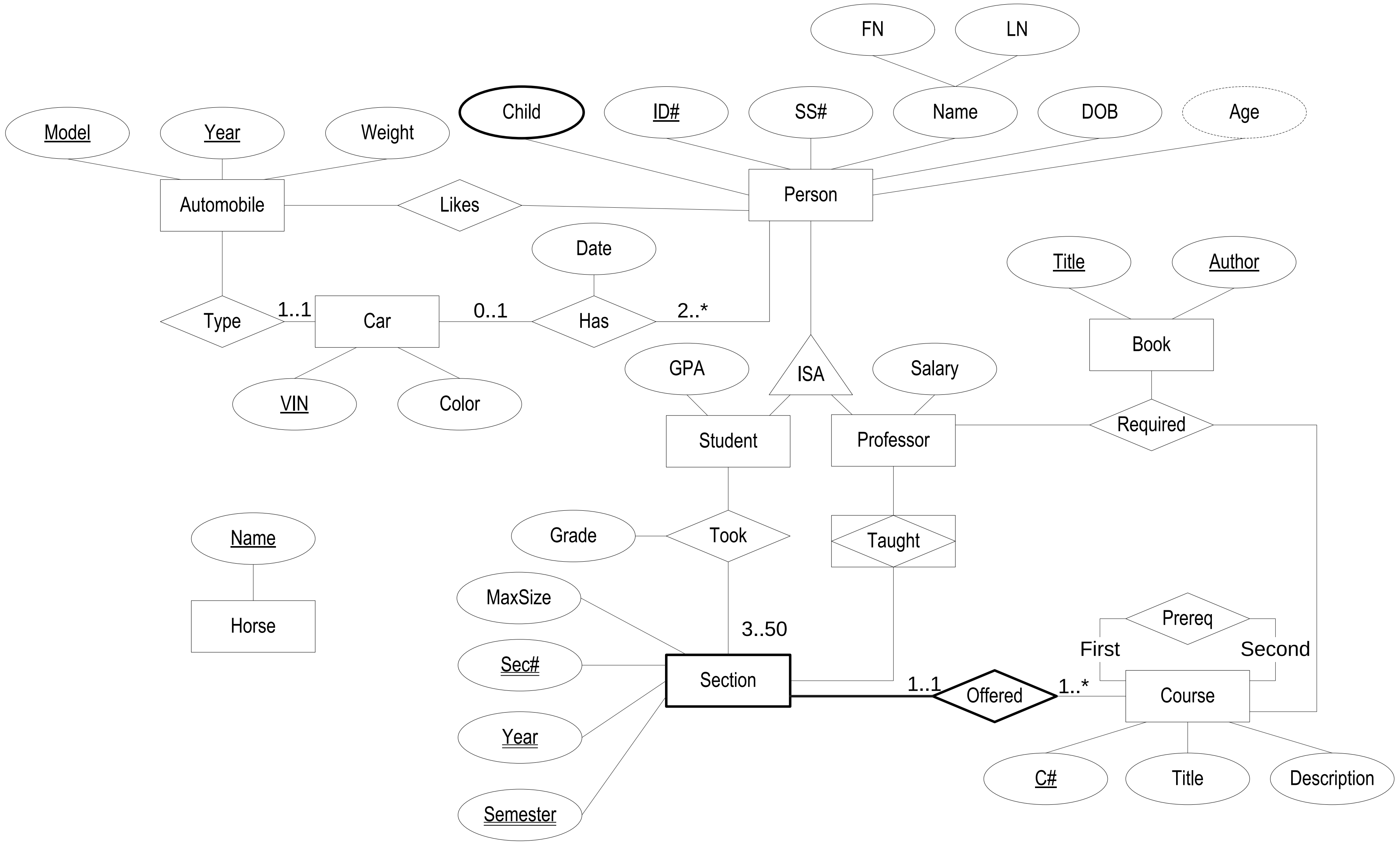
Our ER Diagram



Taught

- We want to think of Taught as an entity
  - We will see soon why

Our ER Diagram

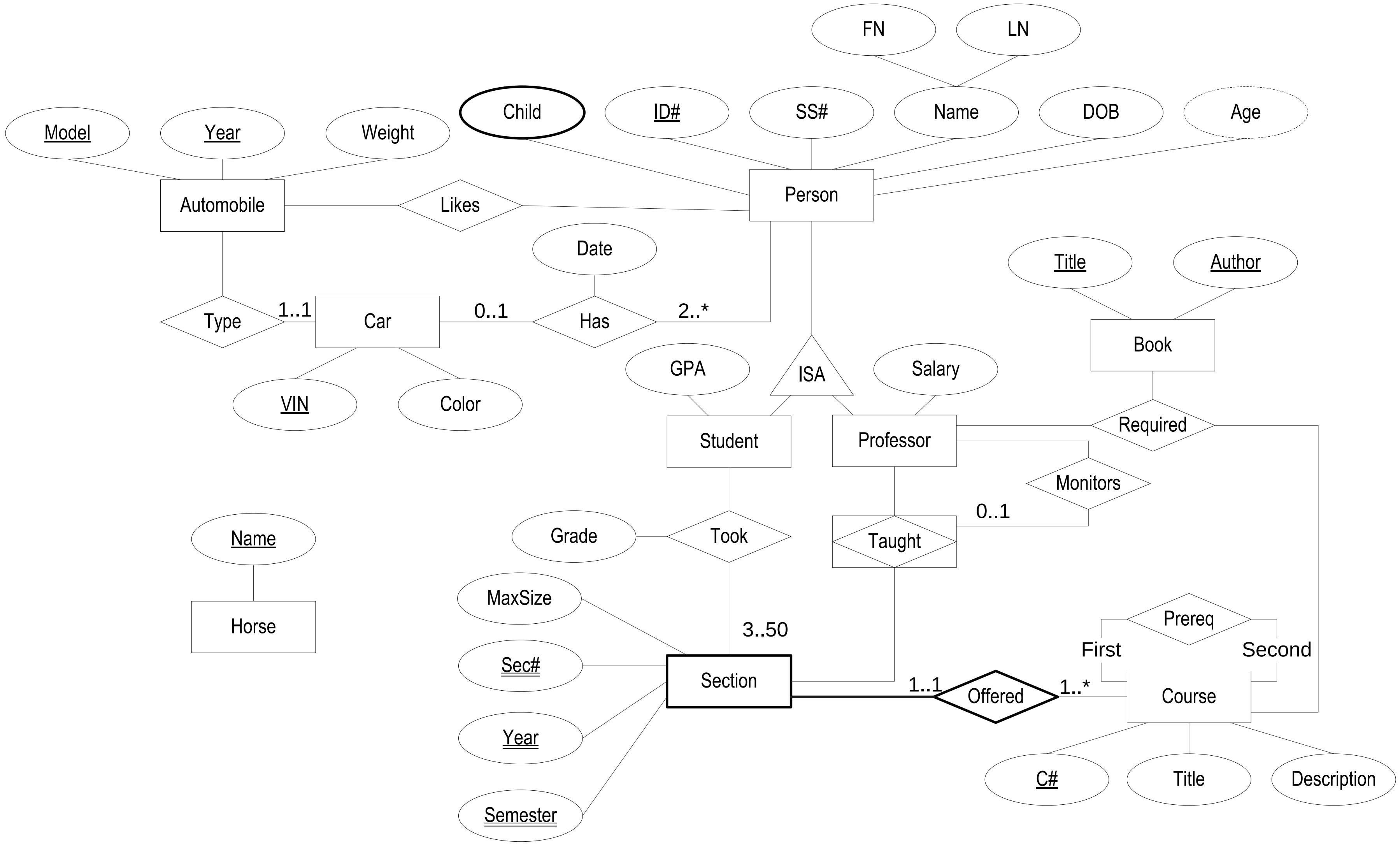


Monitors

- ***Monitors***; relationship
- Relationship among/between
  - ***Professor***
  - ***Taught*** (considered as an entity)
- Attributes
- Constraints
  - Cardinality: 0..1 between Taught and Professor

