



# Database Systems

## Session 4 – Main Theme

### Practical Relational Database Design

Dr. Jean-Claude Franchitti

New York University  
Computer Science Department  
Courant Institute of Mathematical Sciences

*Presentation material partially based on textbook slides  
Fundamentals of Database Systems (7<sup>th</sup> Edition)  
by Ramez Elmasri and Shamkant Navathe  
Slides copyright © 2022*

# Agenda

1 Session Overview

2 ER and EER to Relational Mapping

3 Database Design Methodology and UML

4 Mapping Relational Design to ER/EER Case Study

5 Comparing Notations and Other Topics

6 Summary and Conclusion

# Session Agenda

- Session Overview
- ER and EER to Relational Mapping
- Database Design Methodology and UML
- Mapping Relational Design to ER/EER Case Study
- Summary & Conclusion

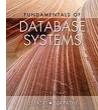
# What is the class about?

- Course description and syllabus:

- » <http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001>
- » <http://cs.nyu.edu/courses/fall22/CSCI-GA.2433-001/>

- Textbooks:

- » *Fundamentals of Database Systems (7<sup>th</sup> Edition)*



Ramez Elmasri and Shamkant Navathe

Addision Wesley

ISBN-10: 0133970779, ISBN-13: 978-0133970777 7<sup>th</sup> Edition (06/18/15)

# Icons / Metaphors



Information



Common Realization



Knowledge/Competency Pattern



Governance



Alignment



Solution Approach

# Agenda

1 Session Overview

2 ER and EER to Relational Mapping

3 Database Design Methodology and UML

4 Mapping Relational Design to ER/EER Case Study

5 Comparing Notations and Other Topics

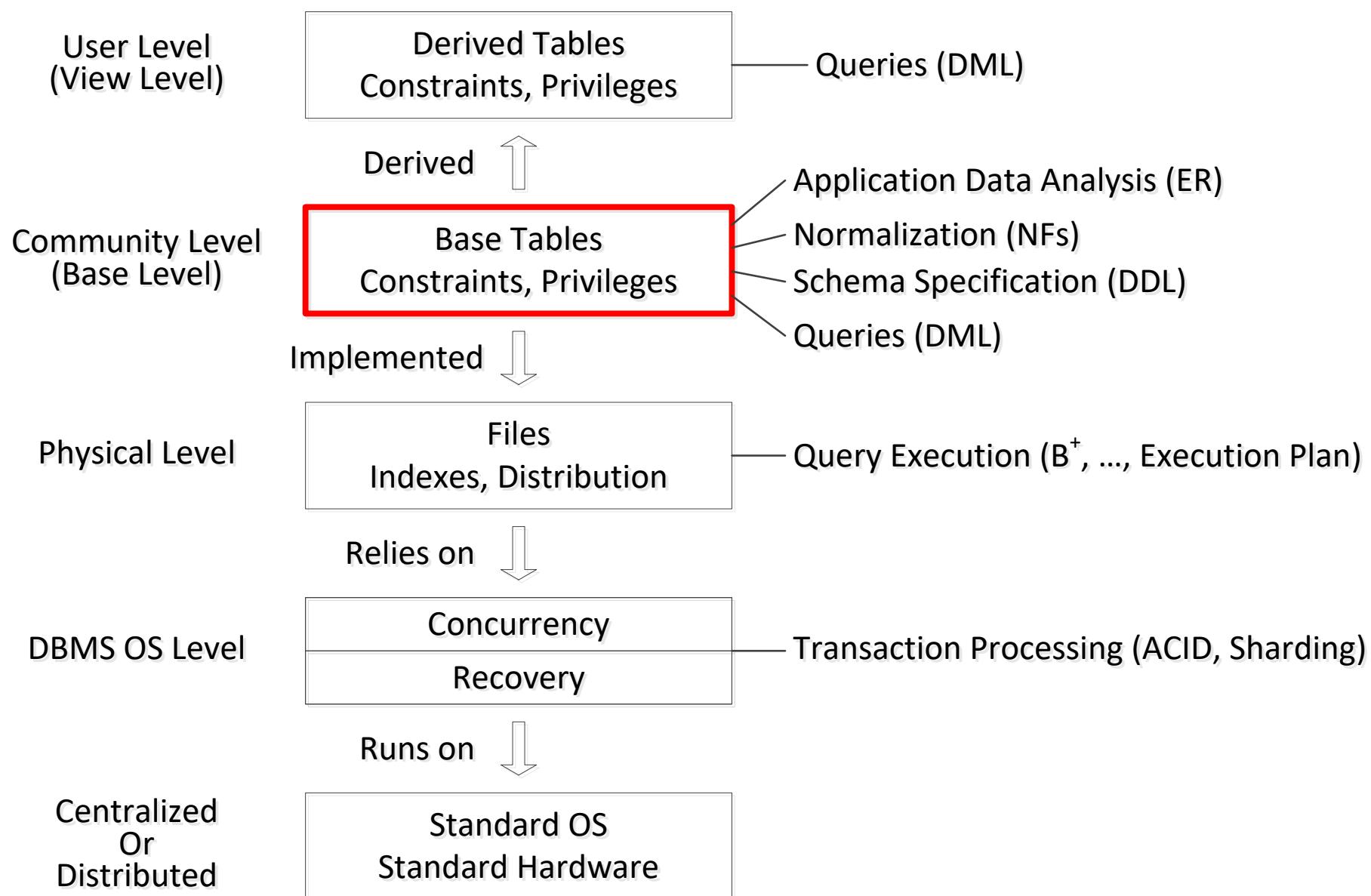
6 Summary and Conclusion





- Introduction
- Sets
- Relations and tables
- Relational schema
- Primary keys
- Relational Database Design Using ER-to-Relational Mapping
- Mapping EER Model Constructs to Relations
- Design a relational database schema
  - Based on a conceptual schema design
- Seven-step algorithm to convert the basic ER model constructs into relations
- Additional steps for EER model

# Relational Tables Implementation in RDMBS Context





# Overview

- In this unit, we learn how to implement an ER diagram as a relational database; we learn the semantics of specifying a relational database, later we will learn the syntax of SQL for doing this
- We call this process ***logical design***
- Sometimes it is called ***physical design***, if the creation of an ER diagram is referred to as logical design
  - » We do not do that
- Some use both “logical design” and “physical design” terms for this process, then
  - » Logical design: relational implementation not targeting any specific DBMS: we will focus on this
  - » Physical design: further development/refinement of a relational implementation targeting a specific DBMs
- Our process will be “almost an algorithm,” but not quite
- Therefore, it is important to understand the reasons for the various steps in the process



# Importance of Good Design

- In contrast, to what is frequently taught, we spend more time on that as opposed to learning how to write SQL statements
- Quoting from [Oracle Database Development Guide](#)

The key to database and application performance is design, not tuning. While tuning is quite valuable, it cannot make up for poor design. Your design must start with an efficient data model, well-defined performance goals and metrics, and a sensible benchmarking strategy. Otherwise, you will encounter problems during implementation, when no amount of tuning will produce the results that you could have obtained with good design. You might have to redesign the system later, despite having tuned the original poor design.

- Measure twice, cut once



# Sets, Relations, and Tables

- The basic “datatype”, or “variable” of a relational database is a ***relation***
- In this unit, such a variable will be a ***set***
- Later, we will extend this, and such a variable will be a ***multiset***
- This distinction is not important for now
- In SQL, such a variable is called a ***table***
- We may use the term “table” and “relation” interchangeably and not pay attention to whether we are talking about a set or a multiset



# Sets

- We will not use axiomatic set theory
- A **set** is a “bag” of elements, some/all of which could be sets themselves and a binary relationship “is element of” denoted by  $\in$ , such as  $2 \in \{2, 5, 3, 7\}$ ,  $\{2, 8\} \in \{2, \{2, 8\}, 5, 3, 7\}$ ,
- You cannot specify
  - » How many times an element appears in a set (if you could, this would be a **multiset**)
  - » In which position an element appears (if you could, this would be a **sequence**)
- Therefore, as sets:  $\{2, 5, 3, 7\} = \{2, 7, 5, 3, 5, 3, 3\}$
- Note: in many places you will read: “an element can appear in a set only once”

This is not quite right. And it is important not to assume that statement, as we will see in a later unit



# Sets

- Two sets  $A$  and  $B$  are equal iff (if and only if) they have the same elements
- In other words, for every  $x$ :  $x$  is an element of  $A$  iff (if and only if)  $x$  is an element of  $B$
- In still a different way:  $A$  and  $B$  are equal iff for every possible  $x$ , the questions “is  $x$  in  $A$ ?” and “is  $x$  in  $B$ ?” give the same answer
  - Note that there is no discussion or even a way of saying how many times  $x$  appears in a set, really either **zero** or **at least once**
- “More mathematically” and recalling that  $\forall x$  means “for all  $x$ ”  
 $\forall x \{ x \in A \Leftrightarrow x \in B \}$  if and only if  $A = B$
- Therefore, as sets:  $\{2, 5, 3, 7\} = \{2, 7, 5, 3, 5, 3, 3\}$ 
  - 3 is in both and 4 is in neither, etc.
- This reiterates what we have said previously
  - You cannot specify multiplicity
  - You cannot specify position



# Relation

- Consider a table, with a fixed number of columns where elements of each column are drawn from some specific domain
- The columns are labeled and the labels are distinct
- We will consider such a table to be **a set of rows** (another word for “row”: **tuple**)
- Here is an example of a table S of two columns A and B

S	A	B
a		2
a		2
b		3
c		4
d		3

- A **relation** is such a table
- We will also write  $S(A,B)$  for table S with columns A and B



# Relational Schema

- What we saw was an ***instance*** (current value for a relation with the defined columns and domains)
- To specify this relation in general (not the specific instance) we need to talk about a ***relational schema***
- A relational schema defines a set of relations
- In databases everything is finite, so a relational schema defines a finite set of finite relations



# Relational Schema

- Here is an informal, but complete, description of what is a relational schema of one relation
- We want to define a structure for some table
  1. We give it a name (we had S)
  2. We chose the number of columns (we had 2) and give them distinct names (we had A and B)
  3. We decide on the domains of elements in the columns (we had letters for A and integers for B)
  4. We decide on constraints, if any, on the permitted values (for example, we can assume as it was true for our example that any two rows that are equal on A must be equal on B)

***This part must not be omitted if such constraints hold!***



# Relational Schema

- Let's verify
  - »  $A$  all lower case letters in English
  - »  $B$  all positive integers less than 100
  - »  $S(A,B)$  satisfies the condition that any two tuples that are equal on  $A$  must also be equal on  $B$
- Our example was an instance of this relational schema

S	A	B
a		2
a		2
b		3
c		4
d		3



## Relations (1/8)

- Since relations are *sets* of tuples, ***the following two relations are equal*** (are really one relation written in two different ways)  
(This is a different example, not an instance of the previous relational schema)

S	A	B
a		2
a		56
b		2

S	A	B
a		56
a		2
b		2
a		56
a		2
a		56



## Relations (2/8)

- Since ***the positions*** in the tuple (1<sup>st</sup>, 2<sup>nd</sup>, etc.) are labeled with the column headings, ***the following two relations are equal*** (are really one relation written in two different ways)

S	A	B
a		2
a		56
b		2

S	B	A
56	a	
2	a	
2	b	
56	a	
2	a	
56	a	



## Relations (3/8)

- To specify relations, it is enough to do what we have done above
- If we understand what are the domains for the columns, the following are formally fully specified relations
  - » Relational (schema) P(Name, SSN, DOB, Grade) with some (not specified, but we should have done it) domains for attributes
  - » Relational (schema) Q(Grade, Salary) with some (not specified, but we should have done it) domains for attributes

P	Name	SSN	DOB	Grade
	A	121	2367	2
	B	101	3498	4
	C	106	2987	2

Q	Grade	Salary
1		90
2		80
3		70
4		70



# NULLs

- Each domain is augmented with a NULL
- NULL can now be considered as a synonym for “unknown”
- We will discuss NULLs in greater depth later in the course



## Keys (1/4)

- We will specify, as appropriate for the schema:
  - » **Primary keys**
  - » **Keys** (beyond primary)
  - » **Foreign keys** and what they reference (we will see soon what this means)
- The above are most important **structurally**
- Later, especially when we talk about SQL DDL, we will specify additional properties and constraints
  - » For example, the height of a person must be positive, if it is known
- Some of the constraints may involve more than one relation



## Keys (2/8)

- Consider relation (schema) Person(FN, LN, Grade, YOB)
- Instance:

Person	FN	LN	Grade	YOB
	John	Smith	8	1976
	Lakshmi	Smith	9	1981
	John	Smith	8	1976
	John	Yao	9	1992

- We are told that any two tuples that are equal on both FN and LN are (completely) equal
  - » We have some tuples appearing multiple times: this is just for clarifying that this is permitted in the definition, we do not discuss here why we would have the same tuple more than one time (we will talk about this later)
- This is a property of **every possible instance** of Person in our application—we are told this
- Then (FN, LN) is a **key** of Person, as this is a minimal set of attributes whose values identify a unique person
- What does this mean exactly?



## Keys (3/8)

- Given some relation R (here Person) as set of attributes, say S (here FN and LN) form a key if and only if
  - » Any two tuples of R that are equal on all the attributes of S are (completely) equal
  - » No proper subset of S has this property (the minimality property)
- To reiterate completely for our example
  - » Any two tuples that are equal on FN and LN are equal
  - » No proper subset of the set FN, LN (formally the set {FN, LN}) has this property

Indeed, there are tuples that equal on a proper subset FN that are not (completely) equal

Indeed, there are tuples that equal on a proper subset LN that are not (completely) equal

- » The three attributes FN, LN, Grade do not form a key as the two attributes FN, LN already form a key (Grade is “too much”)



## Keys (4/4)

- Consider relation (schema) Q(Grade, Salary)
- Example:

Pay	Grade	Salary
	8	128
	9	139
	7	147

- We are told that for any instance of Pay, any two tuples that are equal on Grade are (completely) equal
  - » Of course, if each Grade appears in only one tuple, this is automatically true
- Then, similarly to before, Grade is a key
- What about Salary, is this a key also?
- No, because we ***are not told (that is, we are not guaranteed)*** that any two tuples that are equal on Salary are equal on Grade in every instance of Pay



# Keys (and Superkeys)

- A set of columns in a relation is a **superkey** if it is a superset of a key
- Any two tuples that are equal on the attributes of a superkey are completely equal
  - Because they are equal on the attributes of a key, which is a subset of the given superkey
- **A relation always has at least one superkey**
- The set of all the attributes is a superkey
- Because any two tuples that are equal on all attributes are completely equal
- A minimal superkey, is a key
  - We can try and “prune” the set of all the attributes “down” to a key
  - We just try to remove attributes one by one
  - We stop when removing an attribute would produce a set of attributes whose values “are compatible” with two different tuples



# Keys (and Superkeys)

- There may be more than one key in a relation, as we will see soon
- Exactly one key is chosen as the ***primary key*** (there is no formal way to decide which one)
- Other keys are just keys and sometimes they are called ***candidate keys*** (as they are candidates for the primary key, though not chosen)
- Customarily, the names of the columns of the primary key are underlined

Person	<u>FN</u>	LN	Grade	YOB
John	John	Smith	8	1976
	Lakshmi	Smith	9	1981
	John	Smith	8	1976
	John	Yao	9	1992

- Note that FN, LN, Grade was as superkey but not a key



# Keys (and Superkeys)

- To summarize
- **Superkey**: a set of columns whose values determine the values of all the columns in a row for every instance of the relation
  - » All the columns together form a superkey (trivially)
  - » Superkeys are only used in formal definitions
- **Key**: a set of columns whose values determine the values of all the columns in a row for every instance of the relation, ***but any proper subset of that set of columns does not do that***
- **Primary Key**: a chosen key. ***Important: no attribute of a primary key can be NULL (the value of every attribute must be always known); why will be discussed later***



# Keys (and Superkeys)

- We will underline the attributes of the chosen primary key when writing the schema as text
- Returning to a previous unit and the example of City: City(Longitude, Latitude, Country, State, Name, Size)
- We can have
  - » City(Longitude, Latitude, Country, State, Name, Size)
  - » This implies that (Longitude, Latitude) form the primary key
  - » We also have a candidate key: (Country, State, Name)
- We can have
  - » City(Longitude, Latitude, Country, State, Name, Size)
    - if the value of State is never NULL. This implies that (Country, State, Name) form the primary key
    - » We also have a candidate key: (Longitude, Latitude)
- The situation will be described more precisely concerning candidate keys when we talk about SQL DDL



# Relational Databases

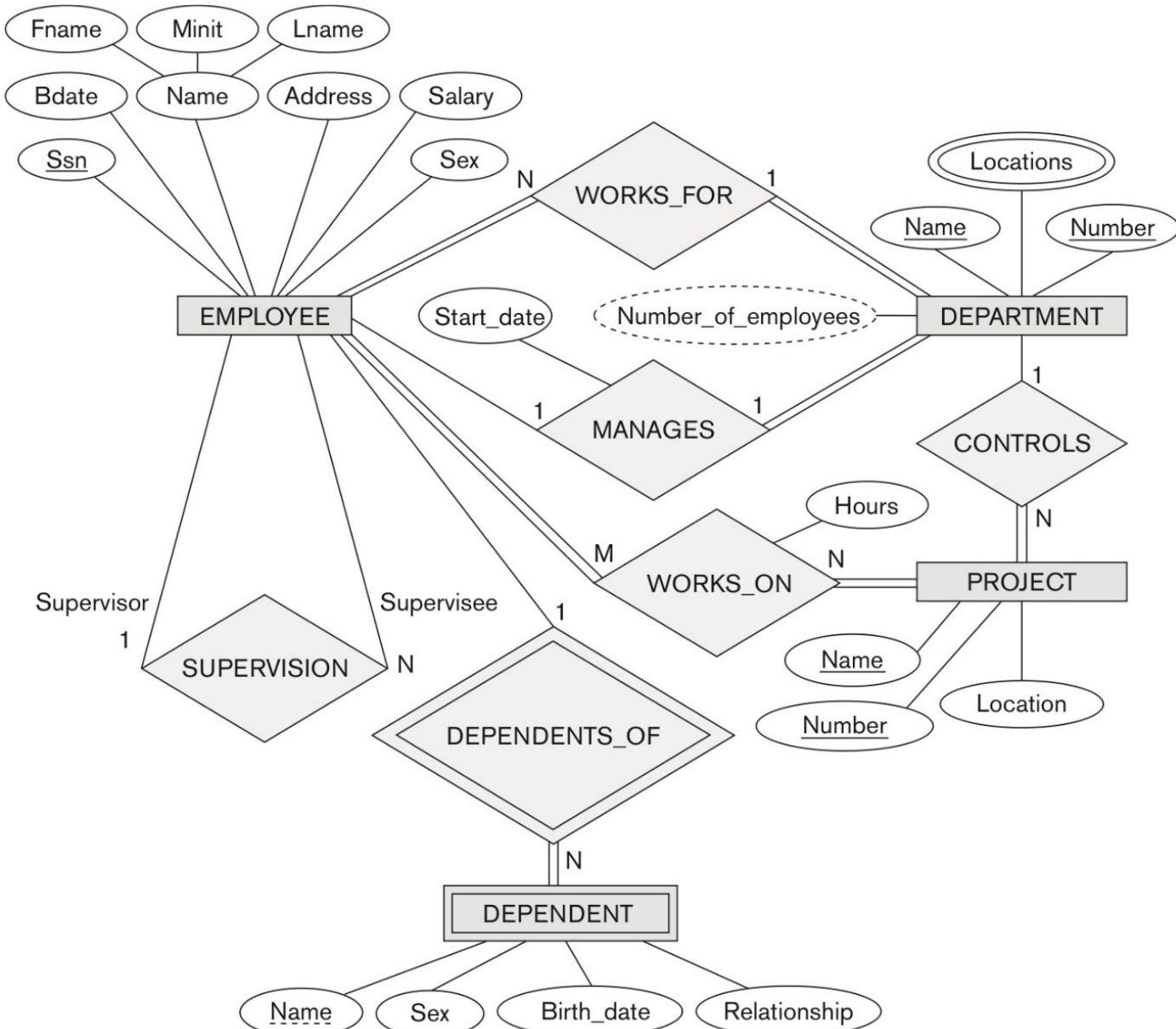
- A **relational database** is basically a set of relations and is an instance of a relational schema
- This is formally correct, ***but it is misleading in practice***
- **As we will see soon, a better way to define a relational database is to say that it is**
  - » **A set of relations**
  - » **A set of binary, many-to-one mappings between them (partial functions), mimicking child-to-mother relationships**

**Looking from the “other side,” they are one-to-many mappings, mimicking mother-to-child relationships**

- » Some other, less fundamental, components are ignored for now
- We will know later what this means exactly, but I did not want to leave you with an overly simplistic concept
- Note: “**partial function**” means “not necessarily defined everywhere”, but could be and then it is also “**total**”



# ER conceptual schema diagram for the COMPANY database





## GOALS during Mapping

- Preserve all information (that includes all attributes)
- Maintain the constraints to the extent possible (Relational Model cannot preserve all constraints- e.g., max cardinality ratio such as 1:10 in ER; exhaustive classification into subtypes, e.g., STUDENTS are specialized into Domestic and Foreign)
- Minimize null values

*The mapping procedure described has been implemented in many commercial tools.*



# Mapping Steps

- **ER-to-Relational Mapping Algorithm**
  - » Step 1: Mapping of Regular Entity Types
  - » Step 2: Mapping of Weak Entity Types
  - » Step 3: Mapping of Binary 1:1 Relation Types
  - » Step 4: Mapping of Binary 1:N Relationship Types.
  - » Step 5: Mapping of Binary M:N Relationship Types.
  - » Step 6: Mapping of Multivalued attributes.
  - » Step 7: Mapping of N-ary Relationship Types.
- **Mapping EER Model Constructs to Relations**
  - » Step 8: Options for Mapping Specialization or Generalization.
  - » Step 9: Mapping of Union Types (Categories).



# ER-to-Relational Mapping Algorithm (1/9)

- COMPANY database example
  - Assume that the mapping will create tables with simple single-valued attributes
- Step 1: Mapping of Regular Entity Types.
  - » For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
  - » Choose one of the key attributes of E as the primary key for R.
  - » If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.
- Example: We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram.
  - » SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown.



# ER-to-Relational Mapping Algorithm (2/9)

## ▪ Step 2: Mapping of Weak Entity Types

- » For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R.
- » Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
- » The primary key of R is the *combination* of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

## ▪ Example: Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT.

- » Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).
- » The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT\_NAME} because DEPENDENT\_NAME is the partial key of DEPENDENT.



# ER-to-Relational Mapping Algorithm (3/9)

**Figure 9.3**

Illustration of some mapping steps.

- a. *Entity* relations after step 1.
- b. Additional weak entity relation after step 2.
- c. *Relationship* relation after step 5.
- d. Relation representing multivalued attribute after step 6.

(a) **EMPLOYEE**

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary
-------	-------	-------	------------	-------	---------	-----	--------

**DEPARTMENT**

Dname	<u>Dnumber</u>
-------	----------------

**PROJECT**

Pname	<u>Pnumber</u>	Plocation
-------	----------------	-----------

(b) **DEPENDENT**

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

(c) **WORKS\_ON**

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

(d) **DEPT\_LOCATIONS**

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------



# ER-to-Relational Mapping Algorithm (4/9)

- **Step 3: Mapping of Binary 1:1 Relation Types**
  - » For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.
- There are three possible approaches:
  1. **Foreign Key ( 2 relations) approach:** Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.
    - Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.
  2. **Merged relation (1 relation) option:** An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.
  3. **Cross-reference or relationship relation ( 3 relations) option:** The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.



## ER-to-Relational Mapping Algorithm (5/9)

- Step 4: Mapping of Binary 1:N Relationship Types.
  - » For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.
  - » Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
  - » Include any simple attributes of the 1:N relation type as attributes of S.
- Example: 1:N relationship types WORKS\_FOR, CONTROLS, and SUPERVISION in the figure.
  - » For WORKS\_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.
- An alternative approach is to use a Relationship relation (cross referencing relation) as in the third option for binary 1:1 relationships – this is rarely done.



# ER-to-Relational Mapping Algorithm (7/9)

- **Step 5: Mapping of Binary M:N Relationship Types.**
  - » For each regular binary M:N relationship type R, *create a new relation S to represent R. This is a relationship relation.*
  - » Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; *their combination will form the primary key of S.*
  - » Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.
- Example: The M:N relationship type WORKS\_ON from the ER diagram is mapped by creating a relation WORKS\_ON in the relational database schema.
  - » The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS\_ON and renamed PNO and ESSN, respectively.
  - » Attribute HOURS in WORKS\_ON represents the HOURS attribute of the relation type. The primary key of the WORKS\_ON relation is the combination of the foreign key attributes {ESSN, PNO}.



# ER-to-Relational Mapping Algorithm (8/9)

- **Step 6: Mapping of Multivalued attributes.**
  - » For each multivalued attribute A, create a new relation R.
  - » This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type or relationship type that has A as an attribute.
  - » The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.
- **Example:** The relation DEPT\_LOCATIONS is created.
  - » The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation.
  - » The primary key of R is the combination of {DNUMBER, DLOCATION}.