- This models the fact that Taught (really a teaching assignment) may be monitored by a professor, and at most one professor is needed for such monitoring
  - We are not saying whether the professor monitoring the assignment has to be different from the teaching professor in this assignment (but we could do it using annotations or in SQL DDL, as we shall see later)

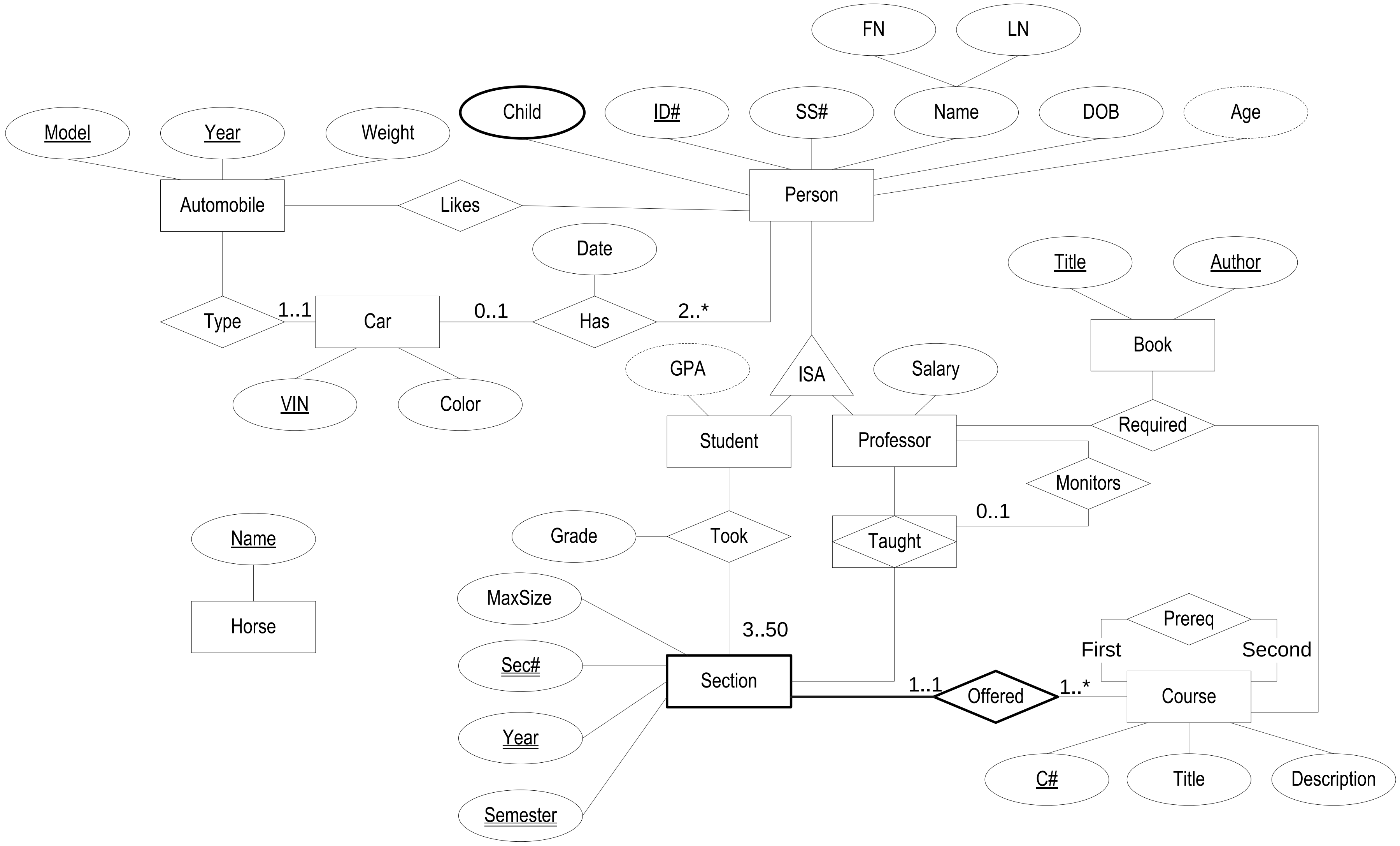
Our ER Diagram



GPA

- After showing the diagram to our customer we realize that GPA should be modeled as a derived attribute, as it is computed from the student’s grade history
- Assume that every Course has 3 points then we know how to compute the GPA from the data reflected in the diagram (and our knowledge of the rule about the 3 points)
- So, we need to revise the diagram
- If different Courses could have different number of points, then the natural thing would be to have Points as an attribute of the Course, which would allow us to compute GPA