# ER-to-Relational Mapping Algorithm (9/9)

- **Step 7: Mapping of N-ary Relationship Types.**
  - » For each n-ary relationship type R, where  $n > 2$ , create a new relationship S to represent R.
  - » Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
  - » Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.
- **Example:** The relationship type SUPPLY in the ER on the next slide.
  - » This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}



# Result of Mapping the COMPANY ER Schema into a RDB Schema

## EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

## DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

## DEPT\_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

## PROJECT

Pname	<u>Pnumber</u>	<u>Plocation</u>	Dnum
-------	----------------	------------------	------

## WORKS\_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

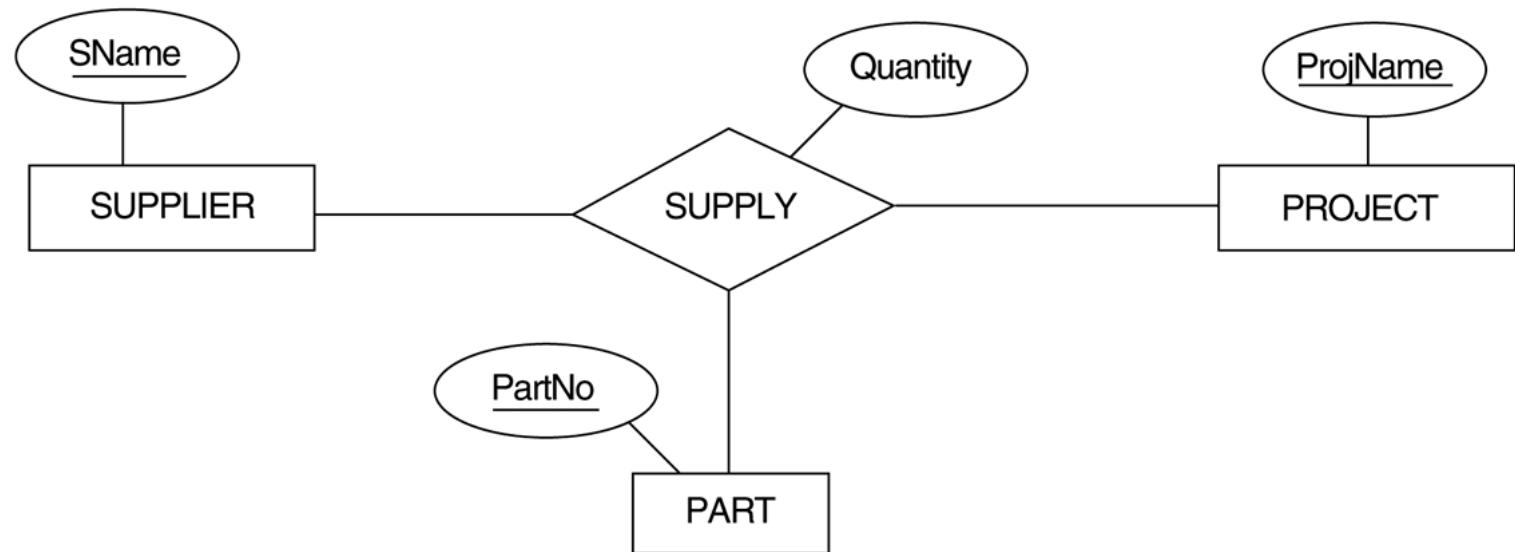
## DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

# Ternary Relationship: SUPPLY

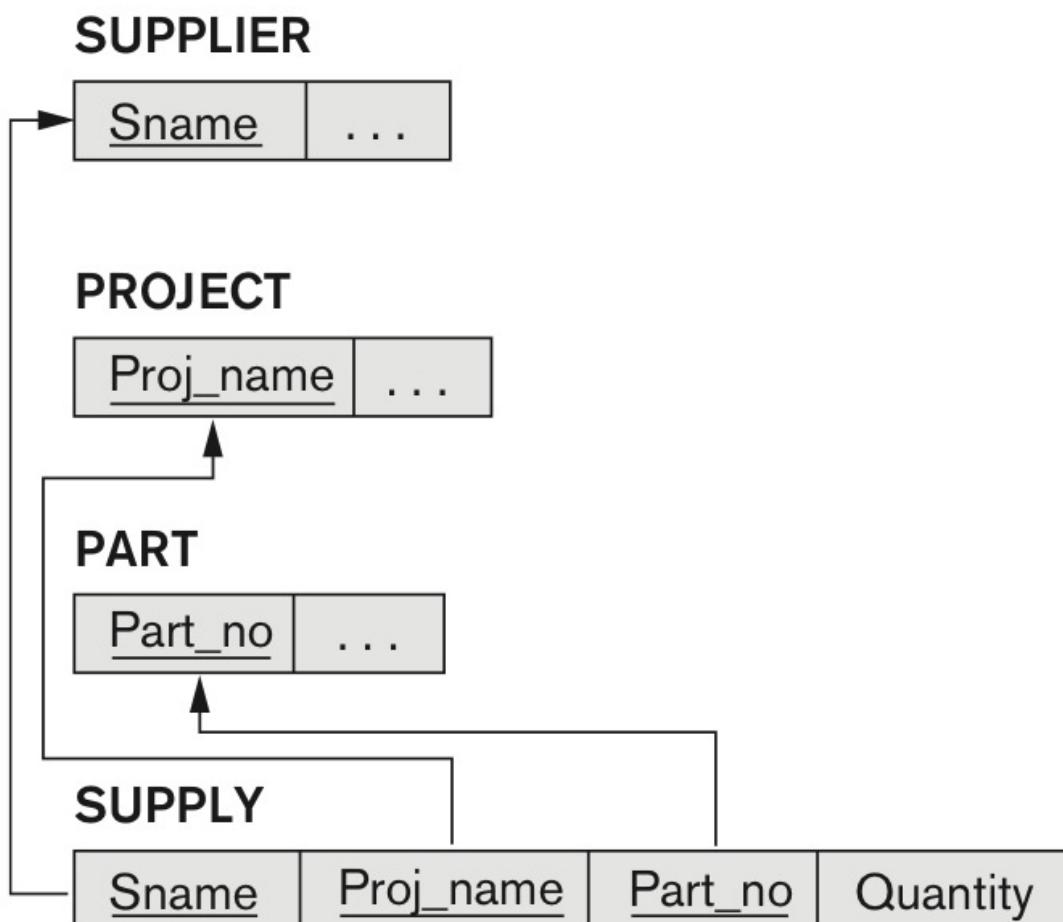


(a)





# Mapping the n-ary Relationship Type SUPPLY





# Discussion and Summary of Mapping for ER Model Constructs (1/2)

**Table 9.1** Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	<i>Entity</i> relation
1:1 or 1:N relationship type	Foreign key (or <i>relationship</i> relation)
M:N relationship type	<i>Relationship</i> relation and <i>two</i> foreign keys
<i>n</i> -ary relationship type	<i>Relationship</i> relation and <i>n</i> foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key



- In a relational schema relationship, types are not represented explicitly
  - Represented by having two attributes  $A$  and  $B$ : one a primary key and the other a foreign key



- **Step8: Options for Mapping Specialization or Generalization**

- » Convert each specialization with m subclasses {S1, S2, ..., Sm} and generalized superclass C, where the attributes of C are {k, a1, ... an} and k is the (primary) key, into relational schemas using one of the four following options:
  - Option 8A: Multiple relations-Superclass and subclasses
  - Option 8B: Multiple relations-Subclass relations only
  - Option 8C: Single relation with one type attribute
  - Option 8D: Single relation with multiple type attributes



# Mapping of Specialization or Generalization (1/2)

- **Option 8A: Multiple relations-Superclass and subclasses**
  - » Create a relation L for C with attributes  $\text{Attrs}(L) = \{k, a_1, \dots, a_n\}$  and  $\text{PK}(L) = k$ . Create a relation  $L_i$  for each subclass  $S_i$ ,  $1 < i < m$ , with the attributes  $\text{Attrs}(L_i) = \{k\} \cup \{\text{attributes of } S_i\}$  and  $\text{PK}(L_i)=k$ . This option works for any specialization (total or partial, disjoint or overlapping)
- **Option 8B: Multiple relations-Subclass relations only**
  - » Create a relation  $L_i$  for each subclass  $S_i$ ,  $1 < i < m$ , with the attributes  $\text{Attr}(L_i) = \{\text{attributes of } S_i\} \cup \{k, a_1, \dots, a_n\}$  and  $\text{PK}(L_i) = k$ . This option only works for a specialization whose subclasses are total (every entity in the superclass must belong to (at least) one of the subclasses)



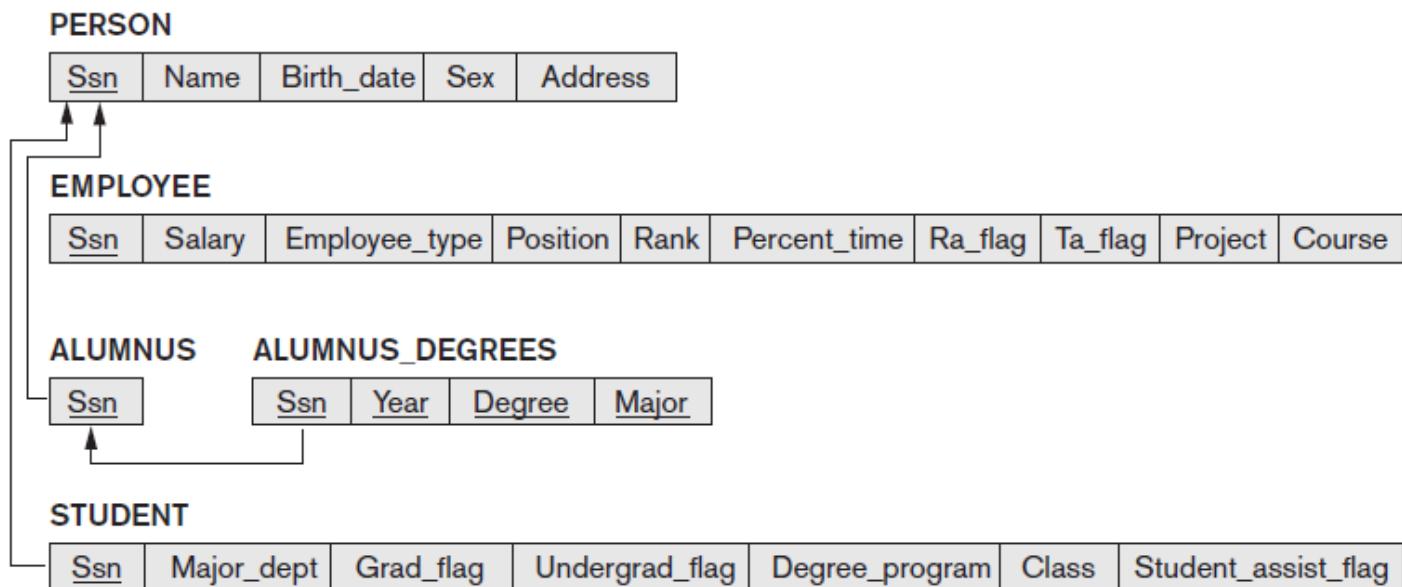
## Mapping of Specialization or Generalization (2/2)

- **Option 8C: Single relation with one type attribute**
  - » Create a single relation L with attributes  $\text{Attrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{attributes of } S_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{t\}$  and  $\text{PK}(L) = k$ .  
The attribute t is called a type (or **discriminating**) attribute that indicates the subclass to which each tuple belongs
- **Option 8D: Single relation with multiple type attributes**
  - » Create a single relation schema L with attributes  $\text{Attrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{attributes of } S_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{t_1, t_2, \dots, t_m\}$  and  $\text{PK}(L) = k$ . Each  $t_i$ ,  $1 < i < m$ , is a Boolean type attribute indicating whether a tuple belongs to the subclass  $S_i$



# Mapping of Shared Subclasses (Multiple Inheritance)

- Apply any of the options discussed in step 8 to a shared subclass



**Figure 9.6**

Mapping the EER specialization lattice in Figure 8.8 using multiple options.



# Different Options for Mapping Generalization Hierarchies

- See next slide options for mapping specialization or generalization
  - » (a) Mapping the EER schema in Figure 4.4 using option 8A
  - » (b) Mapping the EER schema in Figure 4.3(b) using option 8B
  - » (c) Mapping the EER schema in Figure 4.4 using option 8C
  - » (d) Mapping Figure 4.5 using option 8D with Boolean type fields Mflag and Pflag

# EER Diagram Notation for Attribute-Defined Specialization on JobType

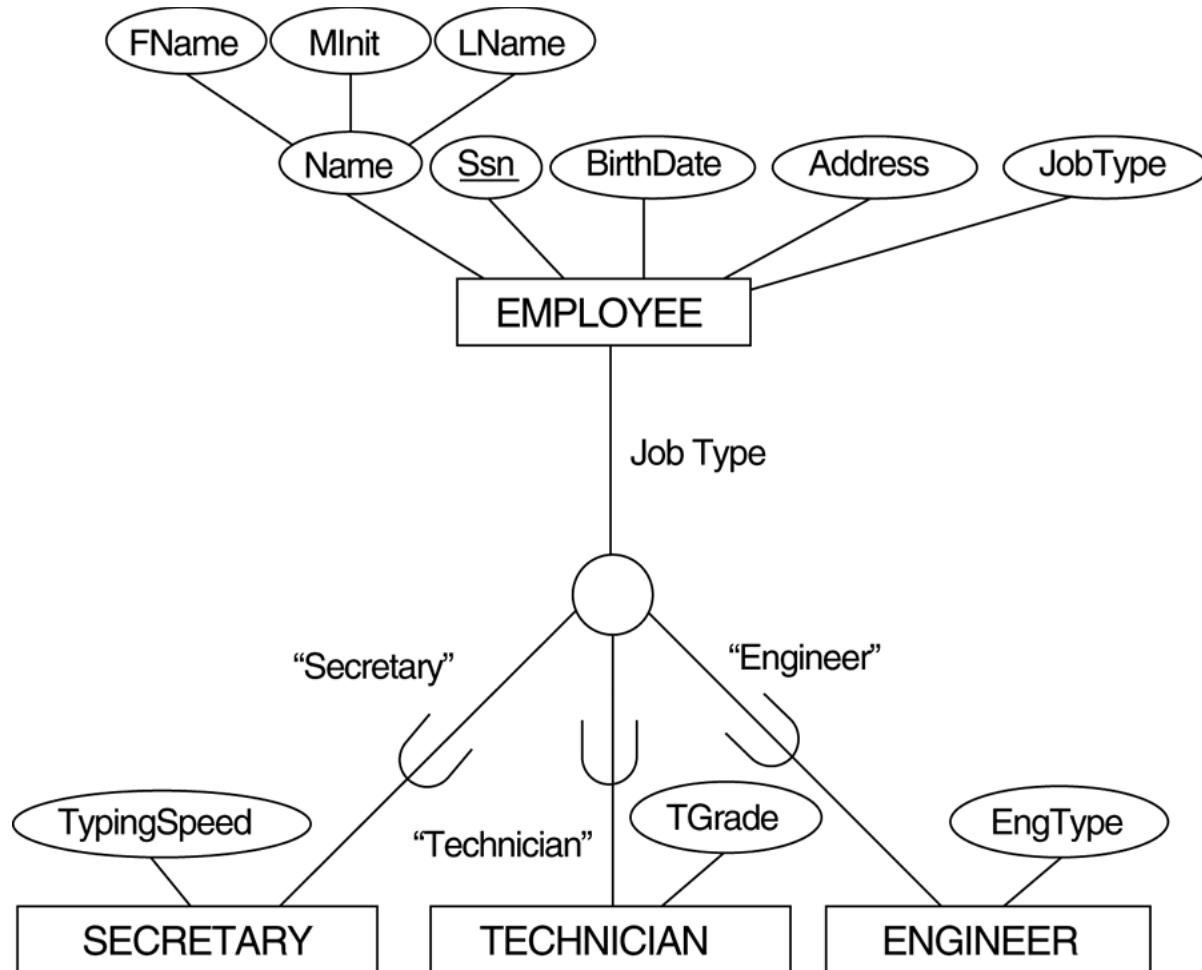


Figure 4.4



# Mapping the EER schema in Figure 4.4 using option 8A

(a) **EMPLOYEE**

<u>SSN</u>	FName	MInit	LName	BirthDate	Address	JobType
------------	-------	-------	-------	-----------	---------	---------

**SECRETARY**

<u>SSN</u>	TypingSpeed
------------	-------------

**TECHNICIAN**

<u>SSN</u>	TGrade
------------	--------

**ENGINEER**

<u>SSN</u>	EngType
------------	---------



# Mapping the EER schema in Figure 4.4 using option 8C

(c) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	
-----	-------	-------	-------	-----------	---------	---------	-------------	--------	--

# Generalizing CAR and TRUCK into the superclass VEHICLE

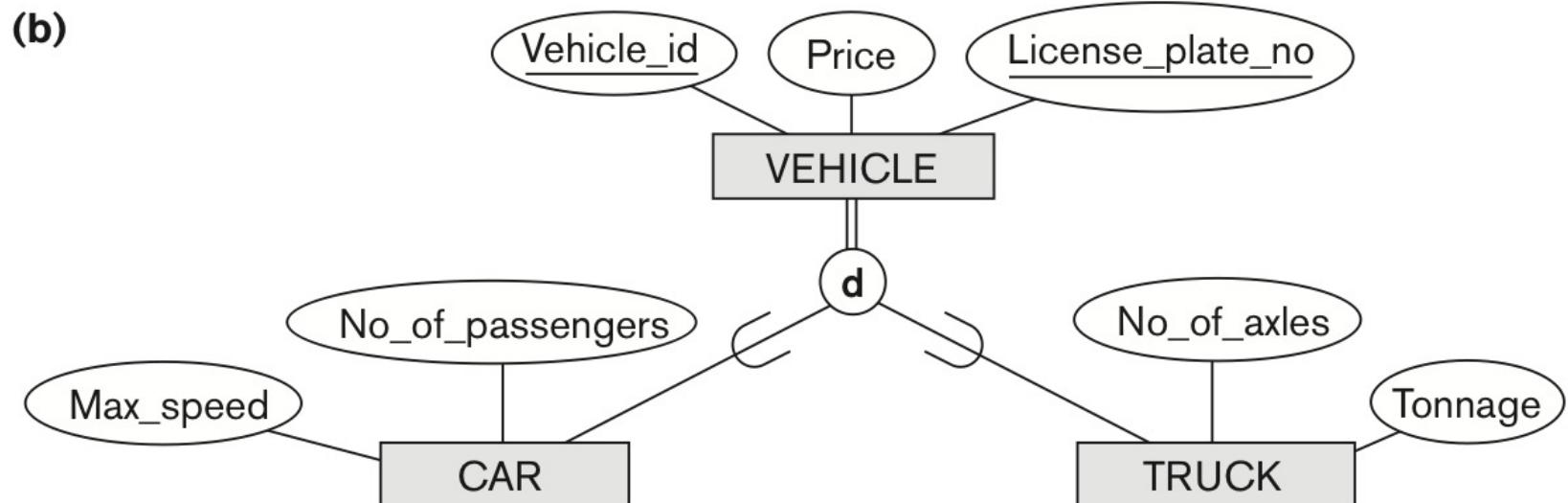
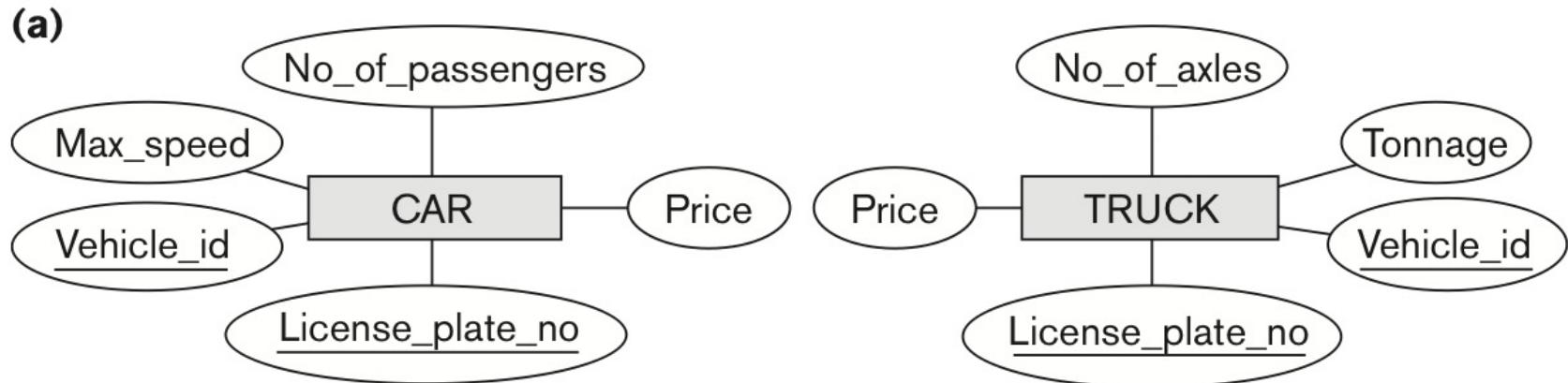


Figure 4.3 (b)



# Mapping the EER schema in Figure 4.3 (b) using option 8B

(b) CAR

<u>VehicleId</u>	LicensePlateNo	Price	MaxSpeed	NoOfPassengers
------------------	----------------	-------	----------	----------------

TRUCK

<u>VehicleId</u>	LicensePlateNo	Price	NoOfAxles	
------------------	----------------	-------	-----------	--



# An overlapping (non-disjoint) specialization.

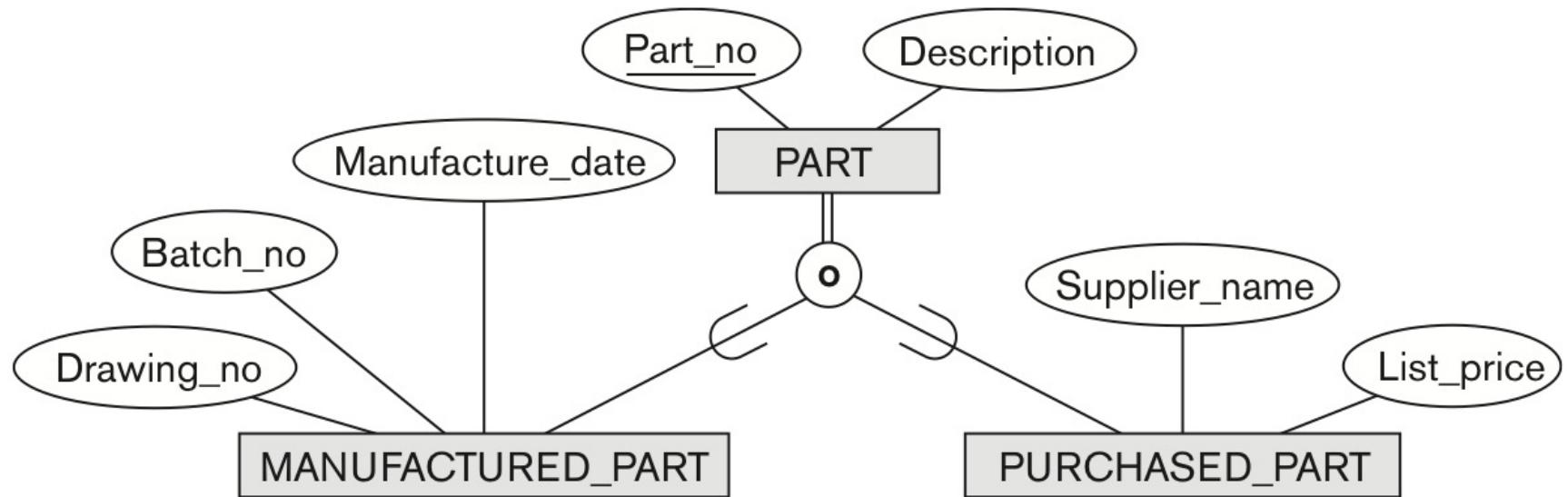


Figure 4.5

# Mapping Fig. 4.5 via option 8B with Boolean Type Fields Mflag & Pflag



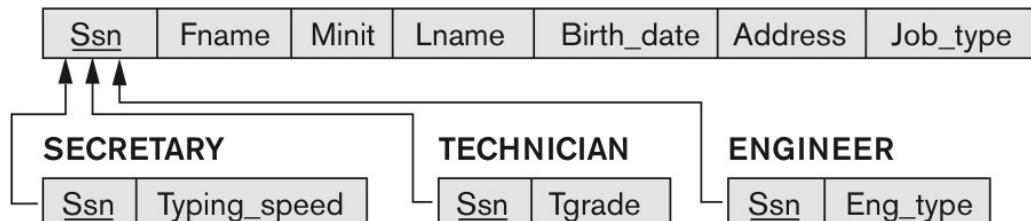
(d) PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice
--------	-------------	-------	-----------	-----------------	---------	-------	--------------	-----------



# Different Options for Mapping Generalization Hierarchies - Summary

## (a) EMPLOYEE



## (b) CAR

<u>Vehicle_id</u>	License_plate_no	Price	Max_speed	No_of_passengers
-------------------	------------------	-------	-----------	------------------

### TRUCK

<u>Vehicle_id</u>	License_plate_no	Price	No_of_axles	Tonnage
-------------------	------------------	-------	-------------	---------

## (c) EMPLOYEE

<u>Ssn</u>	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type
------------	-------	-------	-------	------------	---------	----------	--------------	--------	----------

## (d) PART

<u>Part_no</u>	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price
----------------	-------------	-------	------------	------------------	----------	-------	---------------	------------

Figure 9.5



# Mapping EER Model Constructs to Relations

- Mapping of Shared Subclasses (Multiple Inheritance)
  - » A shared subclass, such as STUDENT\_ASSISTANT, is a subclass of several classes, indicating multiple inheritance. These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category.
  - » We can apply any of the options discussed in Step 8 to a shared subclass, subject to the restriction discussed in Step 8 of the mapping algorithm. Below both 8C and 8D are used for the shared class STUDENT\_ASSISTANT.