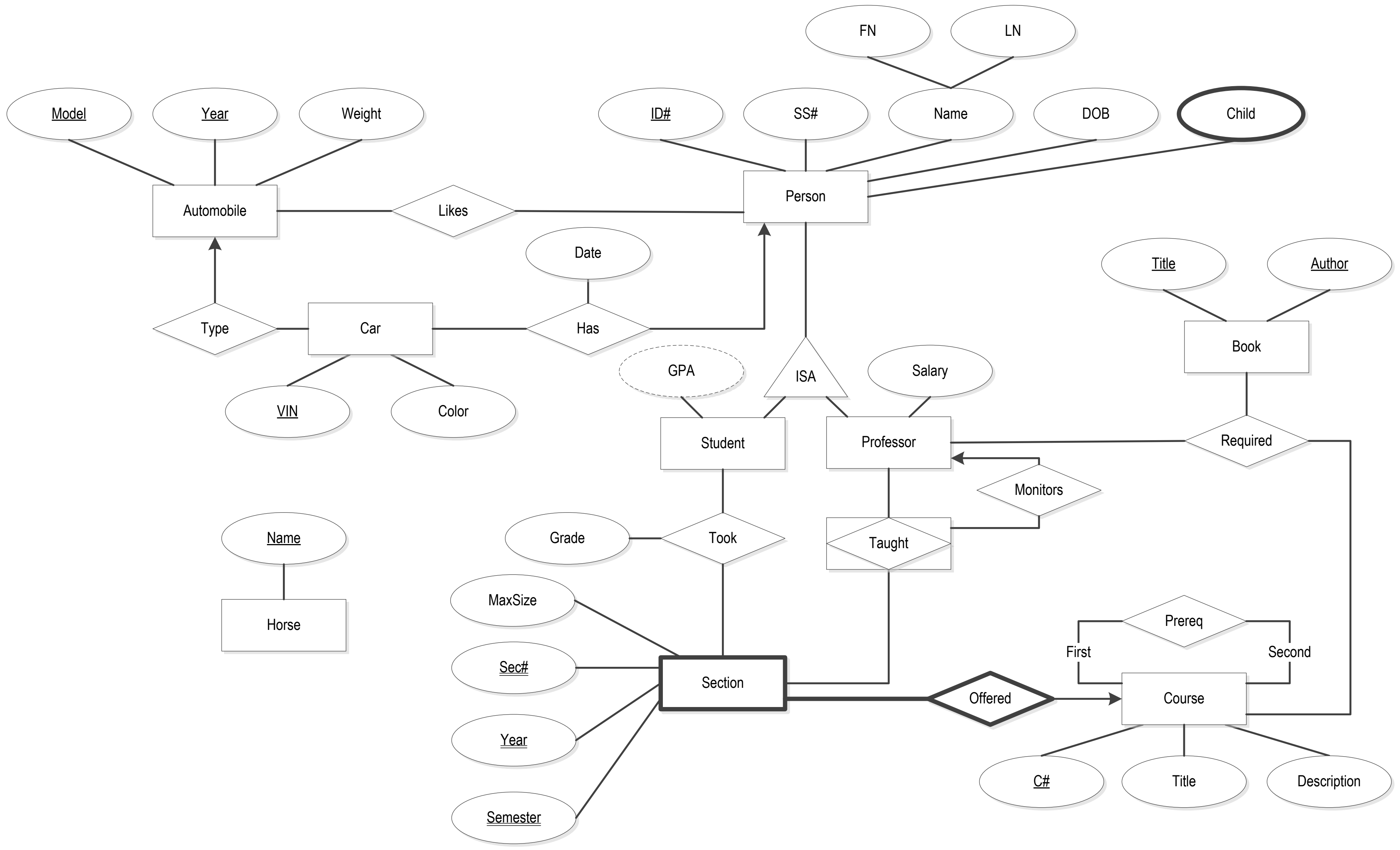
Our ER Diagram



Annotations (The Necessary And Only The Necessary)

- Unique: SS#
- Always known: SS#, LN, DOB, Weight, Salary, Title (in Course)

Our ER Diagram

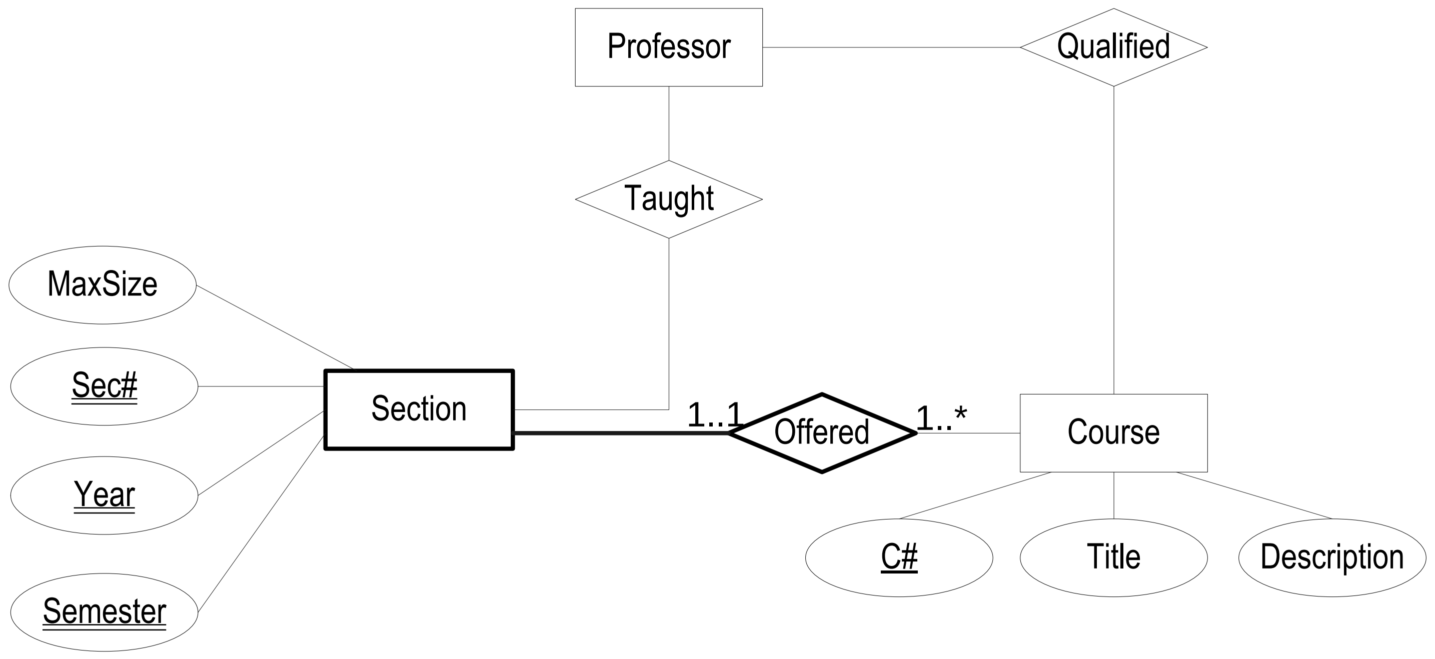


Annotations (The Necessary And Only The Necessary)