# Specialization Lattice with Multiple Inheritance for UNIVERSITY DB

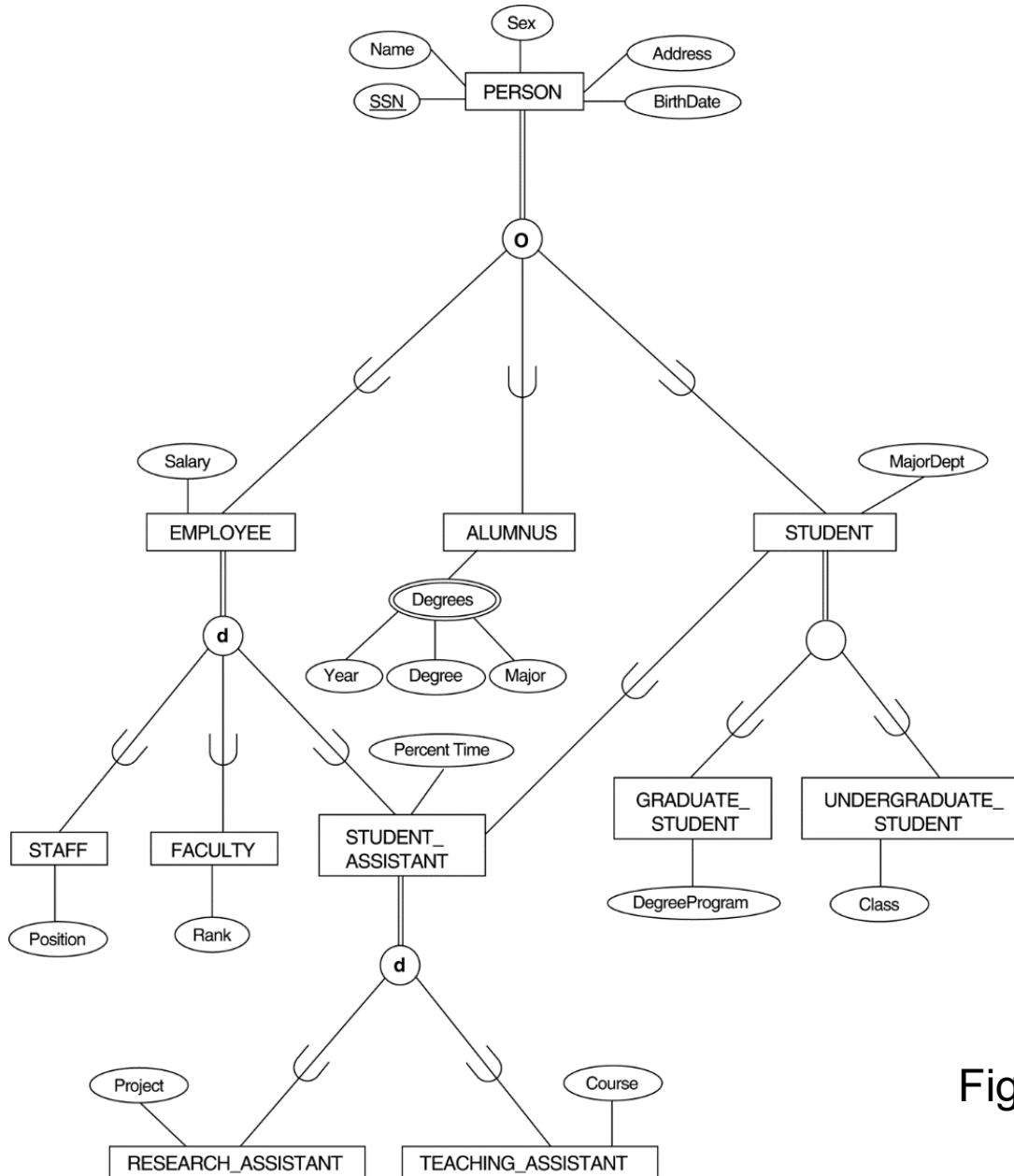
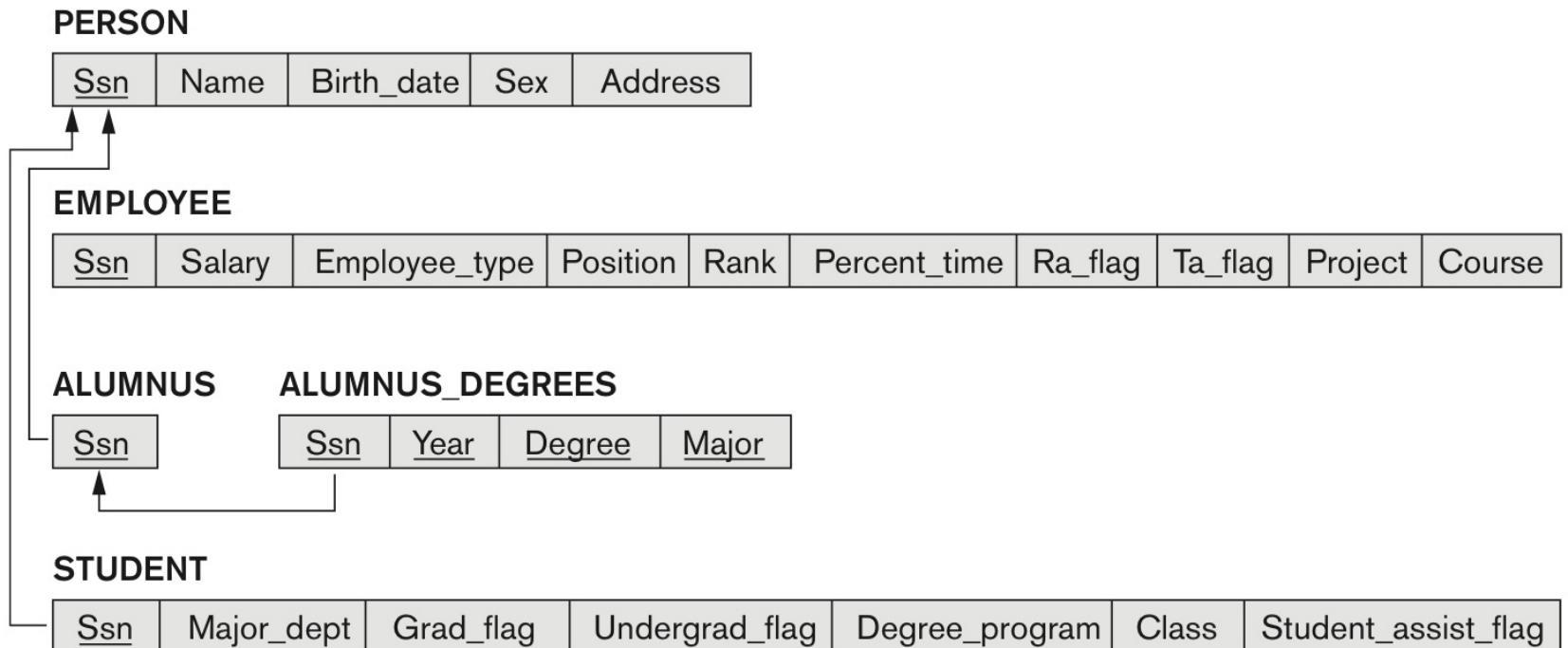


Figure 4.7

# Mapping EER Specialization lattice in Figure 4.7 Using Multiple Options





## Mapping of Categories (Union Types)

- **Step 9: Mapping of Union Types (Categories).**

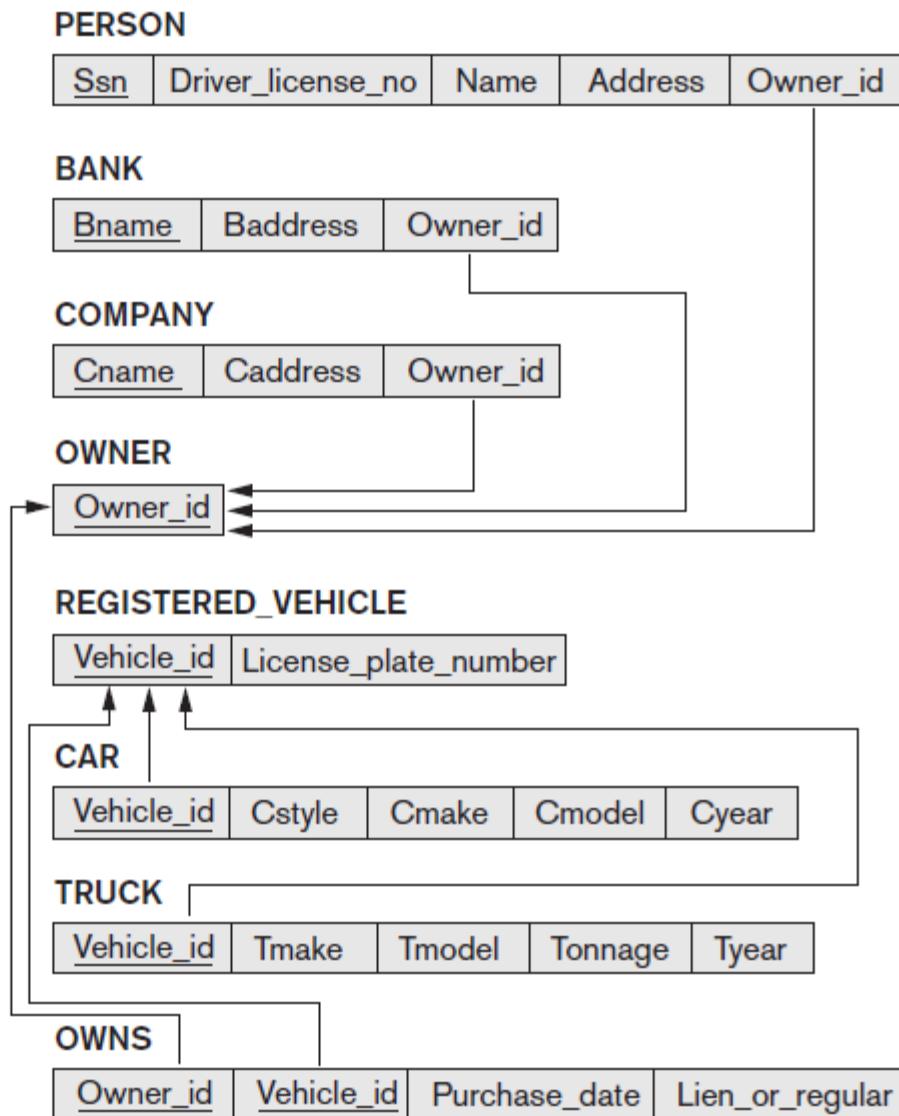
- » For mapping a category whose defining superclass have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category.
- » In the example below we can create a relation OWNER to correspond to the OWNER category and include any attributes of the category in this relation. The primary key of the OWNER relation is the surrogate key, which we called OwnerId.



# Sample Mapping of EER Categories to Relations

**Figure 9.7**

Mapping the EER categories (union types) in Figure 8.8 to relations.





## Two categories (union types): OWNER and REGISTERED\_VEHICLE

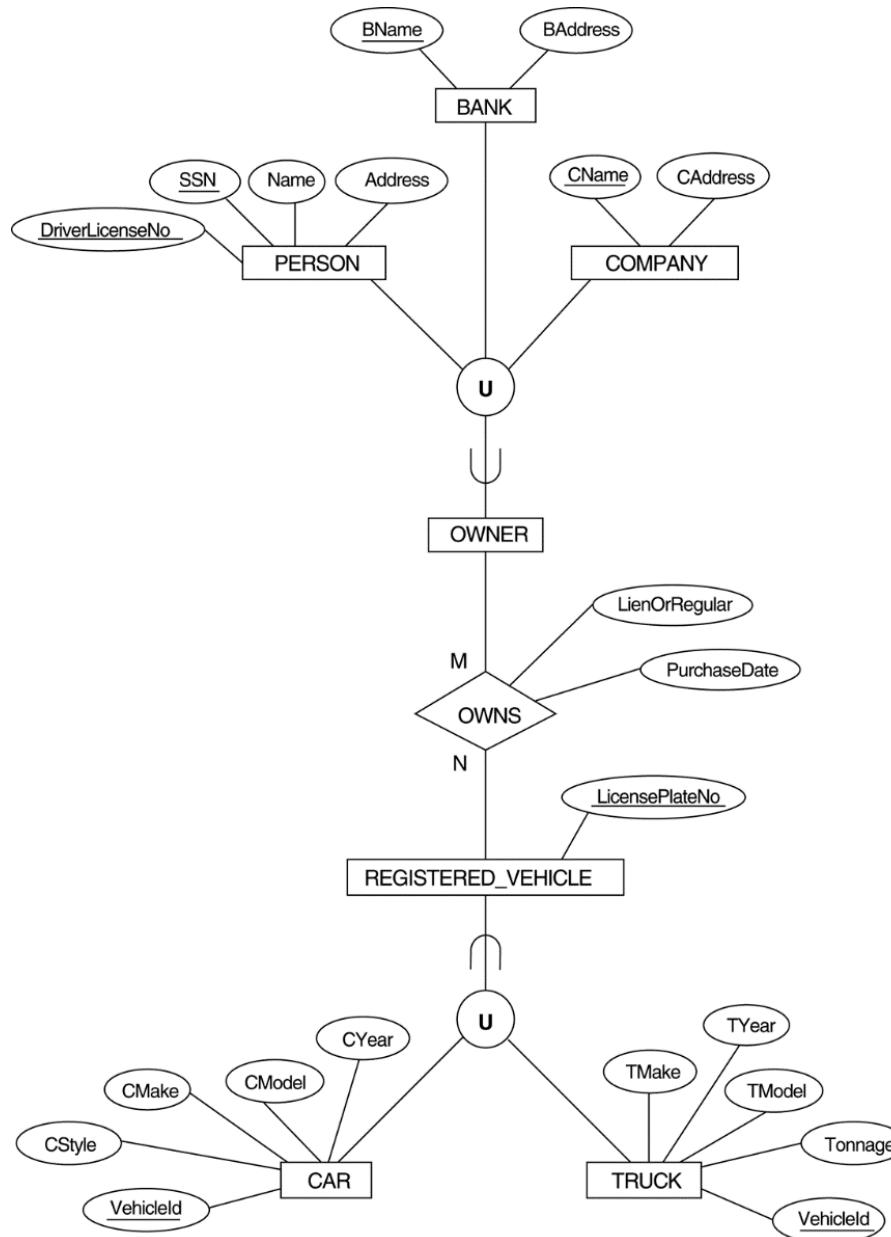


Figure 4.8



# Mapping the EER Categories (Union Types) in Figure 4.8 to Relations

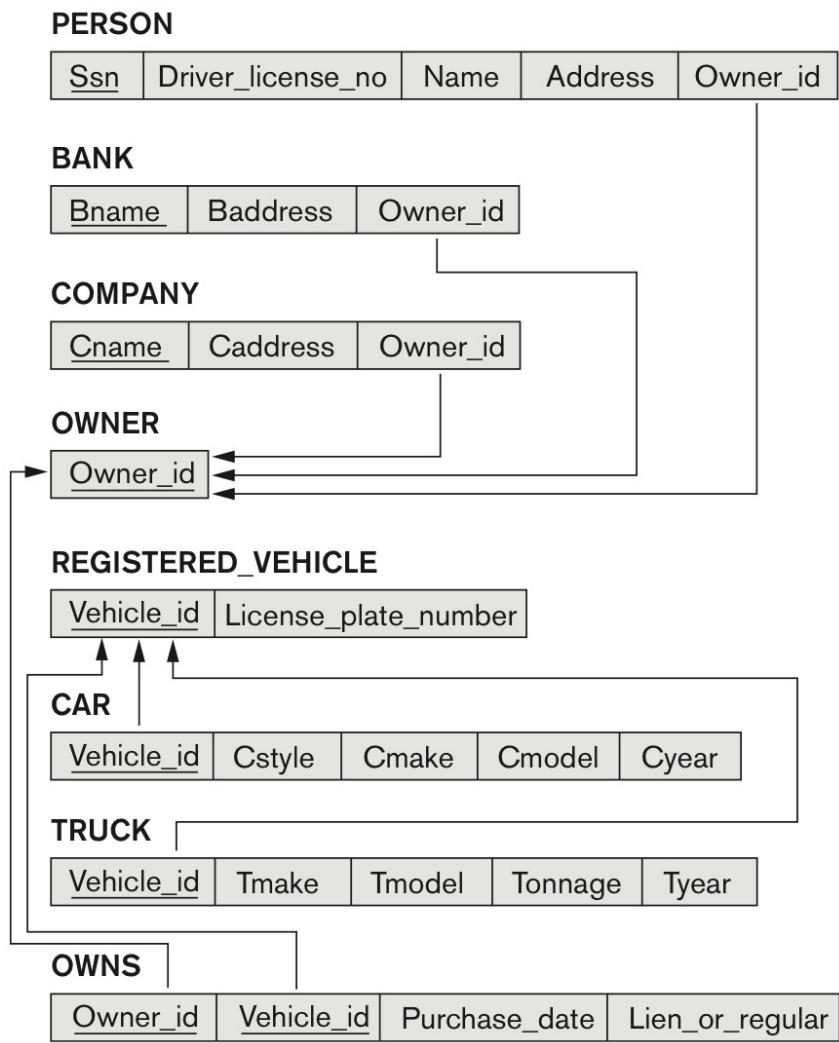
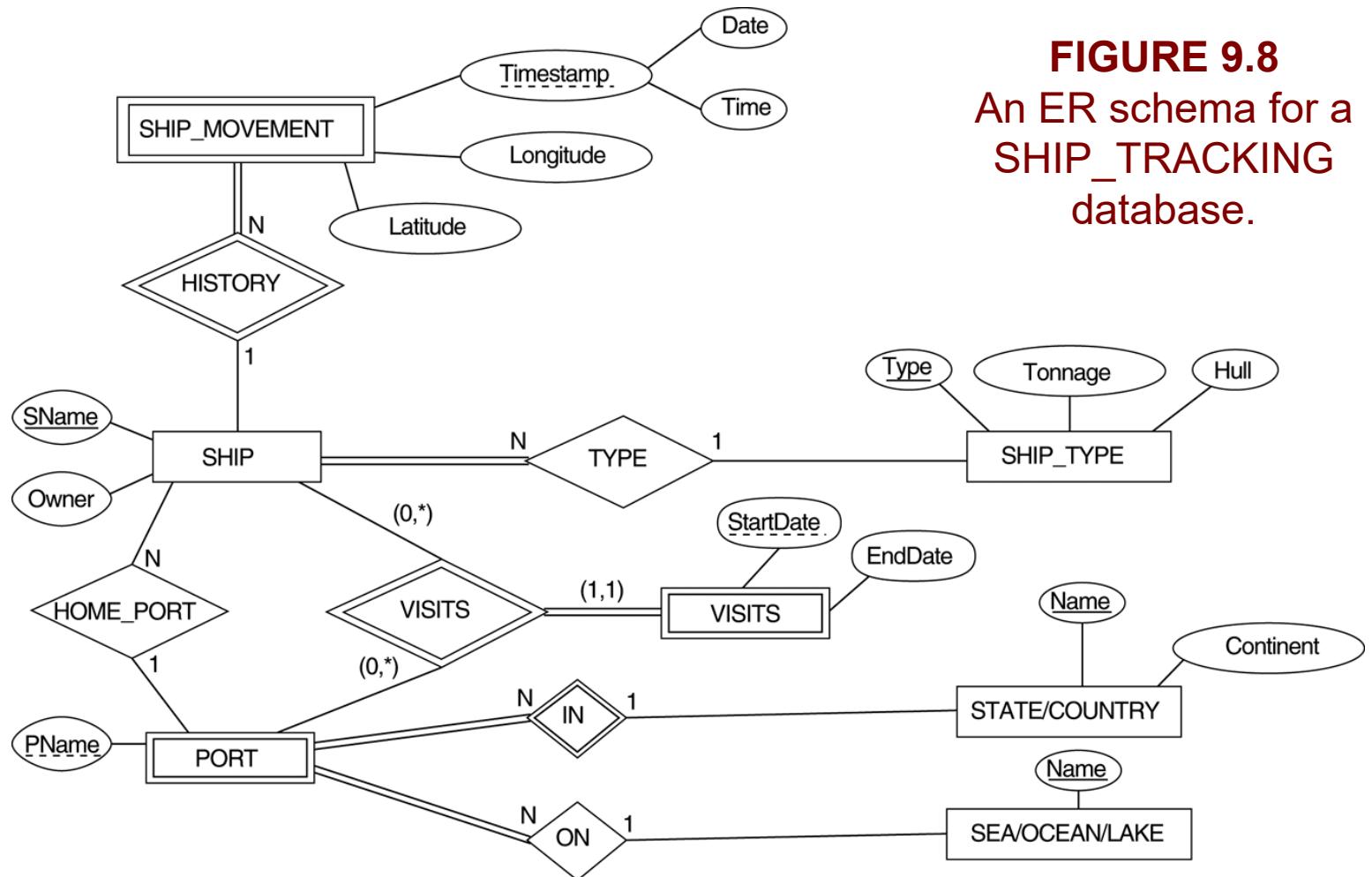


Figure 9.7



# Mapping Exercise-1

Exercise 9.4 : Map this schema into a set of relations.

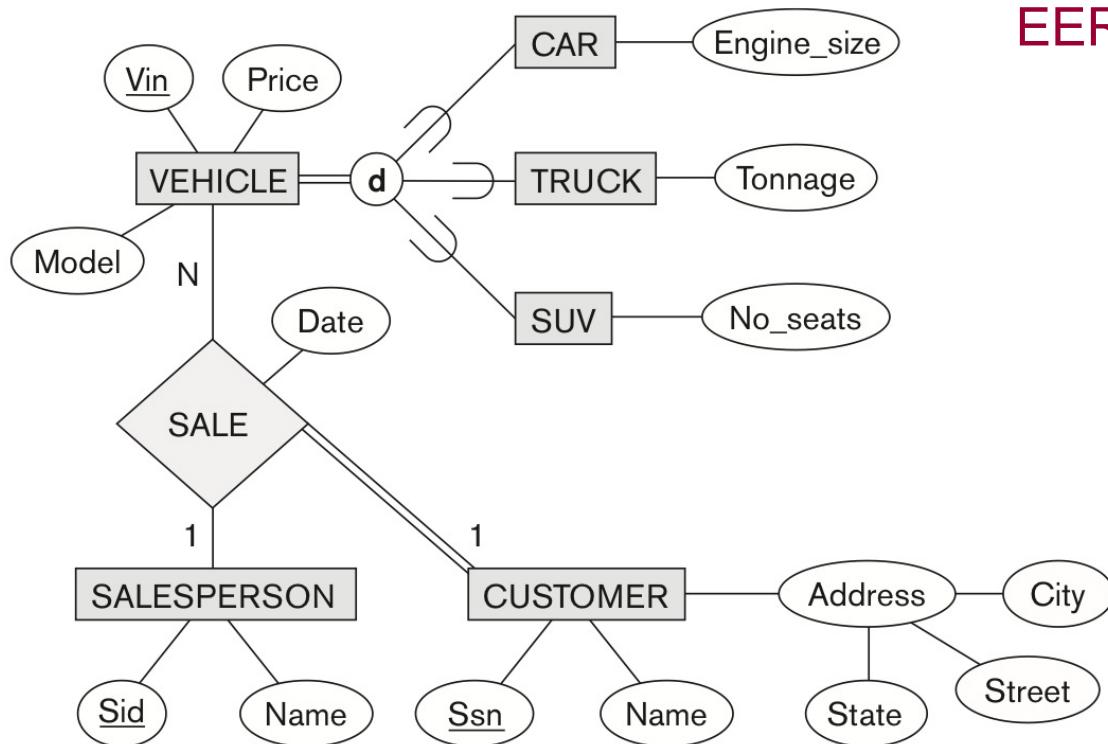


**FIGURE 9.8**  
An ER schema for a  
**SHIP\_TRACKING**  
database.



# Mapping Exercise-2

Exercise 9.9 : Map this schema into a set of relations



**FIGURE 9.9**  
EER diagram for a  
car dealer

## Summary (1/2)

- Map conceptual schema design in the ER model to a relational database schema
  - Algorithm for ER-to-relational mapping
  - Illustrated by examples from the COMPANY database
- Include additional steps in the algorithm for mapping constructs from EER model into relational model

## Summary (2/2) – Mapping Steps

- **ER-to-Relational Mapping Algorithm**

- » Step 1: Mapping of Regular Entity Types
- » Step 2: Mapping of Weak Entity Types
- » Step 3: Mapping of Binary 1:1 Relation Types
- » Step 4: Mapping of Binary 1:N Relationship Types.
- » Step 5: Mapping of Binary M:N Relationship Types.
- » Step 6: Mapping of Multivalued attributes.
- » Step 7: Mapping of N-ary Relationship Types.

- **Mapping EER Model Constructs to Relations**

- » Step 8: Options for Mapping Specialization or Generalization.
- » Step 9: Mapping of Union Types (Categories).

# Agenda

- 
- 1 Session Overview
  - 2 ER and EER to Relational Mapping
  - 3 Database Design Methodology and UML
  - 4 Mapping Relational Design to ER/EER Case Study
  - 5 Comparing Notations and Other Topics
  - 6 Summary and Conclusion



- The Role of Information Systems in Organizations
- The Database Design and Implementation Process
- Use of UML Diagrams as an Aid to Database Design Specification
- Rational Rose: A UML-Based Design Tool
- Automated Database Design Tools



- Design methodology
  - Target database managed by some type of database management system
- Various design methodologies
- **Large database**
  - Several dozen gigabytes of data and a schema with more than 30 or 40 distinct entity types



# The Role of Information Systems in Organizations (1/3)

- Organizational context for using database systems
  - Organizations have created the position of database administrator (DBA) and database administration departments
  - Information technology (IT) and information resource management (IRM) departments
    - Key to successful business management



## The Role of Information Systems in Organizations (2/3)

- Database systems are integral components in computer-based information systems
- Personal computers and database system-like software products
  - Utilized by users who previously belonged to the category of casual and occasional database users
- **Personal databases** gaining popularity
- Databases are distributed over multiple computer systems
  - Better local control and faster local processing



- **Data dictionary systems or information repositories**
  - Mini DBMSs
  - Manage **meta-data**
- High-performance transaction processing systems require around-the-clock nonstop operation
  - Performance is critical



# The Information System Life Cycle (1/4)

- **Information system (IS)**
  - Resources involved in collection, management, use, and dissemination of information resources of organization



# The Information System Life Cycle (2/4)

- **Macro life cycle**
  - Feasibility analysis
  - Requirements collection and analysis
  - Design
  - Implementation
  - Validation and acceptance testing
  - Requirements collection and analysis



## The Information System Life Cycle (3/4)

- The database application system life cycle: **micro life cycle**
  - System definition
  - Database design
  - Database implementation
  - Loading or data conversion



## The Information System Life Cycle (4/4)

- Application conversion
- Testing and validation
- Operation
- Monitoring and maintenance



# The Database Design and Implementation Process (1/4)

- Design logical and physical structure of one or more databases
  - Accommodate the information needs of the users in an organization for a defined set of applications
- Goals of database design
  - Very hard to accomplish and measure
- Often begins with informal and incomplete requirements



## The Database Design and Implementation Process (2/4)

- Main phases of the overall database design and implementation process:
  - 1. Requirements collection and analysis
  - 2. Conceptual database design
  - 3. Choice of a DBMS
  - 4. Data model mapping (also called logical database design)
  - 5. Physical database design
  - 6. Database system implementation and tuning



# Phases of Database Design and Implementation for Large Databases

**Figure 10.1**

Phases of database design and implementation for large databases.