- Unique: SS#
- Always known: SS#, LN, DOB, Weight, Salary, Title (in Course)
- A Person has at least 2 Cars
- For every Course there is at least 1 Section
- A Section has between 3 and 50 students
- Type is total

Some Constraints Are Difficult to Specify

- Imagine that we also have relationship Qualified between Professor and Course specifying which professors are qualified to teach which courses
- We probably use words and not diagrams to say that only a qualified professor can teach a course



Annotate, Annotate, Annotate …

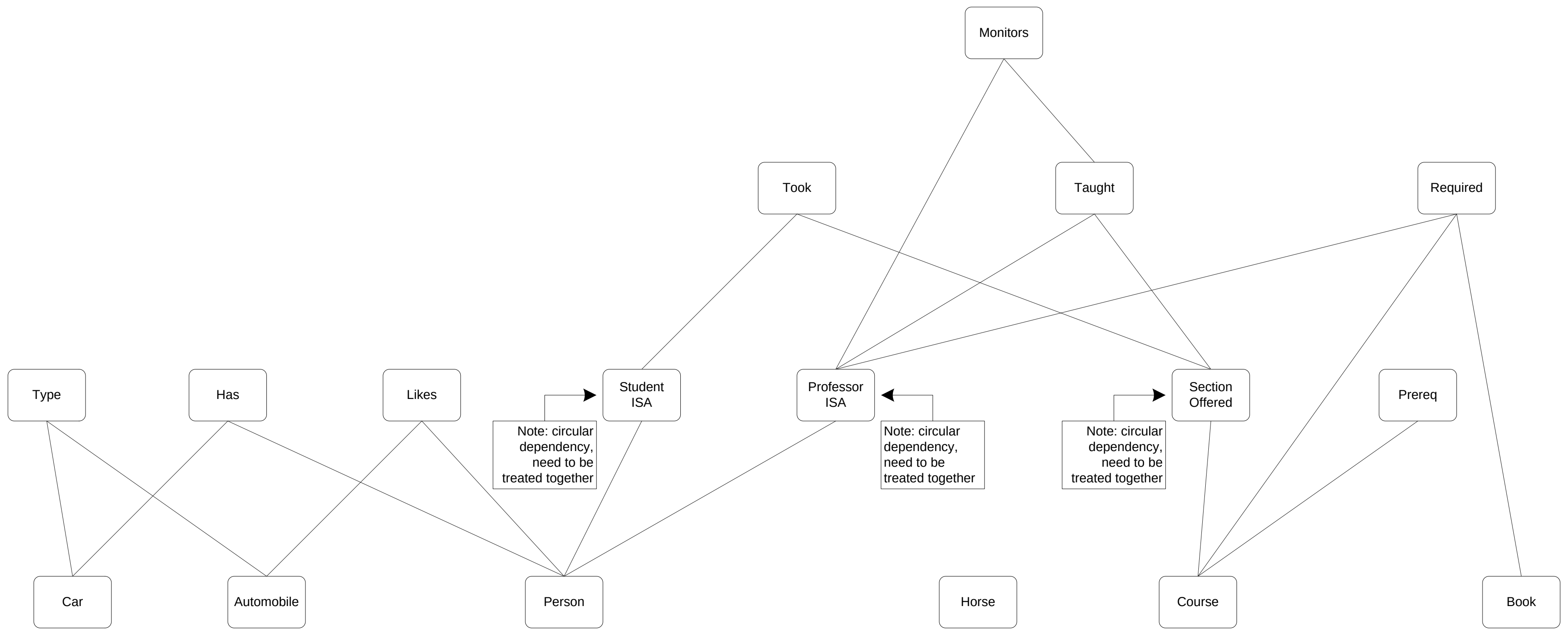
- **An ER diagram should be annotated with all known constraints**
- You do not need to put in annotations any constraints that are reflected in the diagram
- In the homework and exams, you must not put such constraints in annotations just to make sure that you realize which constraints are already reflected in the diagram

- ***Annotations are needed so that whoever converts an ER diagram + annotations into a relational schema + annotations can account for all the constraints imposed on the application***

Hierarchy for Our ER Diagram

- There is a natural hierarchy for our ER diagram
- It shows us going from bottom to top how the ER diagram was constructed
- Section and Offered have to constructed together as there is a circular dependency between them
- Similar issue comes up when dealing with ISA

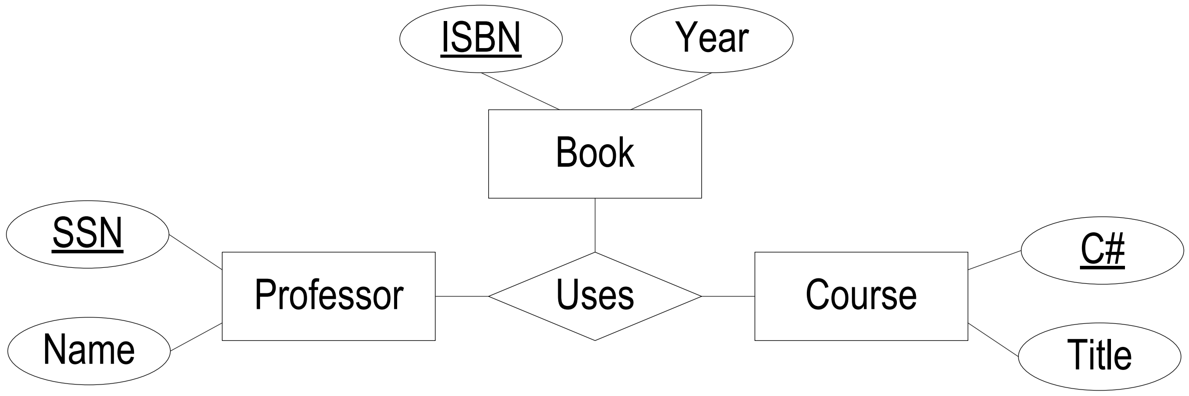
Hierarchy for Our ER Diagram


Monitors
Took
Taught
Required
Type
Has
Likes
Student ISA
Professor ISA
Section Offered
Prereq
Note: circular dependency, need to be treated together
Note: circular dependency, need to be treated together
Note: circular dependency, need to be treated together
Car
Automobile
Person
Horse
Course
Book

Many-To-One Non-Binary Relationships

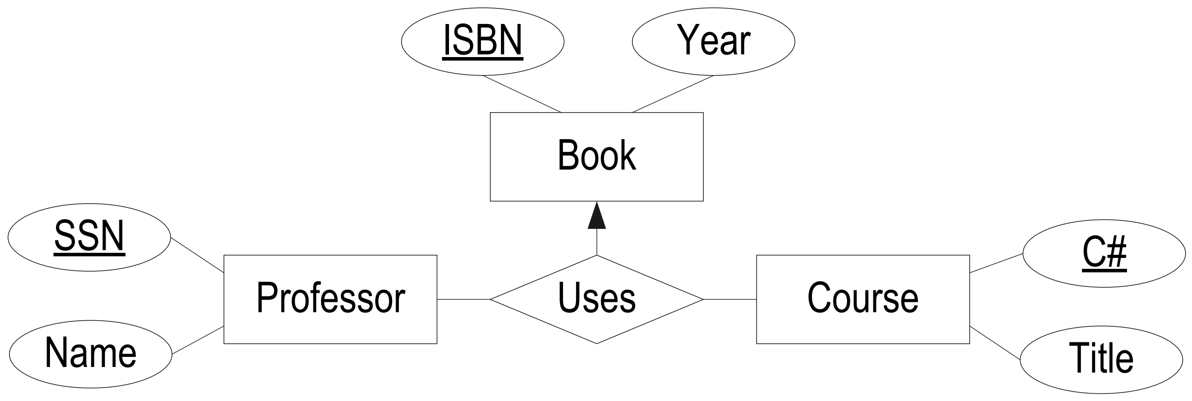
Elaboration on Many-To-One Relationships