**Phase 1:** Requirements collection and analysis

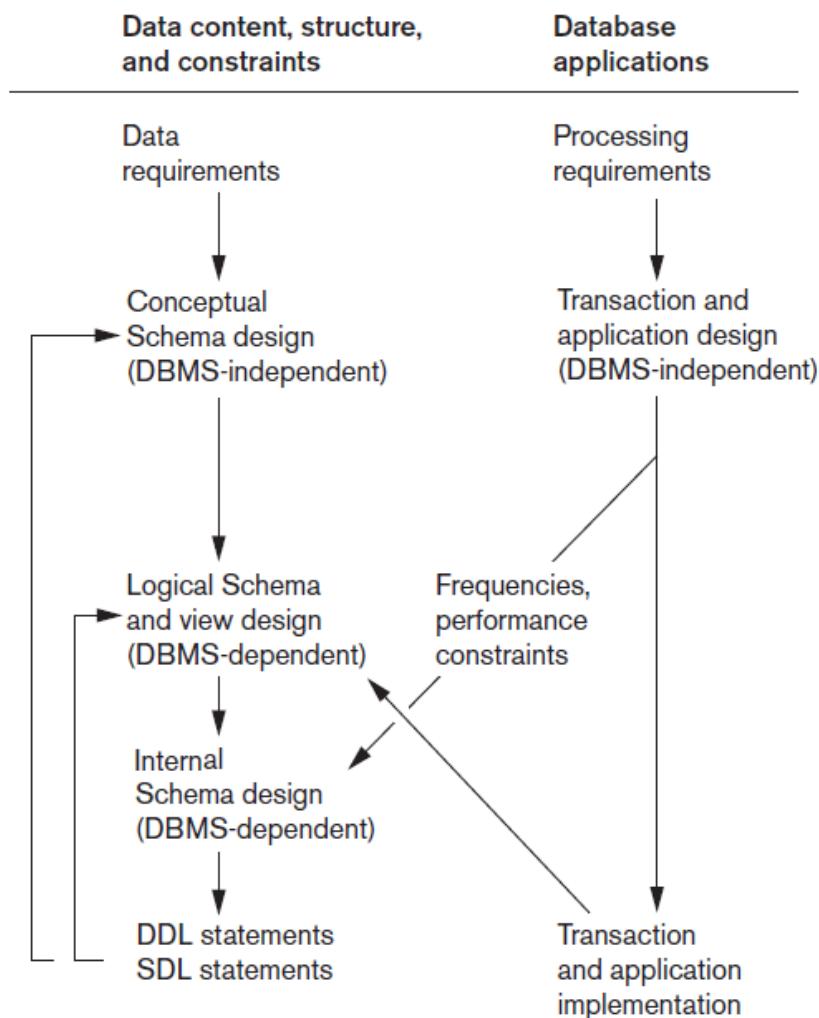
**Phase 2:** Conceptual database design

**Phase 3:** Choice of DBMS

**Phase 4:** Data model mapping (logical design)

**Phase 5:** Physical design

**Phase 6:** System implementation and tuning





# The Database Design and Implementation Process (3/4)

- Parallel activities
  - **Data content, structure, and constraints** of the database
  - Design of database applications
- **Data-driven versus process-driven design**
- **Feedback loops** among phases and within phases are common



## The Database Design and Implementation Process (4/4)

- Heart of the database design process
  - **Conceptual database design (Phase 2)**
  - **Data model mapping (Phase 4)**
  - **Physical database design (Phase 5)**
  - **Database system implementation and tuning (Phase 6)**



# Phase 1: Requirements Collection and Analysis (1/2)

- Activities
  - Identify application areas and user groups
  - Study and analyze documentation
  - Study current operating environment
  - Collect written responses from users



## Phase 1: Requirements Collection and Analysis (2/2)

- **Requirements specification techniques**
  - Oriented analysis (OOA)
  - Data flow diagrams (DFDs)
  - Refinement of application goals
  - Computer-aided



## Phase 2: Conceptual Database Design (1/3)

- Phase 2a: Conceptual Schema Design
  - Important to use a conceptual high-level data model
  - Approaches to conceptual schema design
    - **Centralized (or one shot) schema design approach**
    - **View integration approach**



## Phase 2: Conceptual Database Design (2/3)

- Strategies for schema design
  - **Top-down strategy**
  - **Bottom-up strategy**
  - **Inside-out strategy**
  - **Mixed strategy**
- Schema (view) integration
  - Identify correspondences/conflicts among schemas:
    - **Naming conflicts, type conflicts, domain (value set) conflicts, conflicts among constraints**
  - Modify views to conform to one another
  - Merge of views and restructure



## Phase 2: Conceptual Database Design (3/3)

- Strategies for the view integration process
  - **Binary ladder integration**
  - **N-ary integration**
  - **Binary balanced strategy**
  - **Mixed strategy**
- Phase 2b: Transaction Design
  - In parallel with Phase 2a
  - Specify transactions at a conceptual level
  - Identify **input/output** and **functional behavior**
  - Notation for specifying processes



## Phase 3: Choice of a DBMS

- Costs to consider
  - Software acquisition cost
  - Maintenance cost
  - Hardware acquisition cost
  - Database creation and conversion cost
  - Personnel cost
  - Training cost
  - Operating cost
- Consider DBMS portability among different types of hardware



## Phase 4: Data Model Mapping (Logical Database Design)

- Create a conceptual schema and external schemas
  - In data model of selected DBMS
- Stages
  - System-independent mapping
  - Tailoring schemas to a specific DBMS



## Phase 5: Physical Database Design

- Choose specific file storage structures and access paths for the database files
  - Achieve good performance
- Criteria used to guide choice of physical database design options:
  - Response time
  - Space utilization
  - Transaction throughput



## Phase 6: Database System Implementation and Tuning

- Typically responsibility of the DBA
  - Compose DDL
  - Load database
  - Convert data from earlier systems
- Database programs implemented by application programmers
- Most systems include monitoring utility to collect performance statistics



# Use of UML Diagrams as an Aid to Database Design Specification

- Use UML as a design specification standard
- Unified Modeling Language (UML) approach
  - Combines commonly accepted concepts from many object-oriented (O-O) methods and methodologies
  - Includes **use case diagrams, sequence diagrams, and statechart diagrams**



# UML for Database Application Design

- Advantages of UML
  - Resulting models can be used to design relational, object-oriented, or object-relational databases
  - Brings traditional database modelers, analysts, and designers together with software application developers



# Different Types of Diagrams in UML (1/4)

- Structural diagrams
  - Class diagrams and package diagrams
  - Object diagrams
  - Component diagrams
  - Deployment diagrams

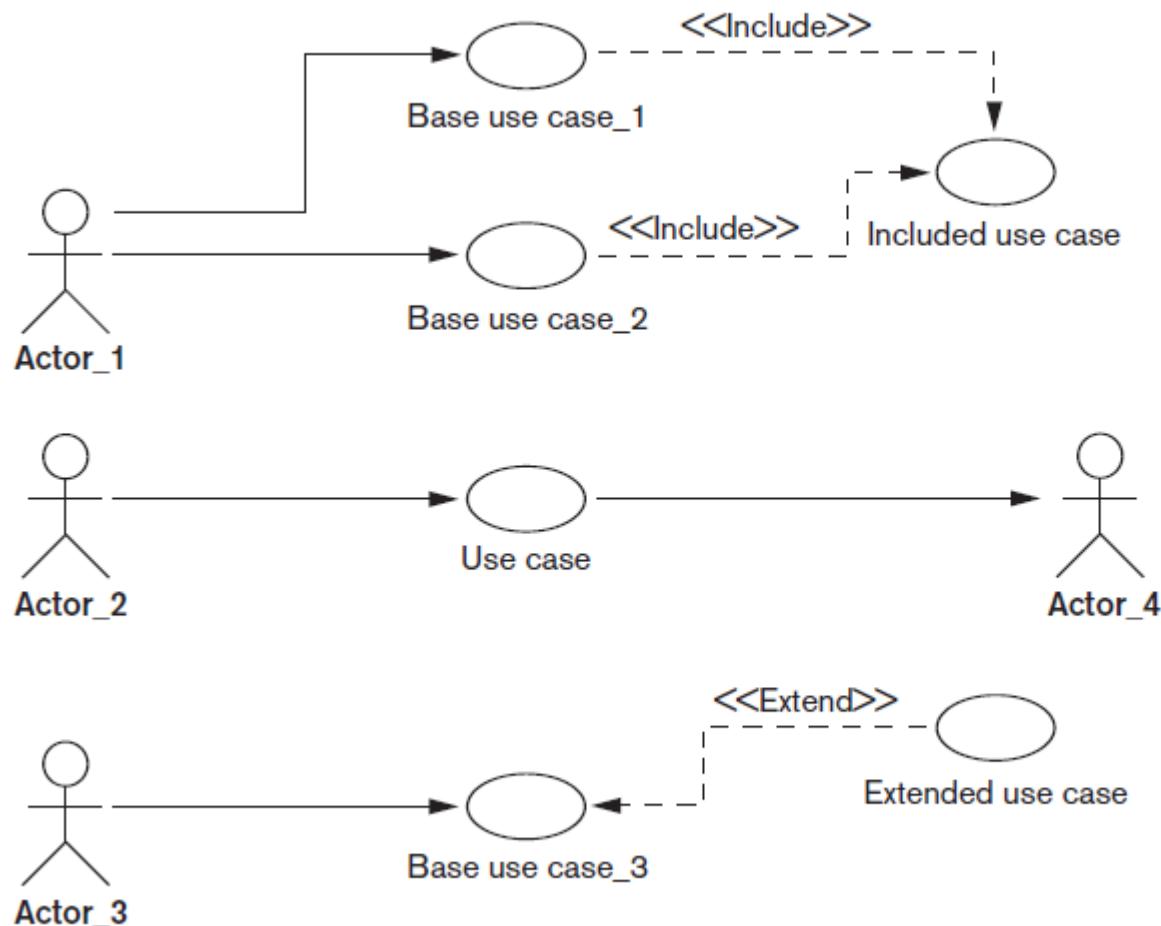


## Different Types of Diagrams in UML (2/4)

- Behavioral diagrams
  - Use case diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Activity diagrams



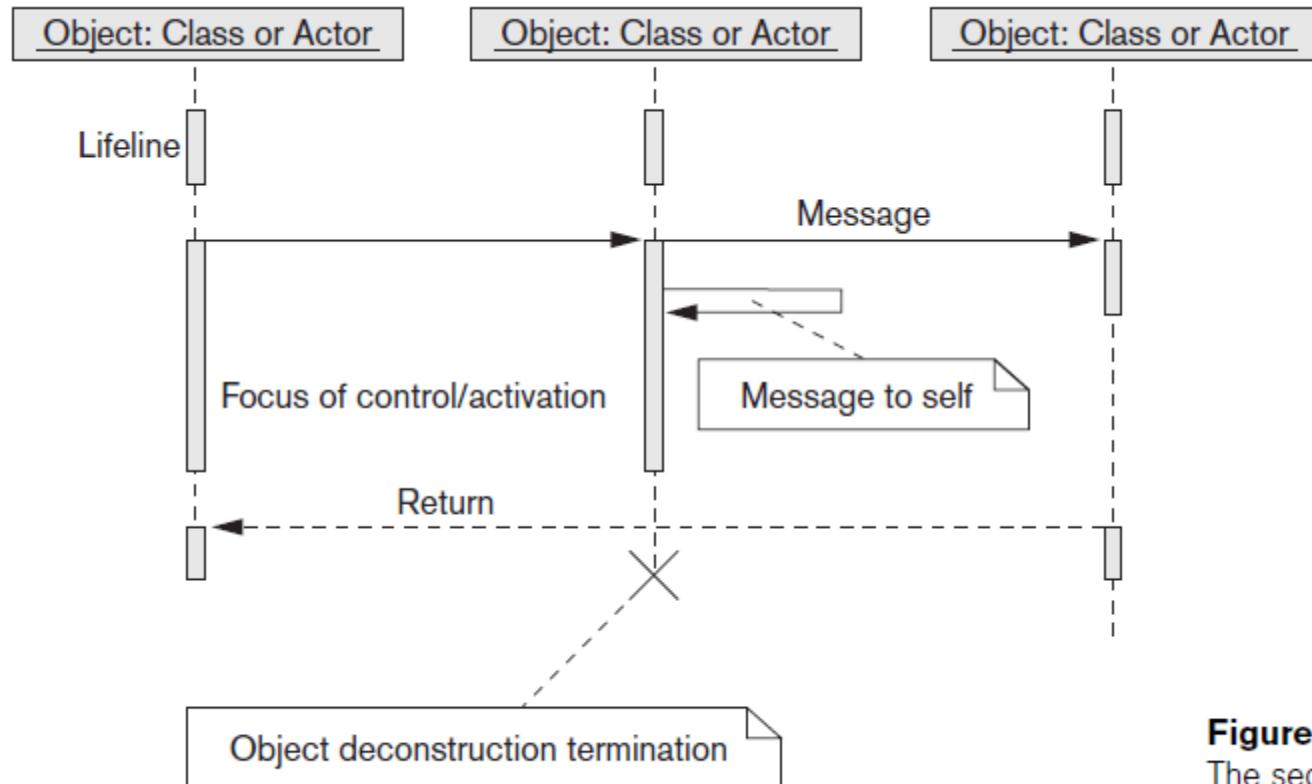
# Use Case Diagram Notation



**Figure 10.7**  
The use case diagram notation.



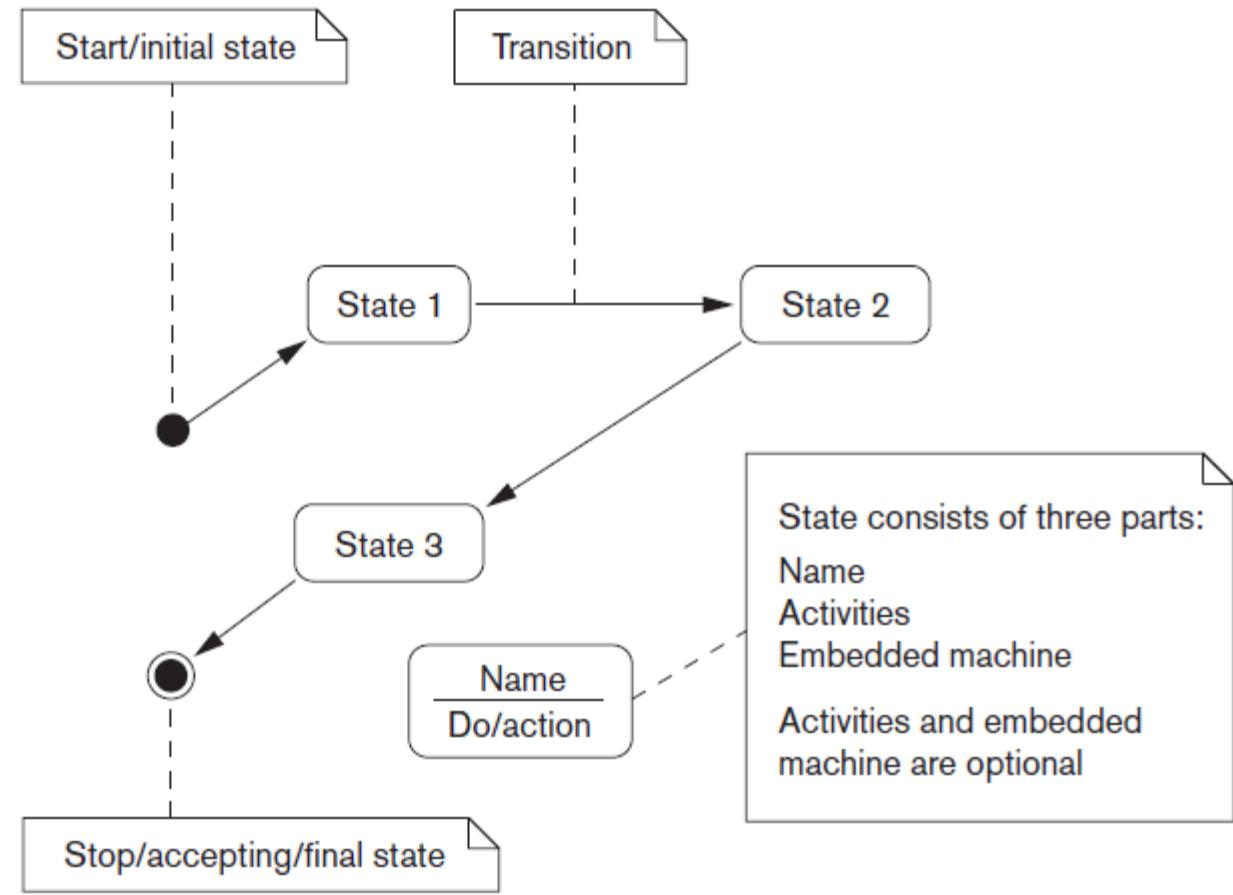
# Different Types of Diagrams in UML (3/4)



**Figure 10.9**  
The sequence diagram notation.

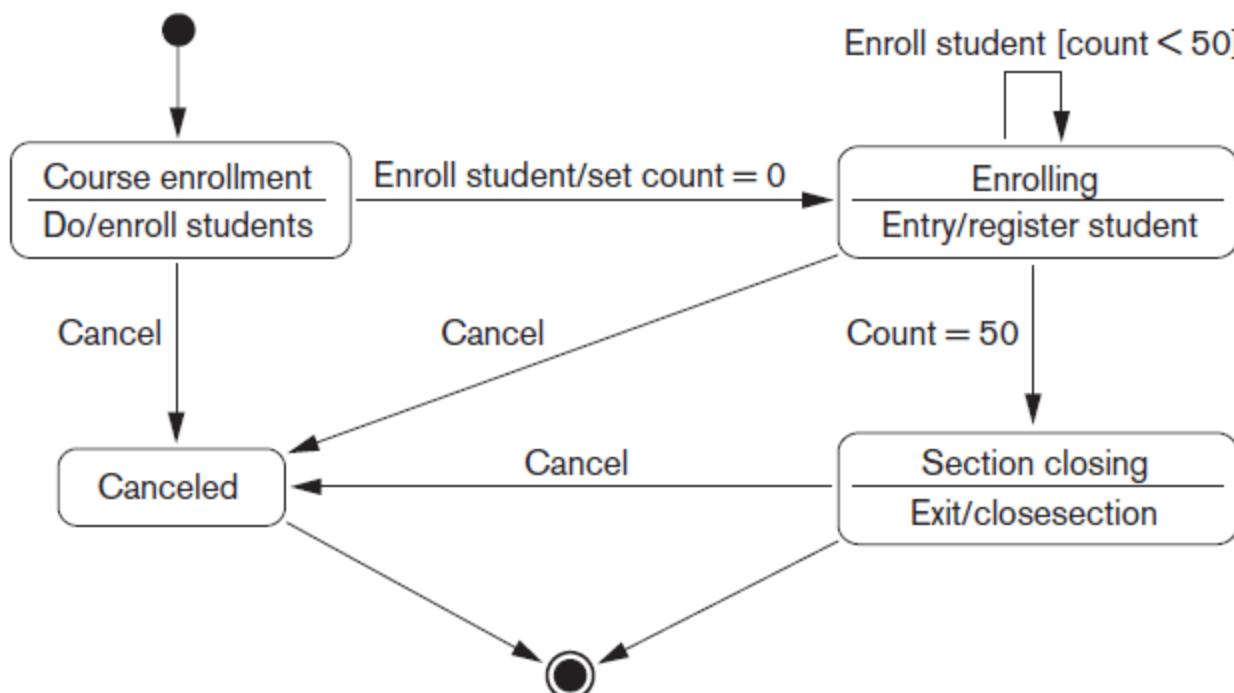


# Different Types of Diagrams in UML (4/4)





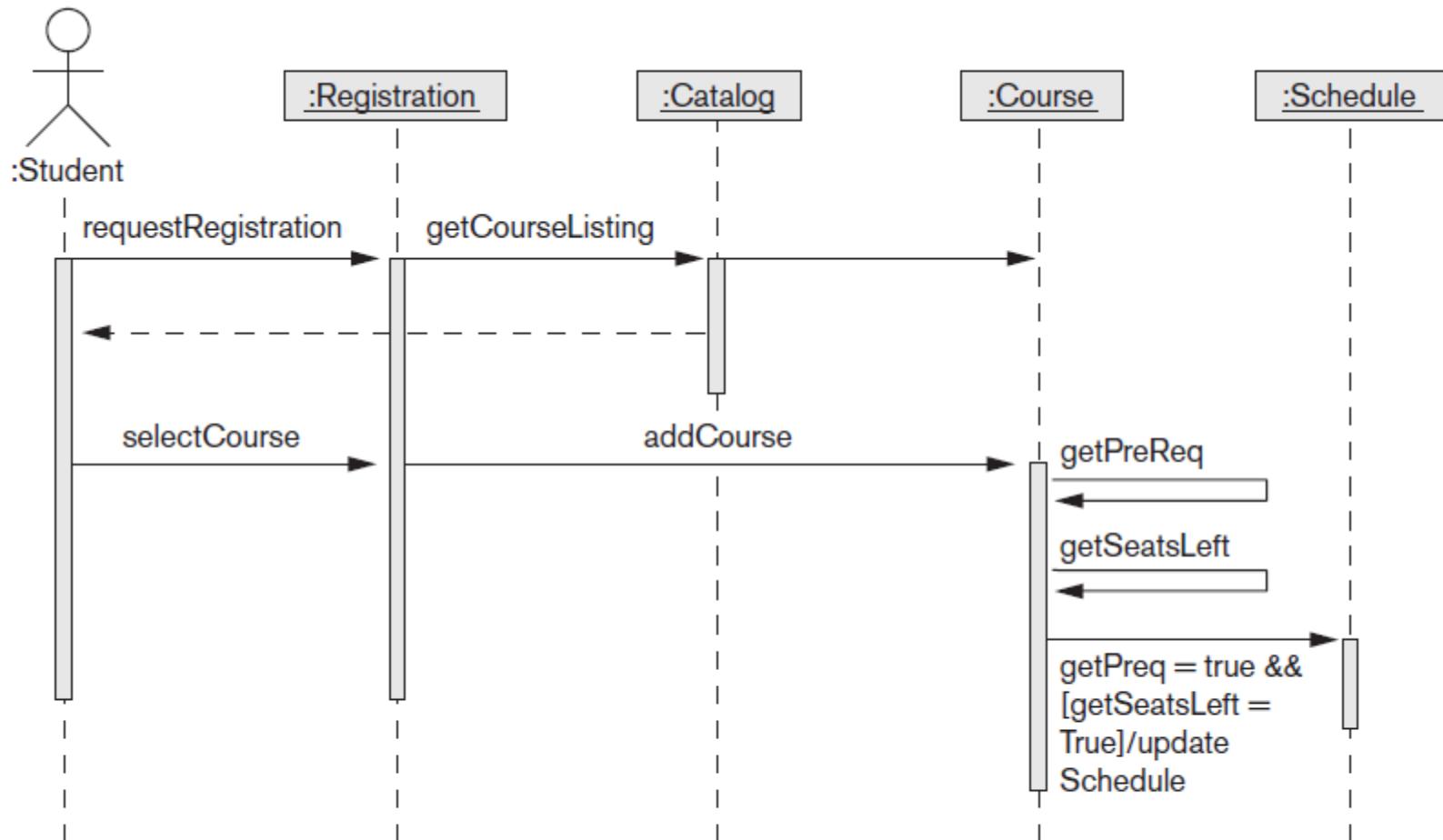
# Modeling and Design Example: UNIVERSITY Database



**Figure 10.11**  
A sample statechart  
diagram for the  
UNIVERSITY  
database.



# Sample Sequence Diagram

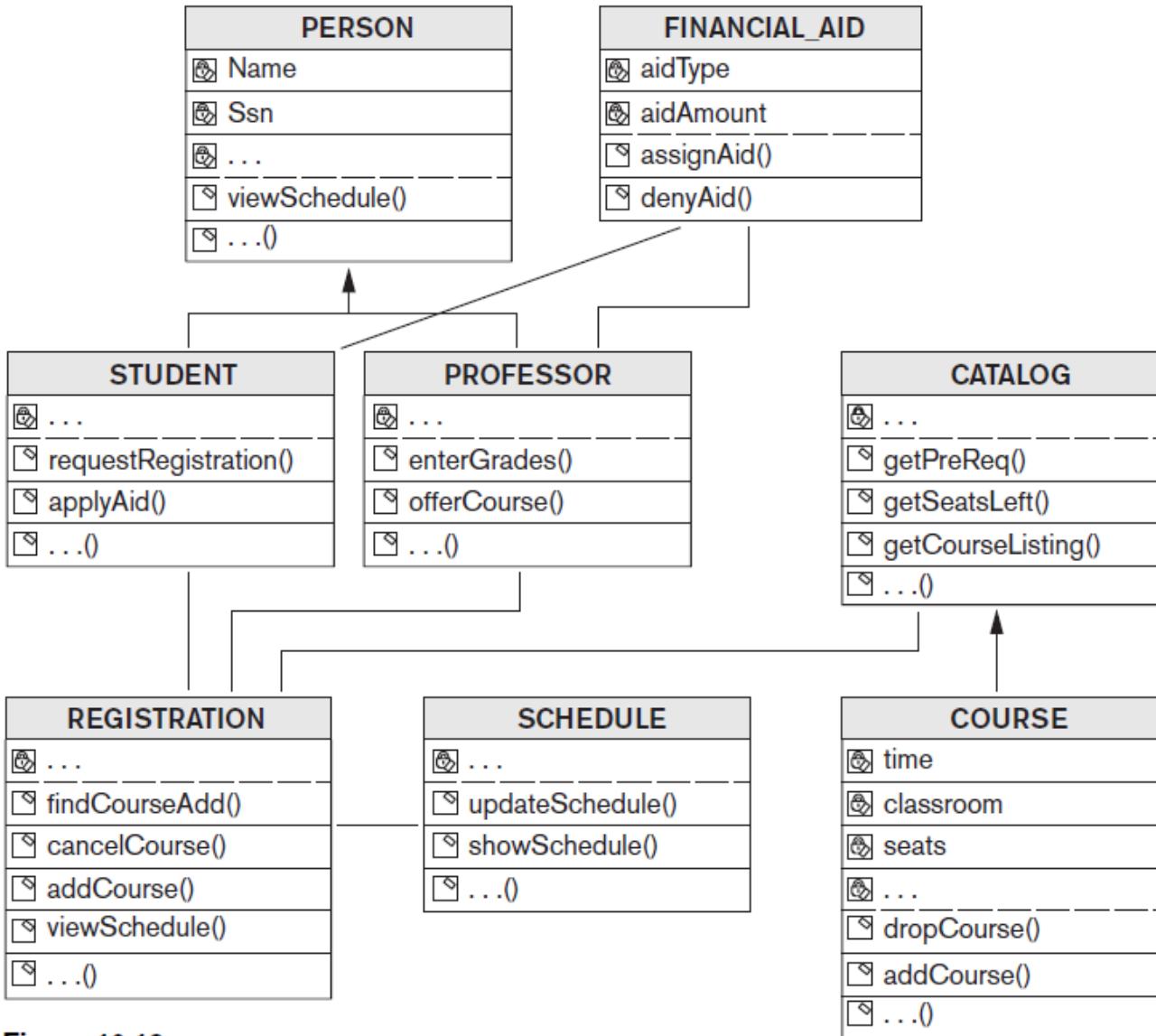


**Figure 10.12**

A sequence diagram for the UNIVERSITY database.