- We will look at an example similar to a small part of our large ER diagram
- We want to specify that professors use books in courses
- Generally there are no constraints and
  - A book can be used by any professor in any course as specified in an instance of the database
- We will use here the arrows notation



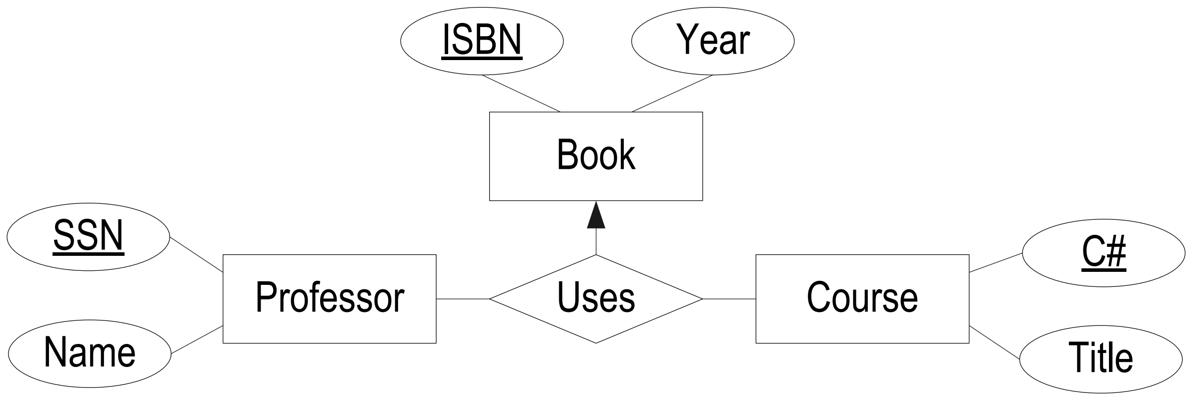
Elaboration on Many-To-One Relationships

- We want to specify that professors use books in courses
- There is a constraint
  - A professor can use only one book in a course
- Therefore, Book is a (partial) function of both Professor and Course
  - Given a pair (p,c), where p in Professor and c in Course there is at most one b in Book



Elaboration on Many-To-One Relationships

- We want to specify that professors use books in courses
- There is a constraint
  - Only one book can used in a course
- Therefore, Book is a (partial) function of Course (only)
  - Given c in Course there is at most one b in Book
- ***We need to annotate the diagram below*** that in the ternary relationship (Professor, Course, Book) the many-to-one relationship (indicated by an arrow) is between Course and Book, as otherwise we will think it is between (Professor, Course) and Book



Key Ideas

Key Ideas

- ER diagrams
- Entity and Entity Set
- Attribute
  - Base
  - Derived
  - Simple
  - Composite
  - Singlevalued
  - Multivalued
- Superkey
- Key
- Candidate Key
- Primary Key
- UNIQUE

Key Ideas

- Relationship
- Binary relationship and its functionality
- Non-binary relationship
- Relationship with attributes
- Aggregation
- Strong and weak entities
- Discriminant
- ISA
  - Disjoint
  - Overlapping
  - Total
  - Partial
  - Specialization
  - Generalization

Key Ideas

- General cardinality constraints
- Using drawing software
- Case study of modeling
- Elaboration on many-to-one relationships (partial functions)