# Sample Class Diagram



**Figure 10.13**

The design of the UNIVERSITY database as a class diagram.



# Rational System Architect: A UML-Based Design Tool

- Rational System Architect for database design
  - Modeling tool used in the industry to develop information systems
- Rational System Architect
  - Visual modeling tool for designing databases
  - Provides capability to:
    - **Forward engineer** a database
    - **Reverse engineer** an existing implemented database into conceptual design
- See comparison with Erwin Data Modeler
  - » [IBM Rational System Architect vs erwin Data Modeler \(DM\) Comparison 2022 | PeerSpot](#)



# Data Modeling Using Rational System Architect (1/4)

- Reverse engineering
  - Allows the user to create a conceptual data model based on an existing database schema specified in a DDL file
- Forward engineering and DDL generation
  - Create a data model directly from scratch in Rational System Architect
  - Generate DDL for a specific DBMS



# Data Modeling Using Rational System Architect (2/4)

- Conceptual design in UML notation
  - Build ER diagrams using class diagrams in Rational System Architect
  - **Identifying relationships**
    - Object in a child class cannot exist without a corresponding parent object
  - **Non-identifying relationships**
    - Specify a regular association (relationship) between two independent classes



## Data Modeling Using Rational System Architect (3/4)

- Converting logical data model to object model and vice versa
  - Logical data model can be converted to an object model
  - Allows a deep understanding of relationships between conceptual and implementation models



## Data Modeling Using Rational System Architect (4/4)

- Synchronization between the conceptual design and the actual database
- Extensive domain support
  - Create a standard set of user-defined data types
- Easy communication among design teams
  - Application developer can access both the object and data models



## Automated Database Design Tools (1/3)

- Many CASE (computer-aided software engineering) tools for database design
- Combination of the following facilities
  - Diagramming
  - Model mapping
  - Design normalization



## Automated Database Design Tools (2/3)

- Characteristics that a good design tool should possess:
  - Easy-to-use interface
  - Analytical components
  - Heuristic components
  - Trade-off analysis
  - Display of design results
  - Design verification



# Automated Database Design Tools (3/3)

[https://en.wikipedia.org/wiki/Comparison\\_of\\_data\\_modeling\\_tools](https://en.wikipedia.org/wiki/Comparison_of_data_modeling_tools)

COMPANY	TOOL	FUNCTIONALITY
Embarcadero Technologies	ER Studio	Database Modeling in ER and IDEF1X
	DB Artisan	Database administration, space and security management
Quest	ERWin	Database Modeling in EER and logical/physical
Oracle	Developer / Designer	Database modeling, application development
Unicom	System Architect	Data modeling, object modeling, process modeling, structured analysis/design
Platinum (defunct)	Enterprise Modeling Suite: Erwin, BPWin, Paradigm Plus	Data, process, and business component modeling
Quest	Toad Data Modeler	Database Modeling in EER and logical/physical
Rational (IBM)	Rational Software Modeling	UML Modeling & application generation in C++/JAVA
Xcase.com	Xcase	Conceptual modeling up to code maintenance
SAP	Enterprise Integration Suite	Data modeling, business logic modeling
Visio	Visio Enterprise	Data modeling, design/reengineering Visual Basic/C++

- Six phases of the design process
  - Commonly include conceptual design, logical design (data model mapping), physical design
- UML diagrams
  - Aid specification of database models and design
- Rational System Architect
  - Provide support for the conceptual design and logical design phases of database design

# Agenda

1 Session Overview

2 ER and EER to Relational Mapping

3 Database Design Methodology and UML

4 Mapping Relational Design to ER/EER Case Study

5 Comparing Notations and Other Topics

6 Summary and Conclusion



# A Case Study

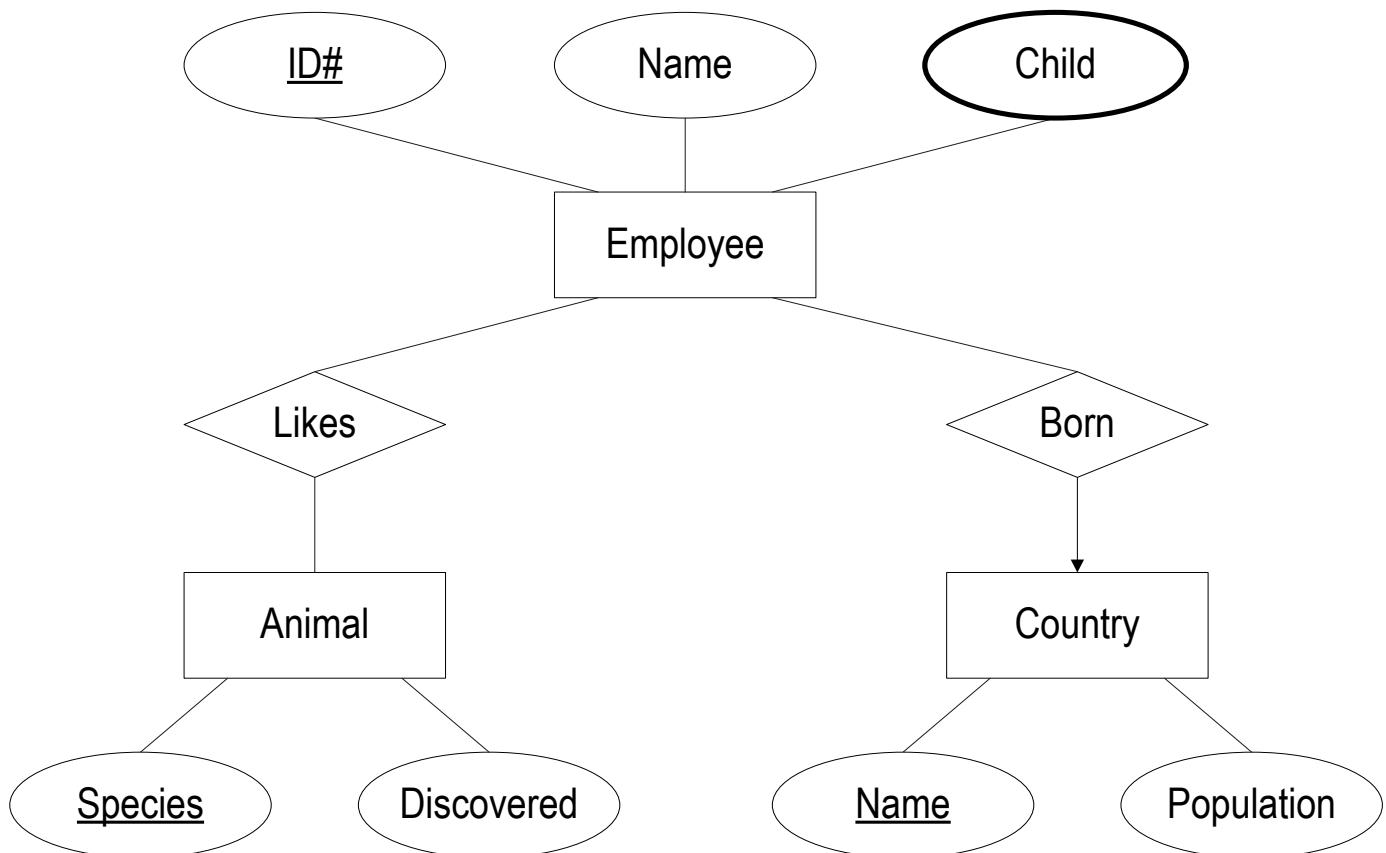
- Implementing an ER diagram as a relational schema (relational database)
- General implementation of strong entities
- Handling attributes of different types
- General implementation of relationships
- Possible special implementation of binary many-to-one relationships
- Implementation of ISA
- Implementation of weak entities Foreign keys
- Primary key / foreign key constraints inducing many-to-one relationships between tables
- Concept of referential integrity
- Crow's feet notation: ends of lines
- Crow's feet notation: pattern of lines

# From ER Diagrams To Relational Database

- We are now ready to convert ER diagrams into relational databases
- ***Generally, but not always***
  - » *An entity set is converted into a table*
  - » *A relationship is converted into a table*
- As stated earlier, this is ***logical design***, though some call it ***physical design*** (as we also saw earlier)
- We will first go ***very carefully*** over a very simple example
- Then, we will be ready to go quickly through our large example
- Then, we look at some additional aspects of interest
- We will use SQL Power Architect for specifying relational databases
- We will briefly look at MySQL Workbench



# Small ER Diagram



# Small Example

- 5 Employees:
  - » 1 is Alice has Erica and Frank, born in US, likes Horse and Cat
  - » 2 is Bob has Bob and Frank, born in US, likes Cat
  - » 4 is Carol
  - » 5 is David, born in IN
  - » 6 is Bob, born in CN, likes Yak
- 4 Countries
  - » US
  - » IN has 1347
  - » CN has 1412
  - » RU
- 5 Animals
  - » Horse in Asia
  - » Wolf in Asia
  - » Cat in Africa
  - » Yak in Asia
  - » Zebra in Africa

## More About the Example

- The given ER diagram is clear, other than
  - » Discovered, which is the continent in which a particular species was first discovered
- Each child is a “dependent” of only one employee in our database
  - » If both parents are employees, the child is “assigned” to one of them (say the one whose ID# is smaller)
- We are given additional information about the application
  - » **Values of attributes in a primary key must not be missing** (this is a general rule as we know, not only for this example so I should not say that, and consider this a pedagogical reminder, not actually stated)
  - » Other than attributes in a primary key, other attributes, unless stated otherwise, may be missing
  - » The value of Name is known (not missing) for every Employee
- To build up our intuition, let’s look at some specific instance of our application

# Country

- There are four countries, listing for them:  
Cname, Population (the latter only when known):

- » US
- » IN, 1347
- » CN, 1412
- » RU

Country	Cname	Population
US		
IN		1347
CN		1412
RU		

- We create a table for Country “in the most obvious way,” by ***creating a column for each attribute*** (underlining the attributes of the primary key) and that works:
- Note that some “slots” contains NULLs, indicated by emptiness

- There are five animals, listing for them: Species, Discovered (note, that even though not required, Discovered happens to be known for every Species):

- » Horse, Asia
- » Wolf, Asia
- » Cat, Africa
- » Yak, Asia
- » Zebra, Africa

Animal	<u>Species</u>	Discovered
Horse		Asia
Wolf		Asia
Cat		Africa
Yak		Asia
Zebra		Africa

- We create a table for Animal as before, and that works:

# Employee

- There are five employees, listing for them: ID#, Name, (name of) Child (note there may be any number of Child values for an Employee, zero or more):
  - » 1, Alice, Erica, Frank
  - » 2, Bob, Bob, Frank
  - » 4, Carol
  - » 5, David
  - » 6, Bob, Frank
- We create a table for Employee in the most obvious way, and that **does not** work:

Employee	<u>ID#</u>	Name	Child	Child
	1	Alice	Erica	Frank
	2	Bob	Bob	Frank
	4	Carol		
	5	David		
	6	Bob	Frank	

# Employee

- Child is a multivalued attribute so, the number of columns labeled “Child” is, in principle, unbounded
- A table **must have** a fixed number of columns
  - » It must be an instance in/of a relational schema
- If we are ready to store up to 25 children for an employee and create a table with 25 columns for children, perhaps tomorrow we get an employee with 26 children, who will not “fit”; Moulay Ismail ibn Sharif had more than 1,000 children
- We **replace our attempted single table** for Employee **by two tables**
  - » One for all the attributes of Employee other than the multivalued one (Child)
  - » One for pairs of the form (primary key of Employee, Child)
- Note that both tables have a fixed number of columns, no matter how many children an employee has, so we have correct structures

# Employee and Child

## ■ Replace (incorrect)

Employee	ID#	Name	Child	Child
	1	Alice	Erica	Frank
	2	Bob	Bob	Frank
	4	Carol		
	5	David		
	6	Bob	Frank	

Employee	ID#	Name	Child	ID#	Child
	1	Alice		1	Erica
	2	Bob		1	Frank
	4	Carol		2	Bob
	5	David		2	Frank
	6	Bob		6	Frank

# Employee and Child With Better Column Names

## (This is not Required but Seems to be Good)

- Replace (incorrect)

Employee	ID#	Name	Child	Child
	1	Alice	Erica	Frank
	2	Bob	Bob	Frank
	4	Carol		
	5	David		
	6	Bob	Frank	

Employee	ID#	Name	Child	Parent	Child
	1	Alice		1	Erica
	2	Bob		1	Frank
	4	Carol		2	Bob
	5	David		2	Frank
	6	Bob		6	Frank

# Employee and Child

- The primary key of the table Employee is ID#
- The primary key of the table Child is the pair: Parent,Child
- One attribute is not sufficient to get a primary key for Child
- By “correlating” ID# with Parent, we can answer queries such as:  
What is the Name of the Child’s Parent
  
- It is clear from the example how to handle any number of simple multivalued attributes an entity has, basically
  - » Create a “main” table with all the attributes other than multivalued ones  
Its primary key is the original primary key of the entity set
  - » Create a table for each multivalued attribute consisting of the primary key for the main table and of that multivalued attribute  
Its primary key is the primary key of the entity combined with the original multivalued attribute

# Foreign Key

- Let us return to our example
- Note that any value of ID# that appears in Child must also appear in Employee
  - Because a child must be a dependent of an existing employee
- This is an instance of a **foreign key**
- ID# in Child is a **foreign key referencing** Employee
  - This means that ID# appearing in Child must appear in some row “under” the columns (here only one) of primary key in Employee (slightly oversimplifying in this Unit: not necessarily primary key)
  - Note that ID# is not a key of Child (but is a part of a key), **so a foreign key in a table does not have to be a key of that table**

Employee	ID#	Name	Child	ID#	Child
	1	Alice		1	Erica
	2	Bob		1	Frank
	4	Carol		2	Bob
	5	David		2	Frank
	6	Bob		6	Frank

# Foreign Key ≡ A Binary Many-To-One Relationship Between Tables (Partial Function)

Employee	ID#	Name	Child	ID#	Child
	1	Alice		1	Erica
	2	Bob		1	Frank
	4	Carol		2	Bob
	5	David		2	Frank
	6	Bob		6	Frank

- Note:
  - » Every row of Child has a single value of a primary key of Employee, so every row of Child “maps” to a single row of Employee
  - » Every row of Employee has zero or more rows of Child mapped to it
    - In other words, no constraint
- Here the function is **total**: defined everywhere

# Foreign Key ≡ A Binary Many-To-One Relationship Between Tables

Employee	ID#	Name	Child	Parent	Child
	1	Alice		1	Erica
	2	Bob		1	Frank
	4	Carol		2	Bob
	5	David		2	Frank
	6	Bob		6	Frank

- Another option
- Note column names do not have to be the same for the mapping to take place
- But you need to specify which column is foreign key referring to what
- Here: Parent in Child is a foreign key referencing the column ID# in Employee
- In SQL DDL, the syntax is more explicit: the specific column(s) of Employee are stated: we will see later

- Born needs to specify which employees were born in which countries (for those for whom this information is known)
- We can list what is the current state
  - » Employee identified by 1 was born in country identified by US
  - » Employee identified by 2 was born in country identified by IN
  - » Employee identified by 5 was born in country identified by IN
  - » Employee identified by 6 was born in country identified by CN

# Born

- Born needs to specify who was born where
- We have tables for
  - » Employee
  - » Country
- We know that ***each employee was born in at most one country*** (we either know the country or not)
- We have a ***binary many-to-one relationship*** between Employee and Country, but no implementation yet

Employee	ID#	Name	Country	CName	Population
	1	Alice		US	
	2	Bob		IN	1347
	4	Carol		CN	1412
	5	David		RU	
	6	Bob			

The diagram illustrates a many-to-one relationship between the Employee table and the Country table. Red arrows originate from the Employee table's rows (ID# 1, 2, 4, 5, 6) and point to the Country table's rows (US, IN, CN, RU). Specifically, Employee ID# 1 points to US, Employee ID# 2 points to IN, Employee ID# 4 points to CN, Employee ID# 5 points to RU, and Employee ID# 6 also points to RU.

# Implementation for Born

- Augment Employee so instead of

Employee	<u>ID#</u>	Name
	1	Alice
	2	Bob
	4	Carol
	5	David
	6	Bob

Country	<u>CName</u>	Population
US		
IN	1347	
CN	1412	
RU		

Employee	<u>ID#</u>	Name	<u>CName</u>
	1	Alice	US
	2	Bob	IN
	4	Carol	
	5	David	IN
	6	Bob	CN

Country	<u>CName</u>	Population
US		
IN	1347	
CN	1412	
RU		

# Implementation for Born

- Augment Employee so instead of

Employee	ID#	Name
1	Alice	
2	Bob	
4	Carol	
5	David	
6	Bob	

Country	CName	Population
US		
IN	1347	
CN	1412	
RU		

we have two tables and a binary many-to-one mapping

Employee	ID#	Name	CName	Country	CName	Population
1	Alice		US	US		
2	Bob		IN	IN	1347	
4	Carol			CN	1412	
5	David		IN			
6	Bob		CN	RU		

# Foreign Key Constraint Implementing Born

- We have again a foreign key constraint
  - Any value of CName in Employee must also appear in Country as a primary key in some row
  - CName in Employee is a foreign key referencing Country
- 
- Note that CName in Employee is not even a part of its primary key
  - Note that the binary many-to-one mapping is not total in this example

Employee	ID#	Name	CName	Country	CName	Population
	1	Alice	US		US	
	2	Bob	IN		IN	1347
	4	Carol			CN	1412
	5	David	IN		RU	
	6	Bob	CN			



The diagram illustrates a many-to-one relationship between the Employee table and the Country table. Red arrows point from the Employee table's CName column to the Country table's CName column. Specifically, the arrow from employee ID 1 points to 'US', the arrow from employee ID 2 points to 'IN', and the arrows from employees 4, 5, and 6 all point to 'CN'. This visualizes how multiple rows in the Employee table can share the same value in the CName column, which must be a primary key in the Country table.

# Foreign Key Constraint Implementing Born

- Perhaps better (and frequently done in practice) use a different name for foreign keys
- Any value of CBirth in Employee must also appear in Country as a primary key in some row
- CBirth in Employee is a foreign key referencing Country
- We will not talk here about such, possibly convenient, renaming

Employee	ID#	Name	CBirth	Country	CName	Population
	1	Alice	US	US		
	2	Bob	IN	IN	1347	
	4	Carol		CN	1412	
	5	David	IN			
	6	Bob	CN	RU		

- Likes needs to specify which employees like which animals
- We can list what is the current state:
  - » Employee identified by 1 likes animal identified by Horse
  - » Employee identified by 1 likes animal identified by Cat
  - » Employee identified by 2 likes animal identified by Cat
  - » Employee identified by 6 likes animal identified by Yak

# Likes

- We can describe Likes by drawing lines between the two tables
- We need to “store” this set of red lines
- ***Likes is a many-to-many relationship***
  - » ***It is not a many-to-one relationship and therefore it is not a partial function***

Employee	ID#	Name	Animal	Species	Discovered
	1	Alice		Horse	Asia
	2	Bob		Wolf	Asia
	4	Carol		Cat	Africa
	5	David		Yak	Asia
	6	Bob		Zebra	Africa

The diagram illustrates a many-to-many relationship between employees and animals. Red lines connect specific employee entries to multiple animal entries. Alice is connected to both Horse and Wolf. Bob is connected to both Wolf and Zebra. Carol is connected to Cat. David is connected to Yak. Bob is also connected to Zebra.

# Likes (impossible implementation)

- Cannot store with Employee (there is no limit on the number of animals an employee likes)

Employee	ID#	Name	Animal	Species	Discovered
	1	Alice		Horse	Asia
	2	Bob		Wolf	Asia
	4	Carol		Cat	Africa
	5	David		Yak	Asia
	6	Bob		Zebra	Africa

Employee	ID#	Name	Species	Species
	1	Alice	Horse	Cat
	2	Bob	Cat	
	4	Carol		
	5	David		
	6	Bob	Yak	

# Likes (impossible implementation)

- Cannot store with Animal (there is no limit on the number of employees who like an animal)

Employee	ID#	Name	Animal	Species	Discovered
	1	Alice		Horse	Asia
	2	Bob		Wolf	Asia
	4	Carol		Cat	Africa
	5	David		Yak	Asia
	6	Bob		Zebra	Africa

Animal	Species	Discovered	ID#	ID#
Horse	Asia	1		
Wolf	Asia			
Cat	Africa	1	2	
Yak	Asia	6		
Zebra	Africa			

# Likes

- Each red line is an **edge** defined by its vertices
- We create a table storing the red lines; that is, its vertices
- We can do this using the **primary keys** of the entities  
We do not need other attributes such as Name or Discovered
- The table for Likes contains tuples:
  - » 1 likes Horse
  - » 1 likes Cat
  - » 2 likes Cat
  - » 6 likes Yak

Likes	ID#	Species
1		Horse
1		Cat
2		Cat
6		Yak

# Likes

- Note that ***there are foreign key constraints***
  - » ID# appearing in Likes is a foreign key referencing Employee
  - » Species appearing in Likes is a foreign key referencing Animal
- And two many-to-one mappings are induced
- Note: ***a binary many-to-many relationship was replaced by a new table and two many-to one relationships***

Employee	ID#	Name	Likes	ID#	Species	Animal	Species	Discovered
	1	Alice		1	Horse		Horse	Asia
	2	Bob		1	Cat		Wolf	Asia
	4	Carol		2	Cat		Cat	Africa
	5	David		6	Yak		Yak	Asia
	6	Bob				Zebra		Africa

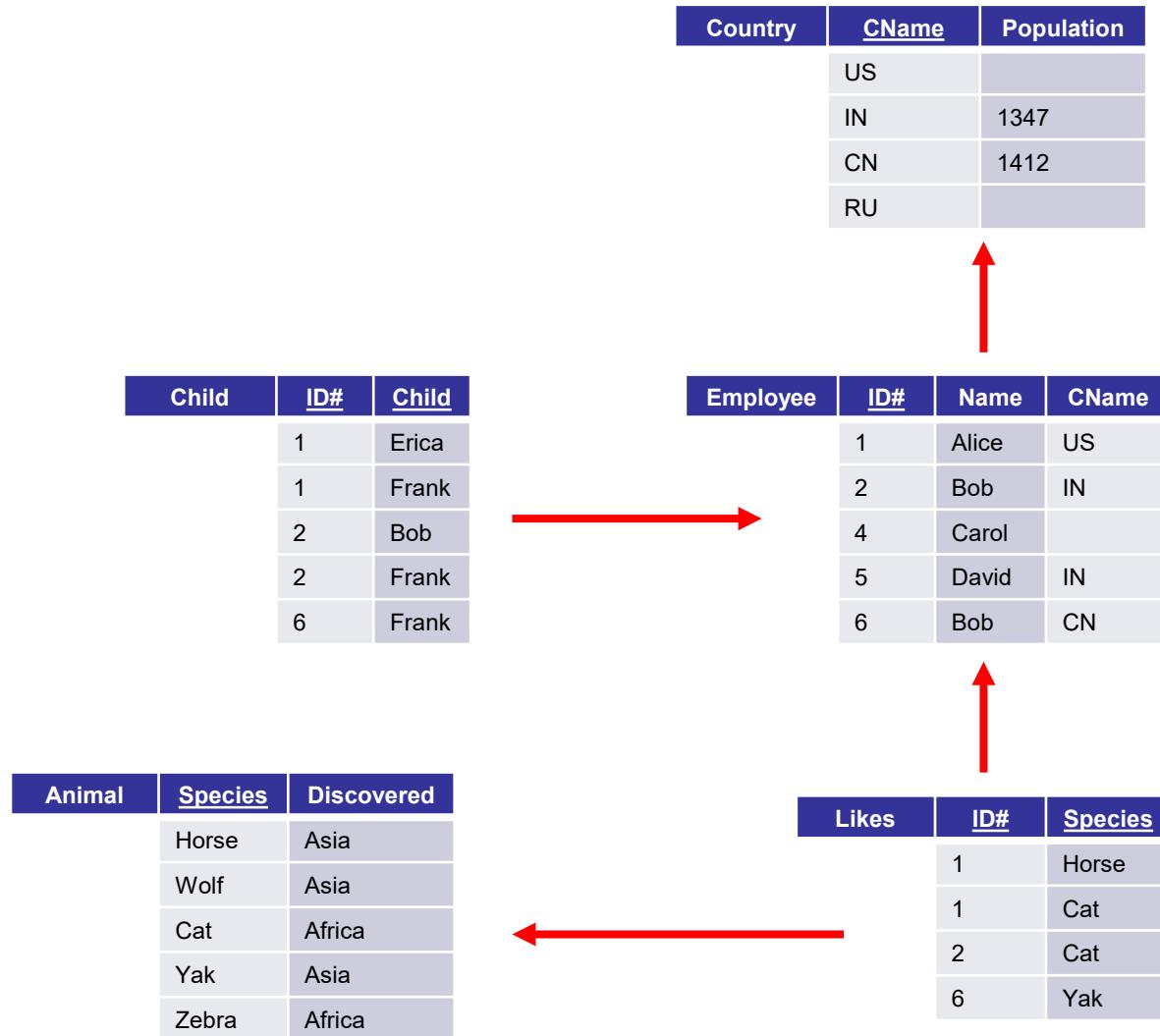
The diagram illustrates the database schema. The Employee table has columns for Employee, ID#, and Name. The Likes table has columns for Likes, ID#, and Species. The Animal table has columns for Animal, Species, and Discovered. Red arrows show the relationships: multiple employees can like the same animal, and multiple species can be liked by the same employee. Specifically, Alice likes Horse, Cat, and Cat; Bob likes Horse, Cat, and Yak; Carol likes Cat; David likes Yak; and Bob likes Horse again. The Animal table shows that Horse is found in Asia, Wolf in Asia, Cat in Africa, Yak in Asia, and Zebra in Africa.

- ***Let us illustrate the process using SQL Power Architect***
  - » It is reasonable and available
  - » It is very lightweight
- We could even use software to generate database specifications from the diagram to SQL DDL (forward engineering)
  - » SQL Power Architect can do that

# Specifying a Relational Implementation Using SQL Power Architect

- A drawing in SQL Power Architect does not deal with Entities and Relationships (though such terms are sometimes used in this context)
- This is good, as ***it produces a relational schema***, which is what we need, but this is a lower-level construct
- ***It focuses on tables and the implicit many-to-one binary relationships induced by foreign keys***
- ***Table***
  - » A rectangle with three vertical subrectangles: name, list of attributes in the primary key, list of attributes not in the primary key
  - » Required attributes characterized as such: NOT NULL
  - » Attributes in the primary key and foreign keys are labeled as such
- ***Relationship***
  - » ***A many-to-one binary*** (or perhaps one-to-one, which is discussed later) relationship induced by a ***foreign key constraint*** is explicitly drawn by means of a line segment with an arrow heads of various types

# Relational Implementation For The Example: Tables And Binary Many-To-One Mappings



# Standard Notation For Cardinality Constraints

## Crow's Feet: Richer Than Standard Arrows

- We will look at a binary many-to-one mapping relationship Likes from Person to Country
- We will ask the following question: If I start from a Person, how many Countries can I reach
- This is the same as: Given a Person, how many Countries can it Like?
- We will ask the following question: If I start from a Country, how many Persons can I reach?
- This is the same as: Given a Country, how many Person can Like it?
- Our new notation will allow us to characterize 6 cases

# Standard Notation For Cardinality Constraints

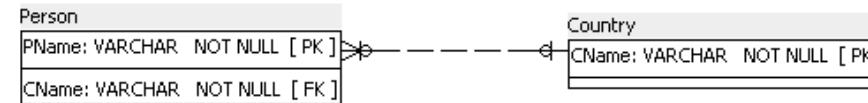
## Crow's Feet: Richer Than Arrows

- The notation will have various “ends of line”
- Our previous notation had 2 types of ends of lines
  - » Arrowhead
  - » Plain, no arrowhead
- We will have ends of lines consisting of some combinations of
  - » Circle, standing for = 0
  - » Vertical bar, standing for = 1
  - » A “foot” with 3 “toes”, standing for > 1
- Next, we look at all the possible cases
- We will have
  - » Person
  - » Country
  - » Binary many-to-one mapping from Person to Country specified, of course, using a foreign key
- The lines will be dashed, not important why for now

# Born With Crow's Feet

## A Case

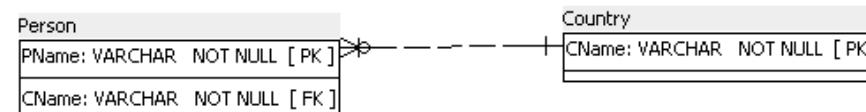
- From Person we can reach 0 or 1 Countries
- From Country we can reach 0, 1, or  $> 1$  Persons



# Born With Crow's Feet

## A Case

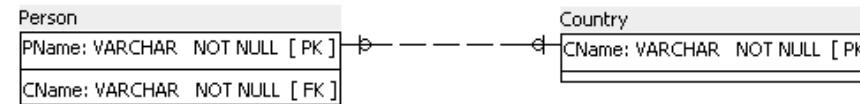
- From Person we can reach 1 Countries
- From Country we can reach 0, 1, or  $> 1$  Persons



# Born With Crow's Feet

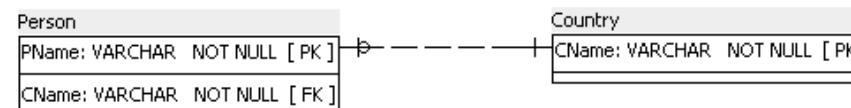
## A Case

- From Person we can reach 0 or 1 Countries
- From Country we can reach 0 or 1 Persons



# Born With Crow's Feet A Case

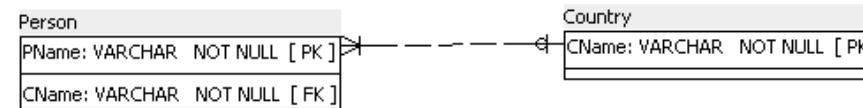
- From Person we can reach 1 Countries
- From Country we can reach 0 or 1 Persons



# Born With Crow's Feet

## A Case

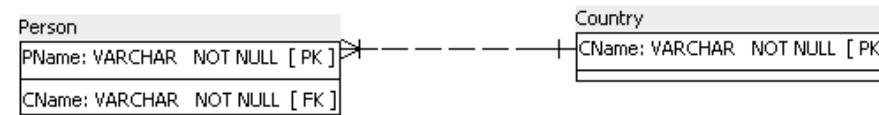
- From Person we can reach 0 or 1 Countries
- From Country we can reach 1 or  $> 1$  Persons



# Born With Crow's Feet

## A Case

- From Person we can reach 1 Countries
- From Country we can reach 1 or  $> 1$  Persons



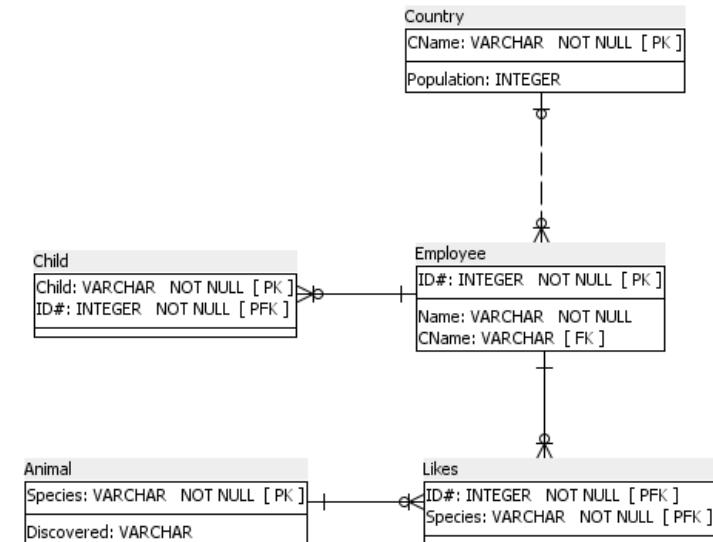
# Relational Implementation Using SQL Power Architect

Child	ID#	Child
1		Erica
1		Frank
2		Bob
2		Frank
6		Frank

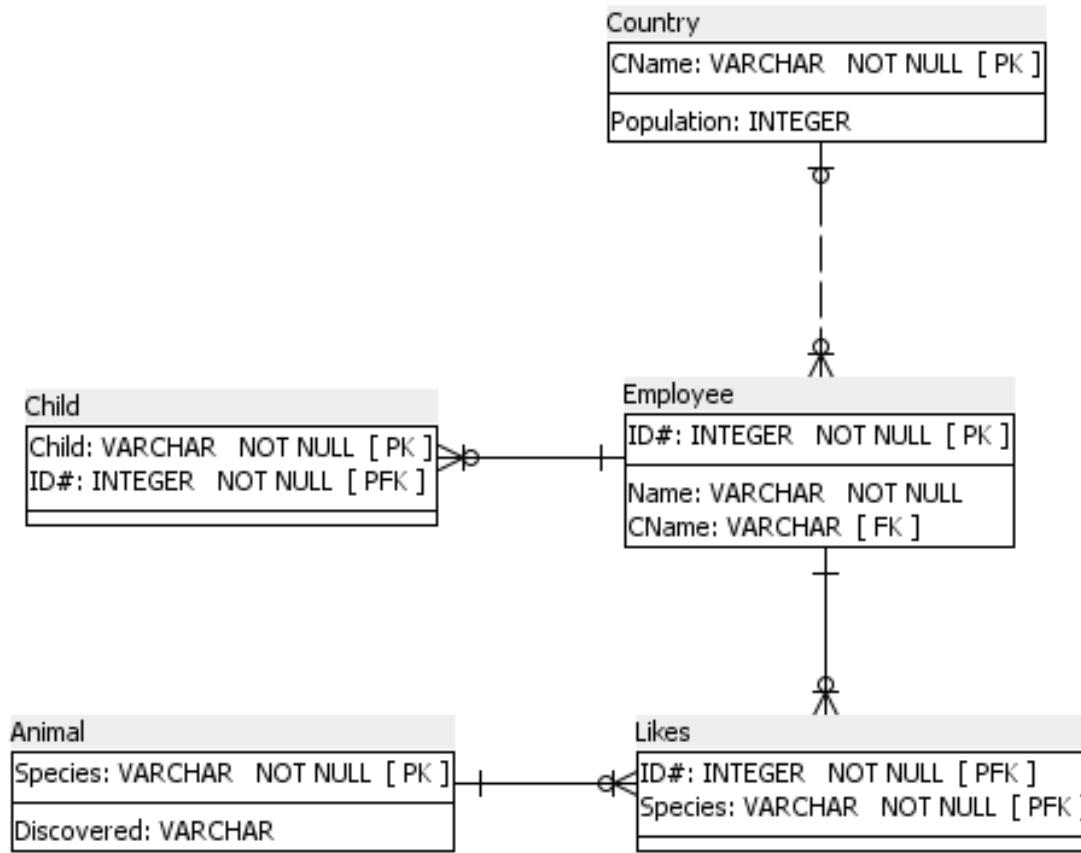
Country	CName	Population
	US	
	IN	1347
	CN	1412
	RU	

Animal	Species	Discovered
Horse	Asia	
Wolf	Asia	
Cat	Africa	
Yak	Asia	
Zebra	Africa	

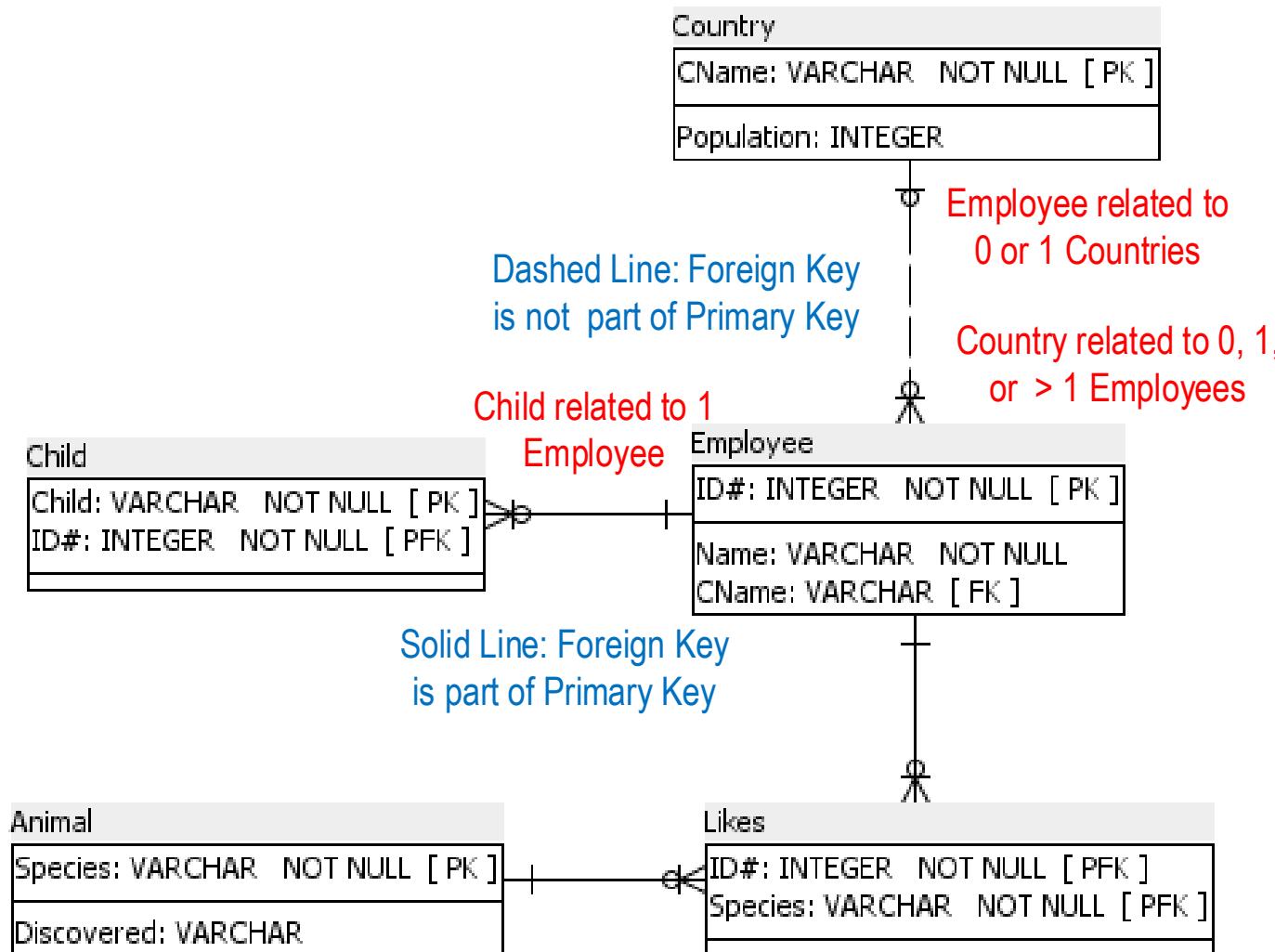
Likes	ID#	Species
	1	Horse
	1	Cat
	2	Cat
	6	Yak



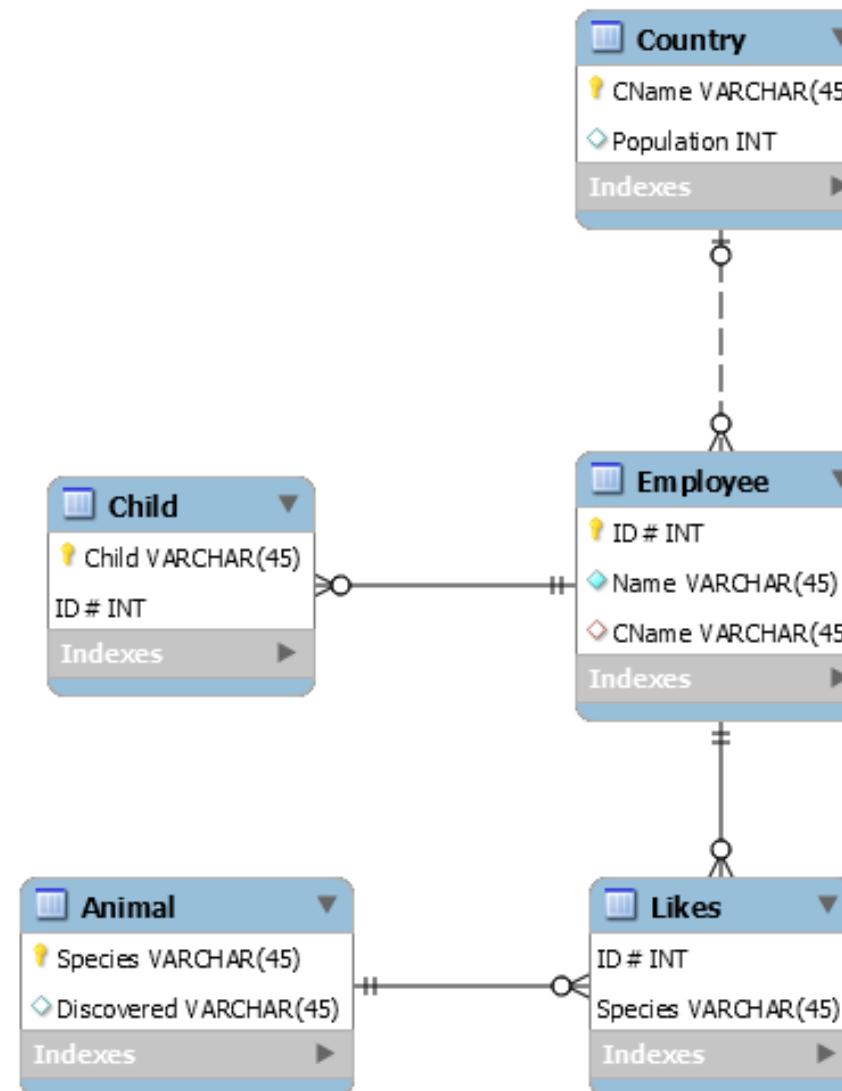
# Same Drawing: Easier To Read



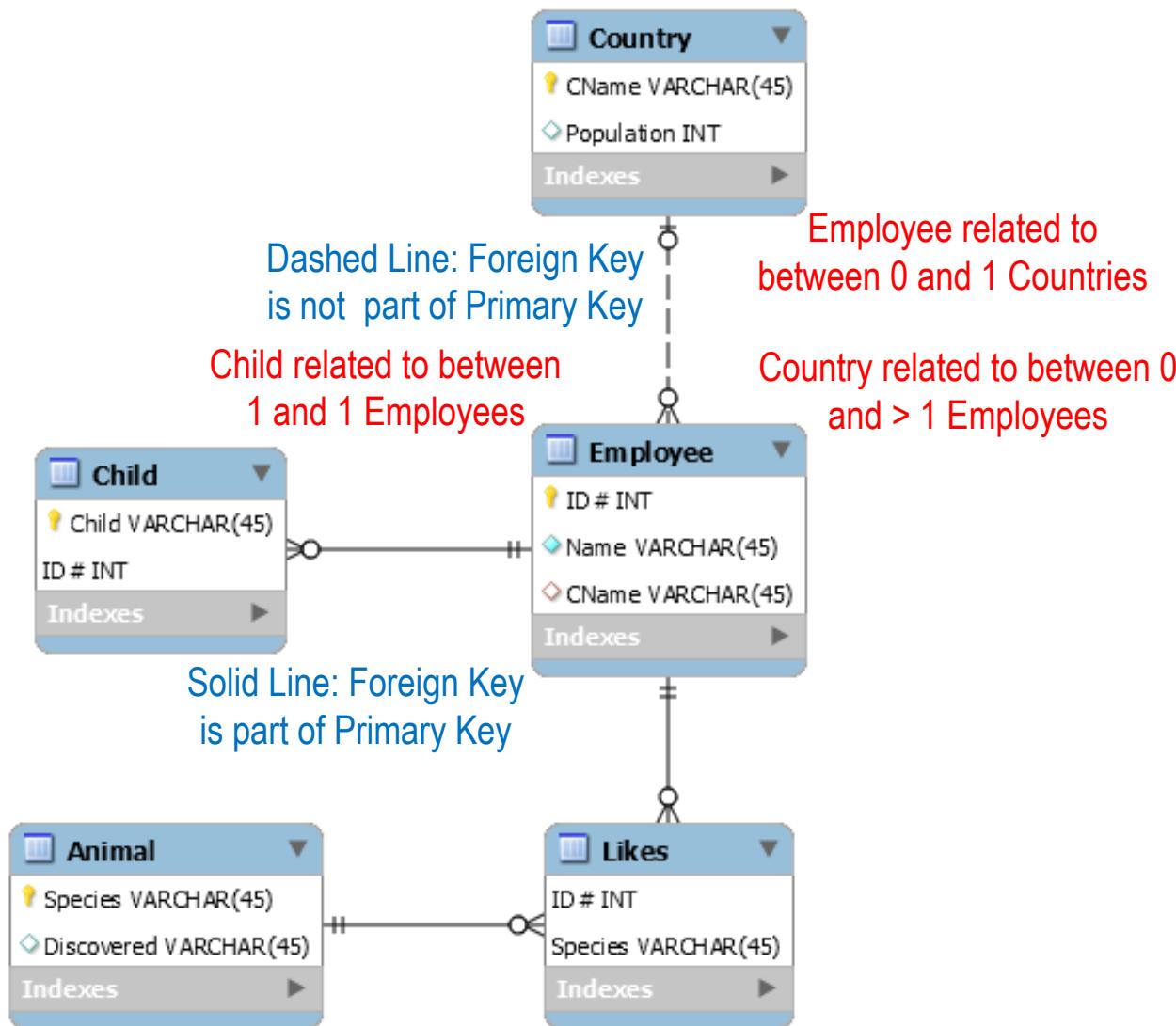
# Observe The Shape Of The Line Ends And The Types Of Lines



# Relational Implementation Using MySQL Workbench



# Observe The Shape Of The Line Ends And The Types Of Lines



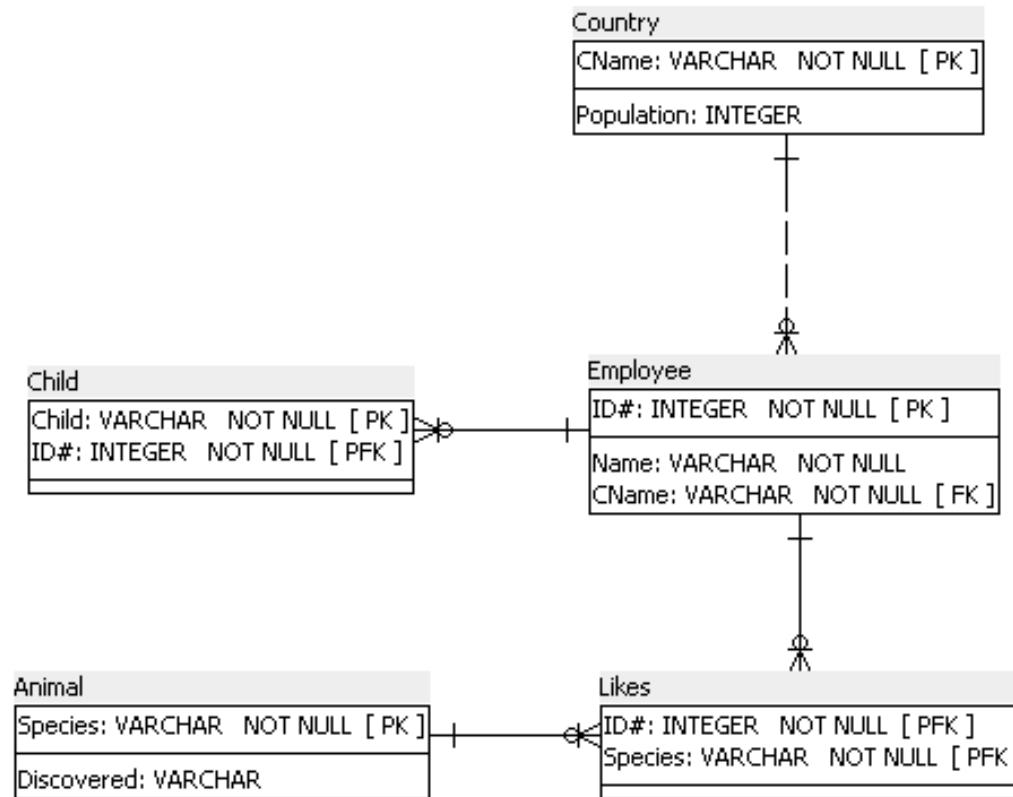
# Pattern of Lines

- The line between Animal and Likes is **solid** because the primary key of the “many side”, Likes, **includes** the primary key of the “one side”, Animal, so Likes “cannot exist” without Animal; the relationship is “**identifying**”
- The line between Employee and Likes is **solid** because the primary key of the “many side”, Likes, **includes** the primary key of the “one side”, Employee, so Likes “cannot exist” without Employee; the relationship is “**identifying**”
- The line between Employee and Child is **solid** because the primary key of the “many side”, Child, **includes** the primary key of the “one side”, Employee, so Child “cannot exist” without Employee; the relationship is “**identifying**”
- The line between Country and Employee is **dashed** because the primary key of the “many side”, Employee, **does not include** the primary key of the “one side”, Country, so Employee “can exist” without Country; the relationship is “**non-identifying**”

- ***This is not a question of the ends of lines “forcing” the pattern of lines***
- In the next slide, we see a slight modification of our example in which all lines have the same pair of endings
- We now require that ***for each Employee the Country of Birth is known***
- Nevertheless, as Cname is not part of the primary key of Country, the line is dashed

# Example

- Assume: Every employee has exactly one Country (that is we know the country of birth)



# Alternative Implementation for Born

- We **need** an “in-between” table for Likes because it is **many-to-many**
- We **do not need** an “in-between” table for Born because it is **many-to-one**
- But we can implement Born using such an “in-between” table
  - » Note that CName is not part of the primary key of Born

Employee	ID#	Name	Born	ID#	CName	Country	CName	Population
1		Alice	1	US		US		
2		Bob	2	IN		IN	1347	
4		Carol	5	IN		CN	1412	
5		David	6	CN		RU		
6		Bob						

The diagram illustrates the relationships between three tables: Employee, Born, and Country. Red arrows point from specific rows in the Employee table to corresponding rows in the Born table, and from the Born table to the Country table. Specifically, Employee row 1 points to Born row 1 (CName US), Employee row 2 points to Born row 2 (CName IN), Employee row 4 points to Born row 5 (CName IN), Employee row 5 points to Born row 6 (CName CN), and Employee row 6 points to Born row 1 (CName US). Additionally, Born row 1 points to Country row 1 (CName US), Born row 2 points to Country row 2 (CName IN), Born row 5 points to Country row 3 (CName CN), and Born row 6 points to Country row 4 (CName RU).

# Alternative Implementation for the Example

Country	Cname	Population
US		
IN	1347	
CN	1412	
RU		

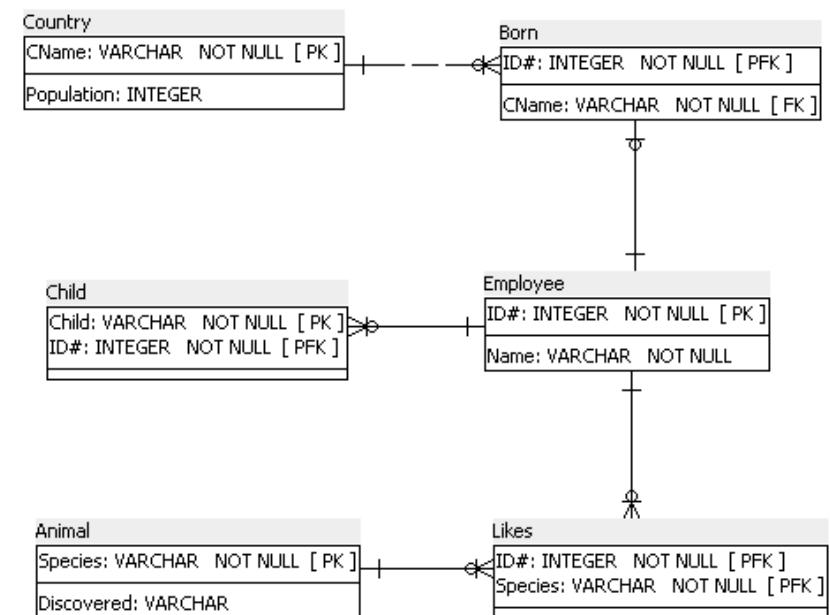
Born	ID#	CName
	1	US
	2	IN
	5	IN
	6	CN

Child	ID#	Child
1		Erica
1		Frank
2		Bob
2		Frank
6		Frank

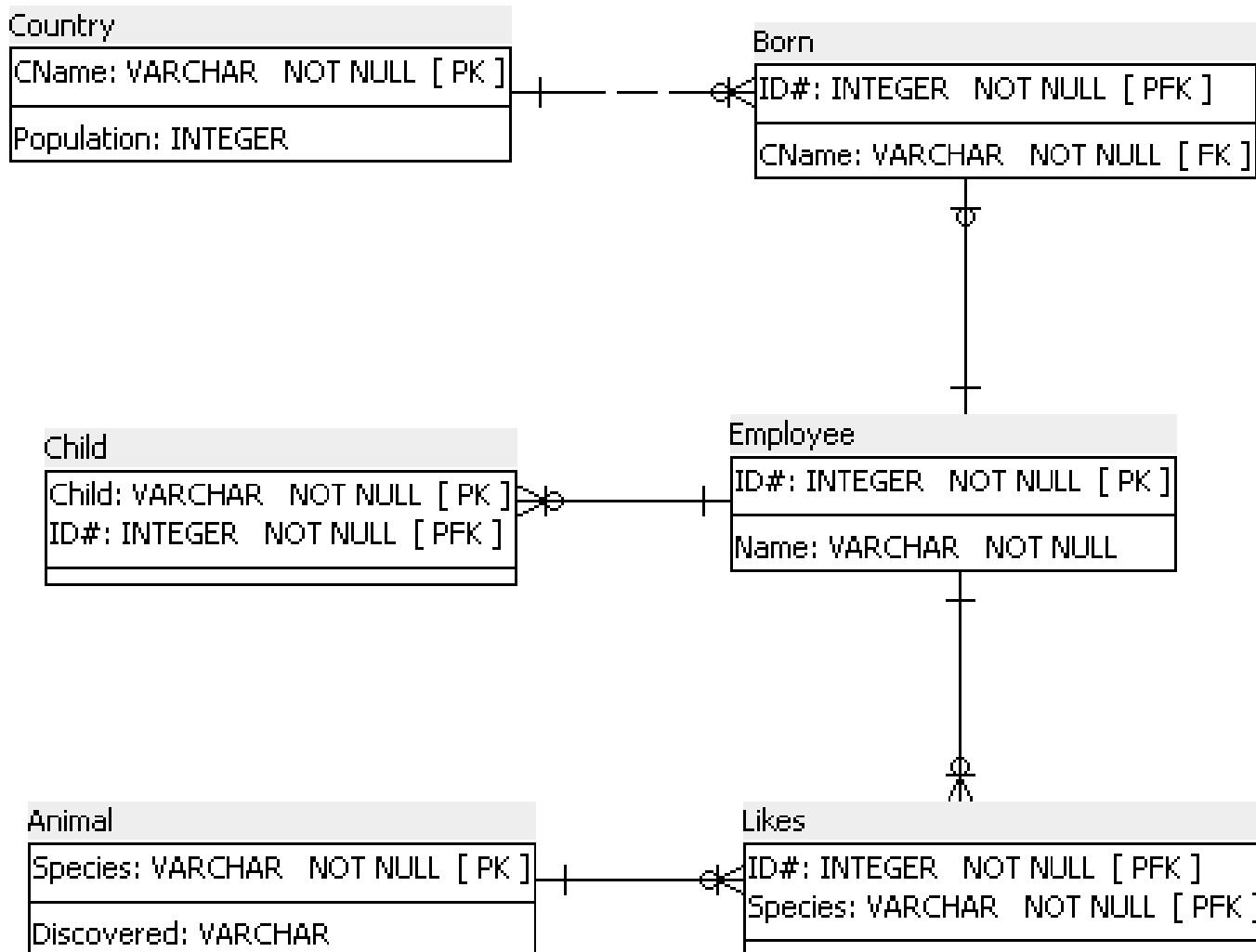
Employee	ID#	Name
	1	Alice
	2	Bob
	4	Carol
	5	David
	6	Bob

Animal	Species	Discovered
Horse	Asia	
Wolf	Asia	
Cat	Africa	
Yak	Asia	
Zebra	Africa	

Likes	ID#	Species
	1	Horse
	1	Cat
	2	Cat
	6	Yak



# Same Drawing: Easier To Read



## General case

- We have a relationship  $R$  among entity sets  $E_1, E_2, \dots, E_n$ , with properties  $P_1, P_2, \dots, P_m$
- Each  $E_i$  is implemented as a table
- We can always implement  $R$  as a table with foreign key constraints referencing  $E_1, E_2, \dots, E_n$ , and with  $R$  also storing properties  $P_1, P_2, \dots, P_m$

## Special case: $R$ is binary many-to-one from $E_1$ to $E_2$

- We can, if we like, avoid introducing a table for  $R$
- We implement  $R$  as a foreign key constraint in  $E_1$  referencing  $E_2$ , and with  $E_1$  also storing properties  $P_1, P_2, \dots, P_m$

# Which Implementation to Use for Born? (And for Such Relationships in General)

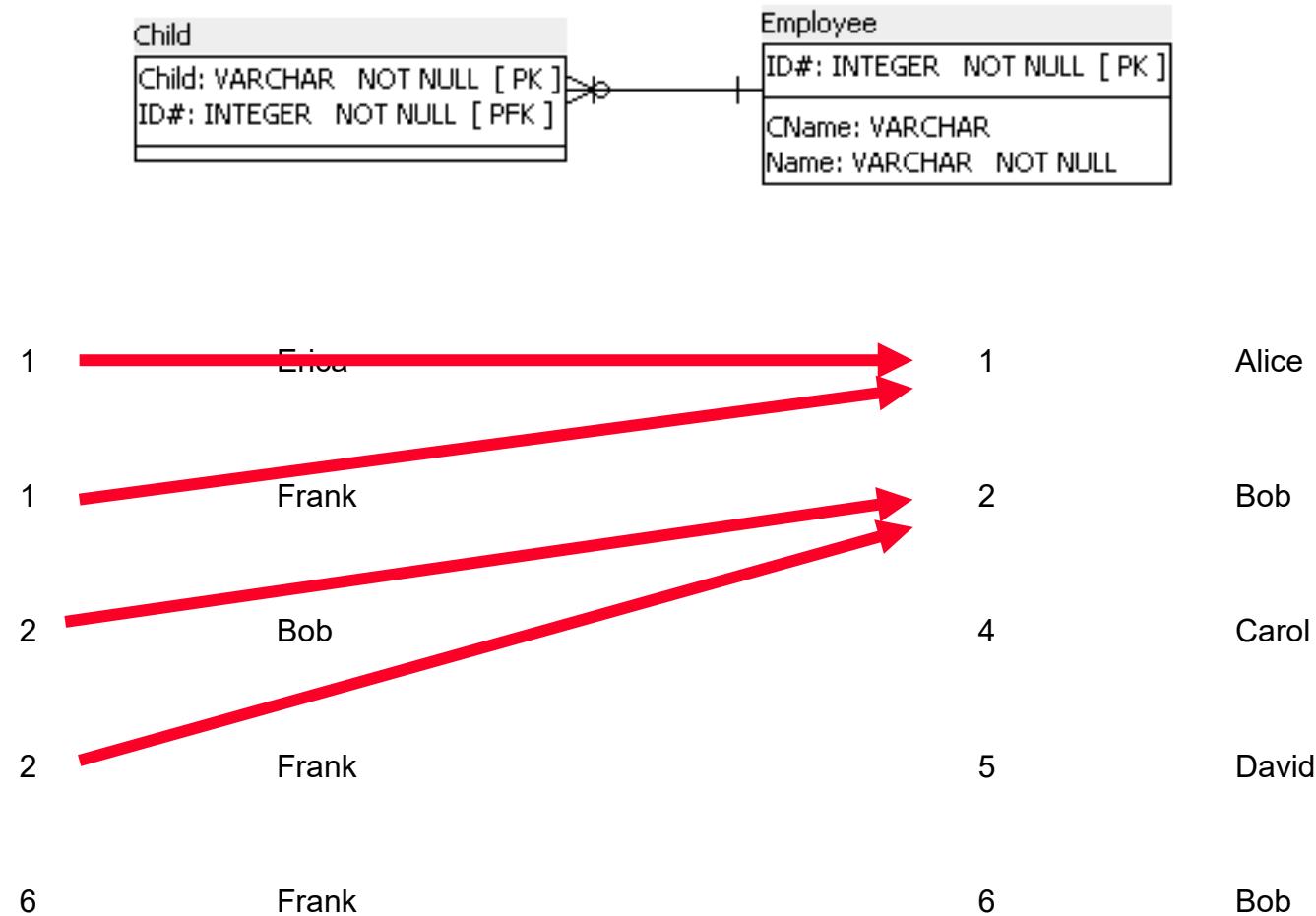
- We cannot give a general rule
- Our original implementation would introduce NULLs for Persons for whom we do not know the country of birth
- Our alternate implementation would introduce an additional table, which would have no NULLs
- ***For the purpose of the class, we will always use our original implementation (without the additional table), to have better exercises and to reinforce the fundamental nature of binary many-to-one relationships***
- ***So do this for all your work when relevant***

# To Remember!

- Structurally, a relational database consists of
  1. A ***set of tables with identifiers (primary keys)***
  2. A ***set of many-to-one binary relationships*** between them, induced by foreign key constraints  
In other words; ***a set of functions (in general partial), each from a table into a table***
- When designing a relational database, you ***should*** specify both (or you will produce a bad specification)
  - » Technically, tables are enough, but this a very bad practice as you do not specify the relationships between tables

# Many-To-One Mapping From Child to Employee (To Reiterate)

- Partial function from a set of rows into a set of rows



# Very Bad Relational Implementation

- Tables are listed with attributes, specifying only which are in the primary key
- Foreign key constraints are not specified
  - So, the database does not know what to enforce

Country
CName: VARCHAR NOT NULL [ PK ]
Population: INTEGER

Child
Child: VARCHAR NOT NULL [ PK ]
ID#: INTEGER NOT NULL [ PK ]

Employee
ID#: INTEGER NOT NULL [ PK ]
Name: VARCHAR NOT NULL
CName: VARCHAR

Animal
Species: VARCHAR NOT NULL [ PK ]
Discovered: VARCHAR

Likes
ID#: VARCHAR NOT NULL [ PK ]
Species: INTEGER NOT NULL [ PK ]

# Terrible Relational Implementation

- Even primary keys are not specified

Country
CName: VARCHAR NOT NULL
Population: INTEGER

Child
Child: VARCHAR NOT NULL
ID#: INTEGER NOT NULL

Employee
ID#: INTEGER NOT NULL
CName: VARCHAR
Name: VARCHAR NOT NULL

Animal
Species: VARCHAR NOT NULL
Discovered: VARCHAR

Likes
Species: VARCHAR NOT NULL
ID#: INTEGER NOT NULL

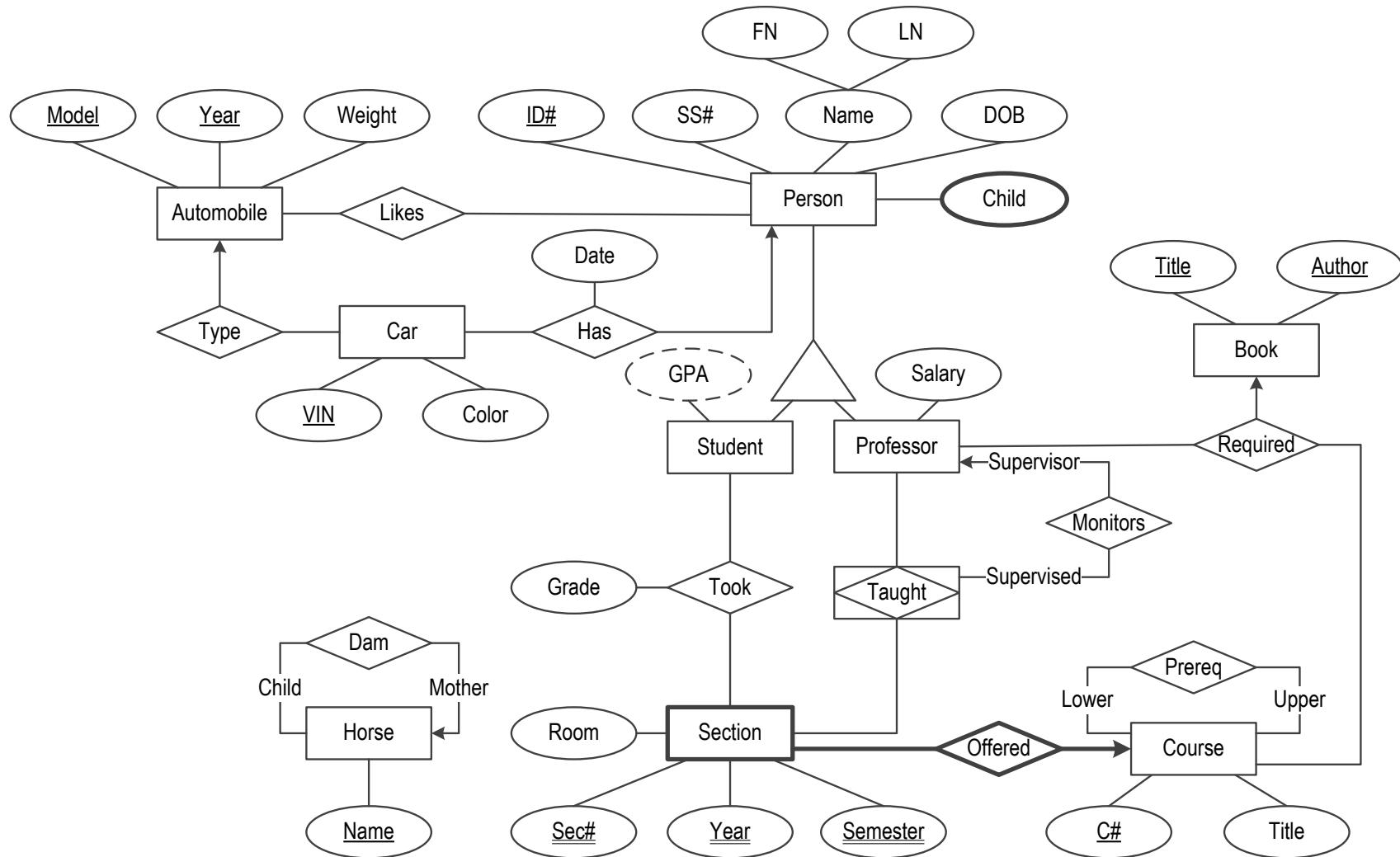
## From ER Diagram to Relational Database

- We now convert our big ER diagram into a relational database
- We specify
  - » Attributes that must not be NULL
  - » Primary keys
  - » Keys (beyond primary)
  - » Foreign keys and what they reference
  - » Cardinality constraints
  - » Some additional “stubs”

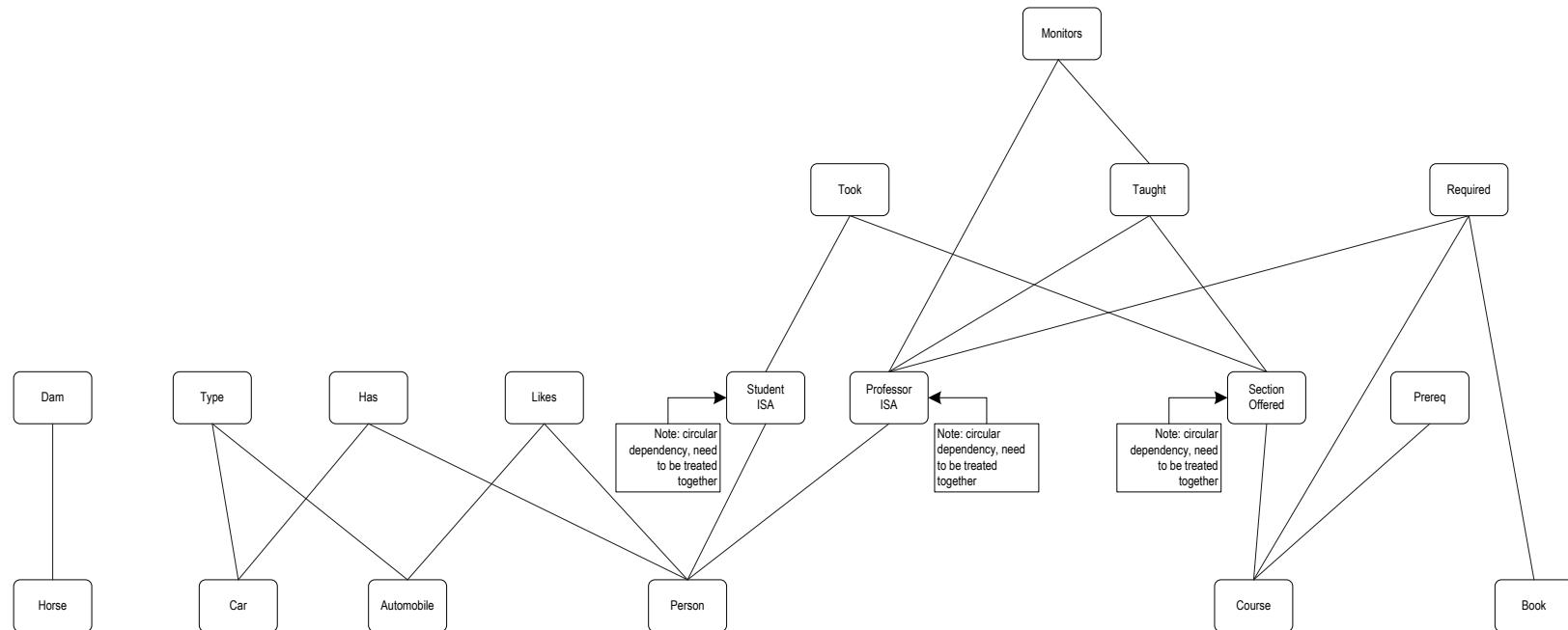
# From ER Diagram to Relational Database

- We both give a narrative description, similar to actual SQL DDL (so we are learning about actual relational databases) and SQL Power Architect
- We should specify domains also, but we would not learn anything from this here, so we do not do that here
- This is really a comprehensive example and not an algorithmic procedure
- It provides intuition to handle cases not explicitly appearing in the example
- We are **not** trying to improve the design: we will do that later in the Normalization Unit: we just implement the ER diagram
- We go bottom up, in the same order as the one we used in constructing the ER diagram

# Our ER Diagram



# Hierarchy for Our ER Diagram



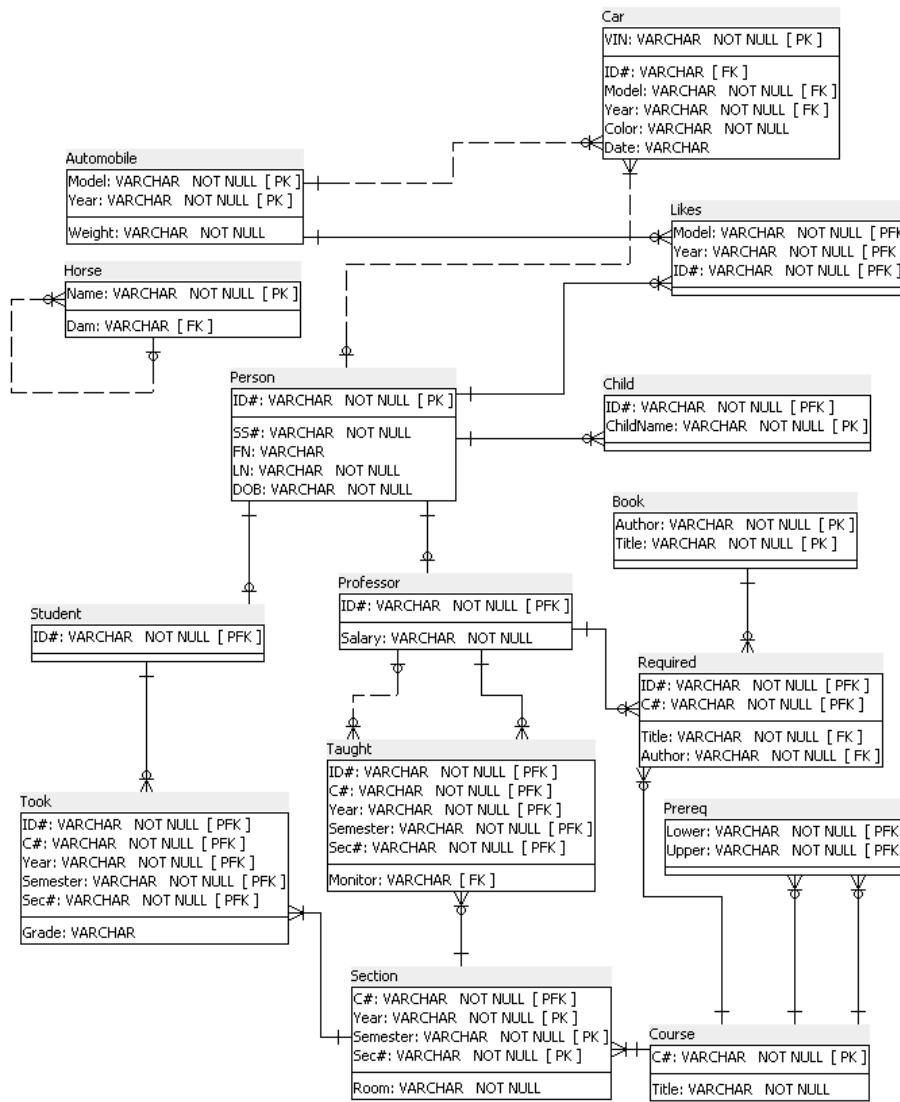
## Annotations For The ER Diagram

- The graph describing the Dam relationship is a forest of rooted trees
- LN, SS#, DOB in Person are always known
- No two Persons can have the same SS#
- Color in Car is always known
- Weight in Automobile is always known
- Type is total
- Every Person Has at least 2 Cars
- Salary is always known
- Title in Course is always known
- Prereq is a DAG (Directed Acyclic Graph)
- Each Course has at least 1 Section related to it through Offered
- A Section has between 3 and 50 Students

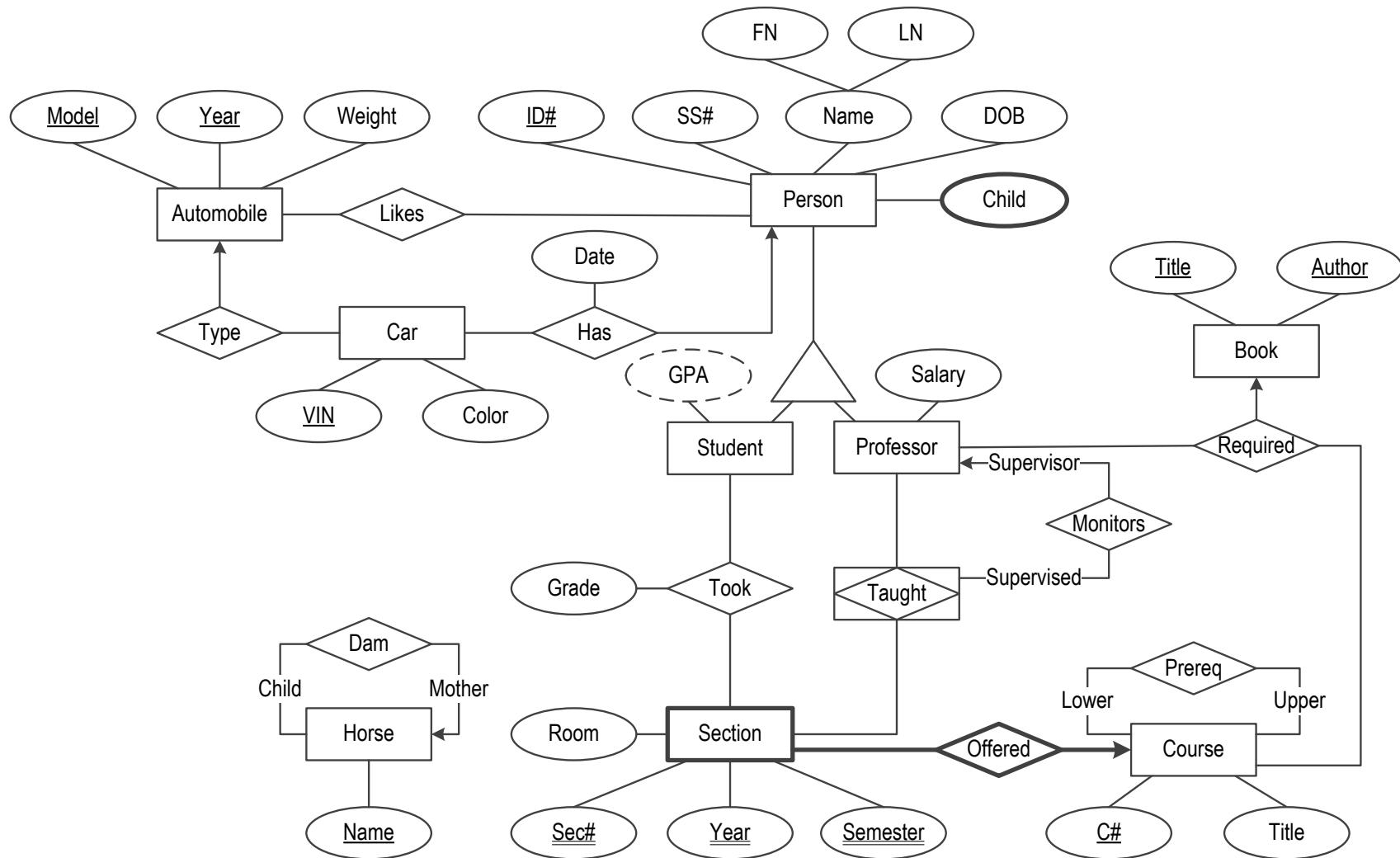
## Annotations For The ER Diagram

- GPA: Computed Attribute for Student by adding all the known numeric Grades, dividing by the number of Sections that the Student Took
- Supervisor and Supervised cannot be the same Professor

# We Will Produce



# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



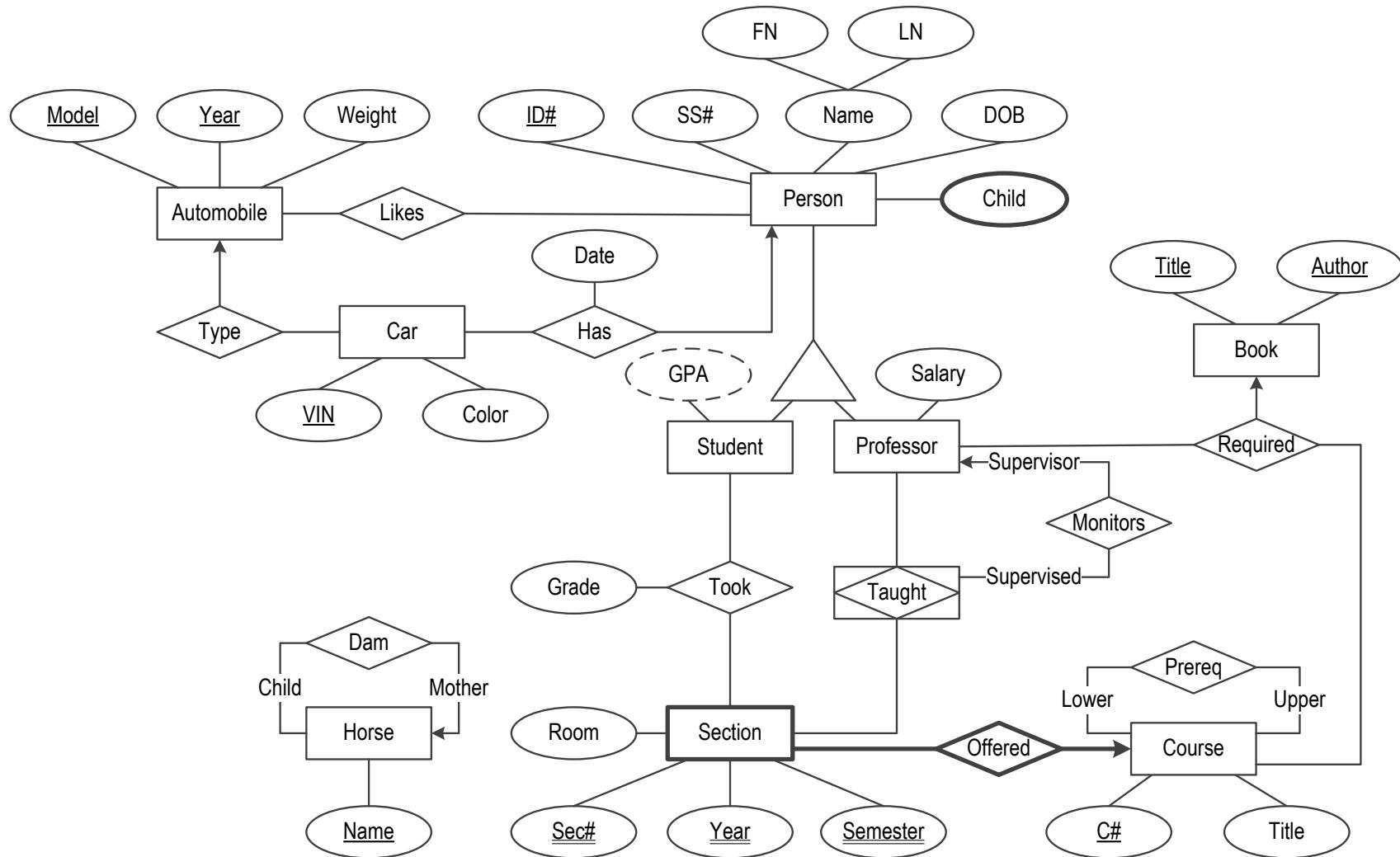
# Relational Database So Far

Horse
Name: VARCHAR NOT NULL [ PK ]

## Comments

- We will handle the Dam relationship at the end
- This is a little subtle case so we will do it later
- But if we were experienced, we could do it now

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



# Relational Database So Far

Horse

Name: VARCHAR NOT NULL [ PK ]
-------------------------------

Person

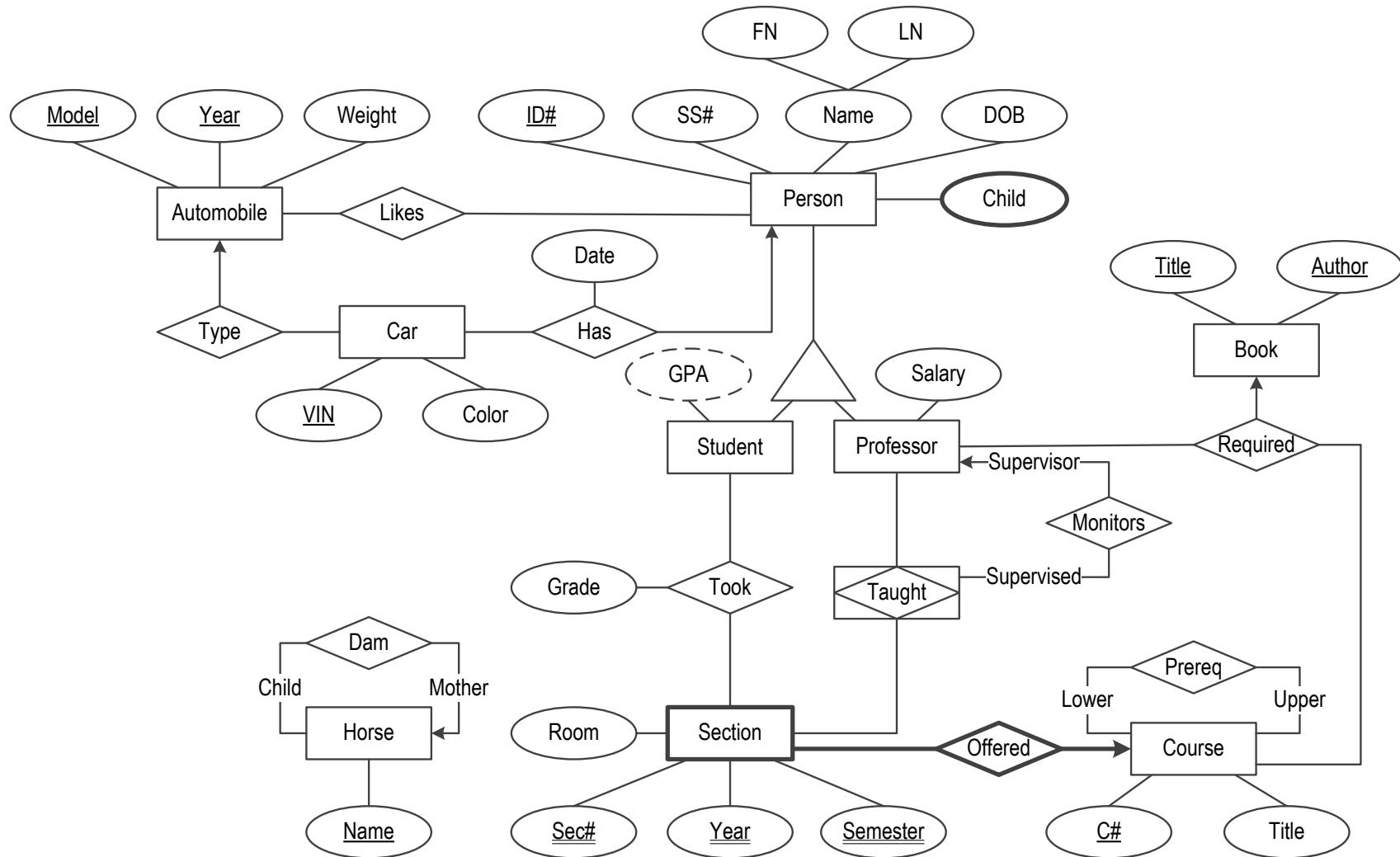
ID#: VARCHAR NOT NULL [ PK ]
SS#: VARCHAR NOT NULL
FN: VARCHAR
LN: VARCHAR NOT NULL
DOB: VARCHAR NOT NULL

## Annotations

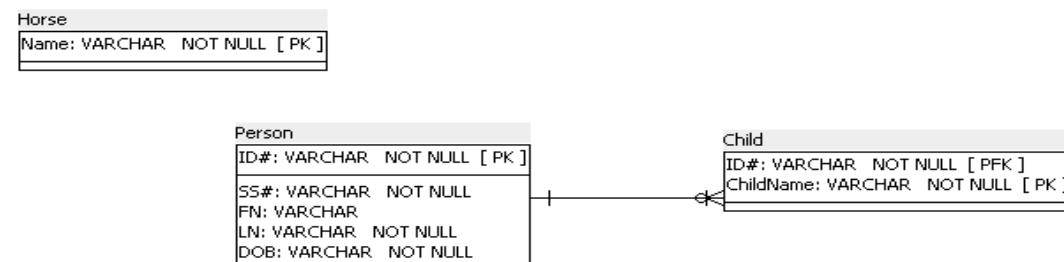
- SS# is UNIQUE in Person

- Note that annotation refers to specific tables and specific columns in the tables
  - » Once our relational implementation is done, we no longer (at least in theory) refer to the ER diagram
- UNIQUE can be specified using an index in a relational design; if we do that, we do not need an annotation

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



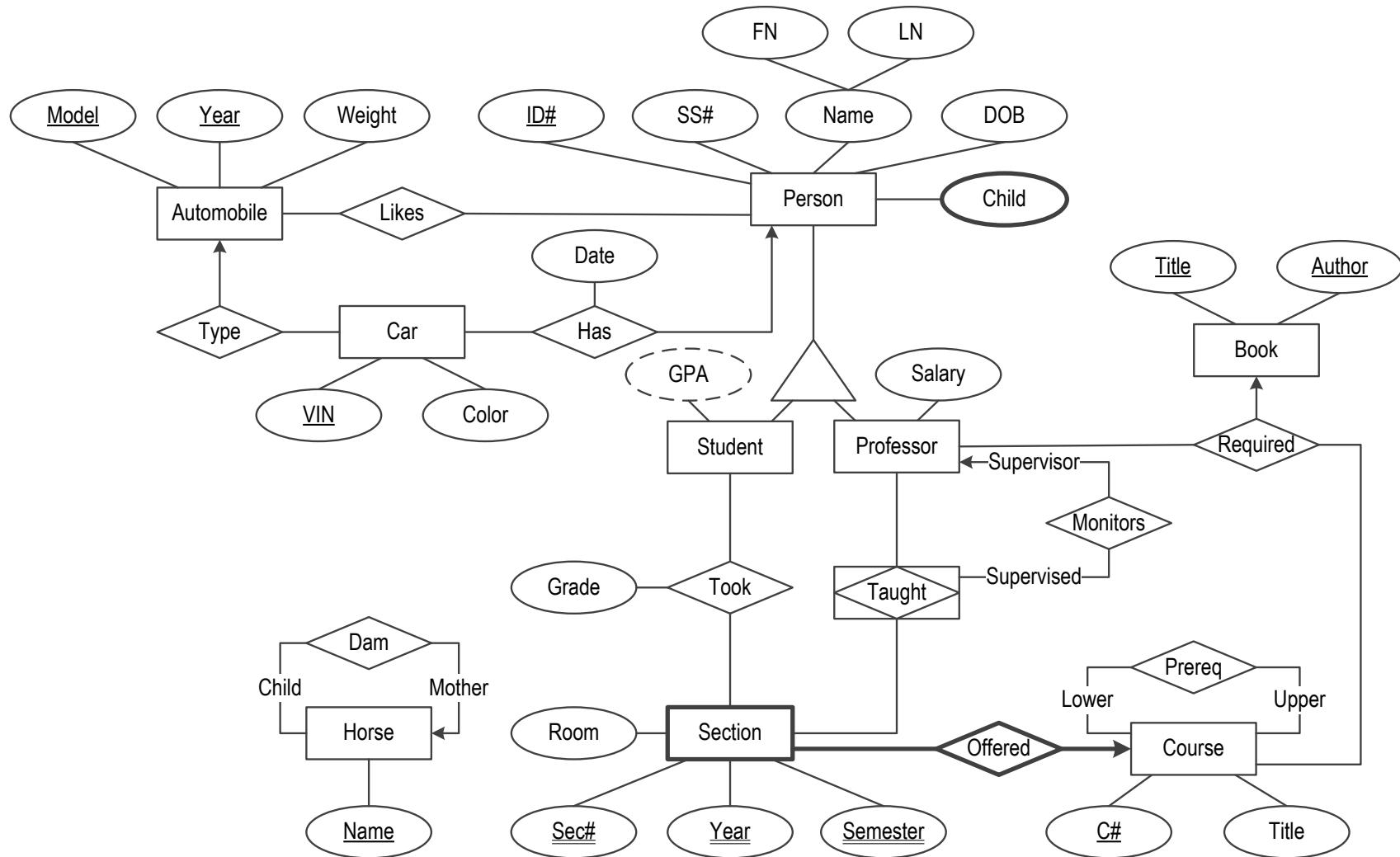
# Relational Database So Far



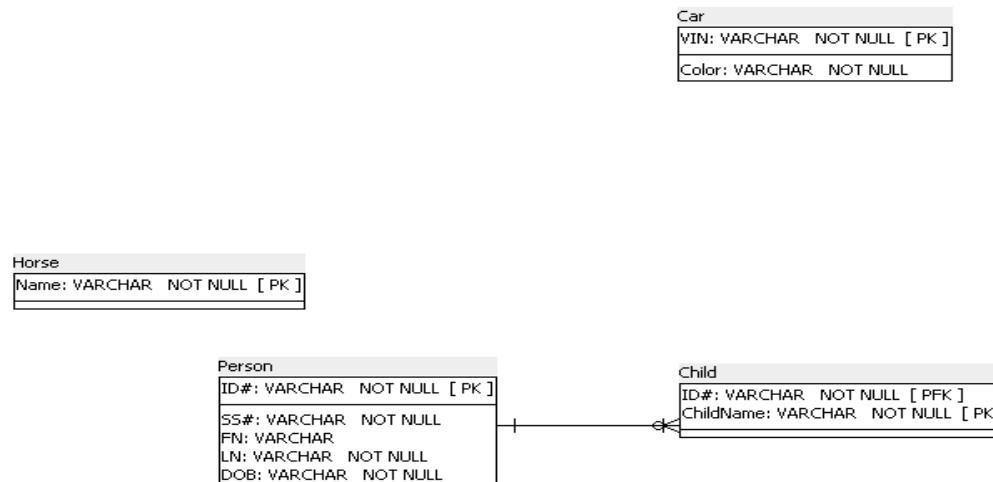
## Comments

- Note the solid line
- We see a relationship that is identifying
- Child needs to be related to Person to be identified

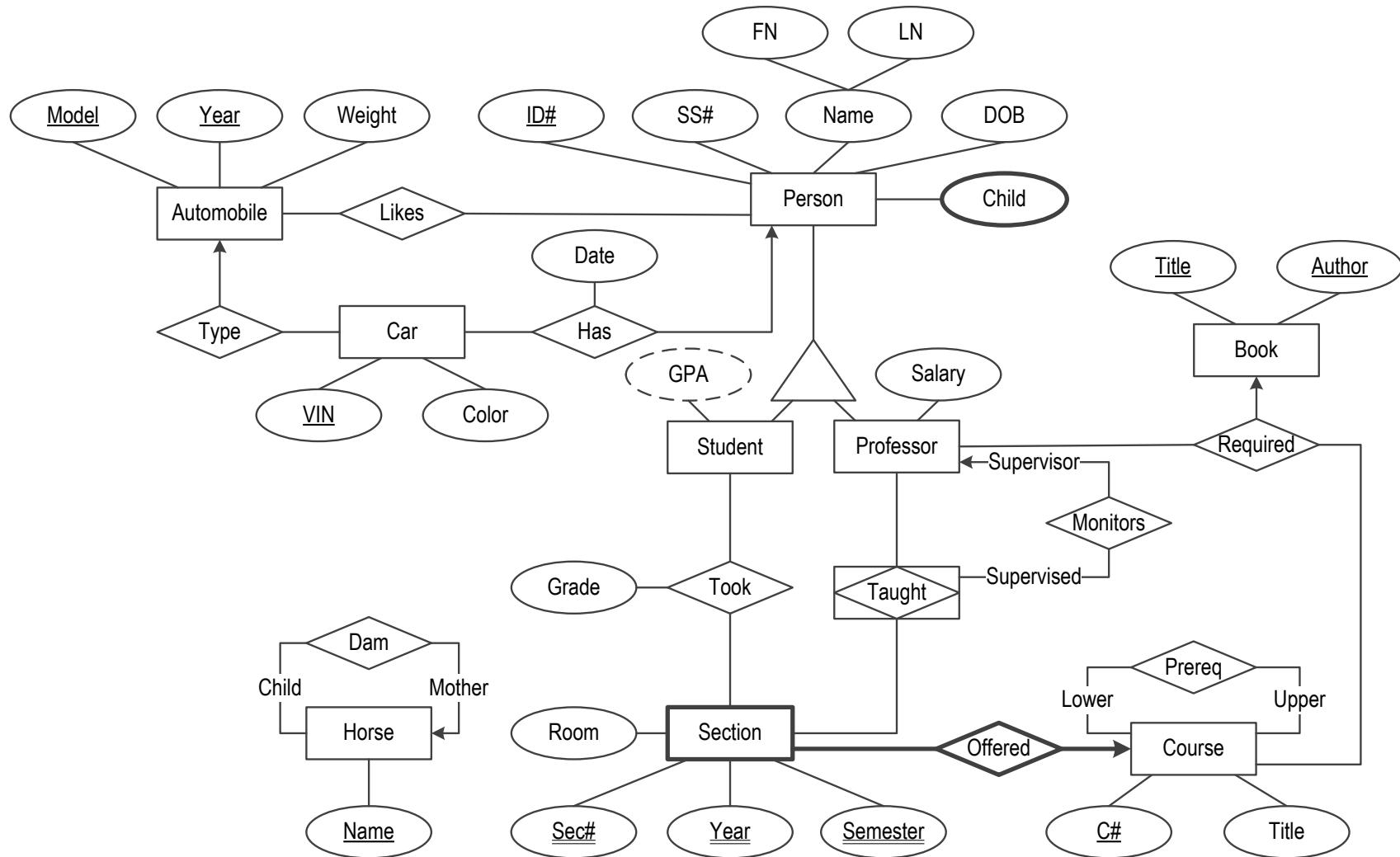
# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



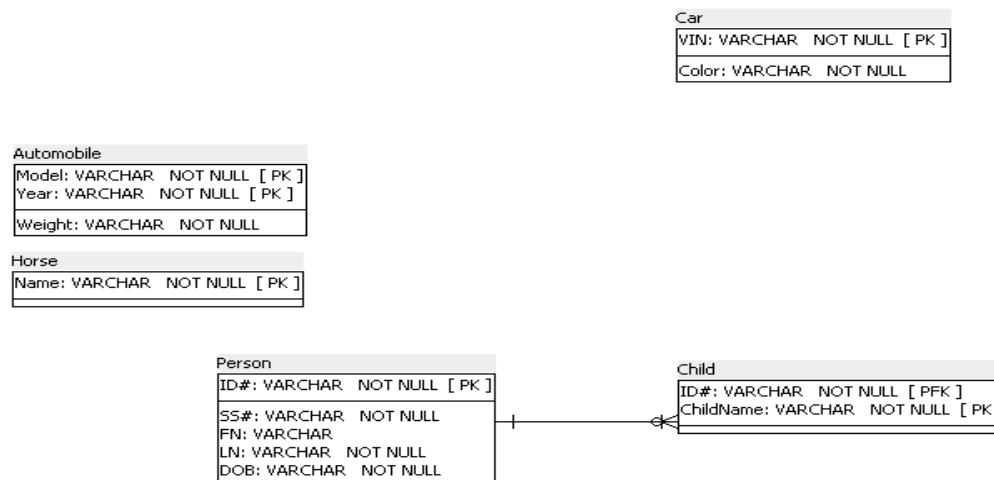
# Relational Database So Far



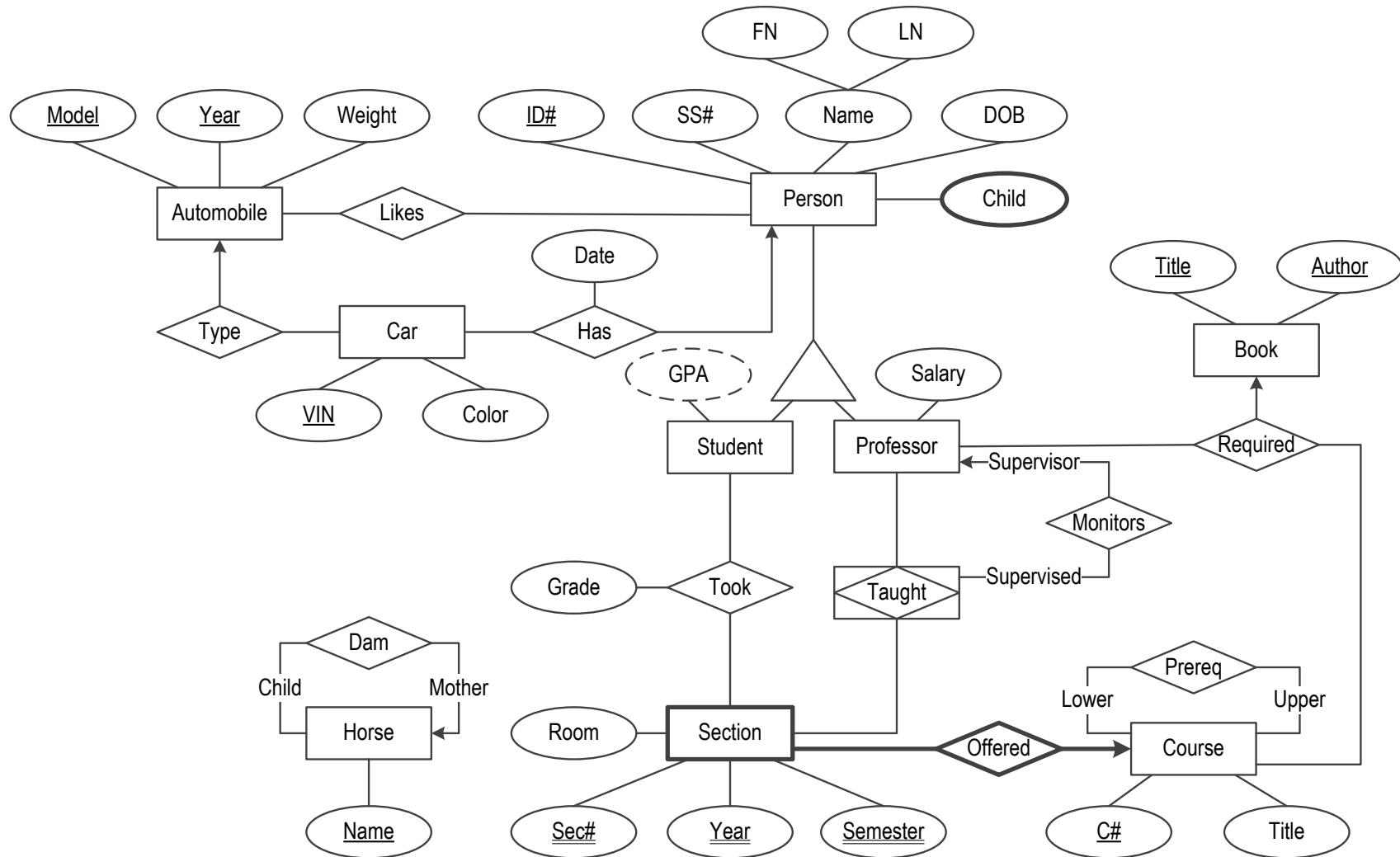
# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



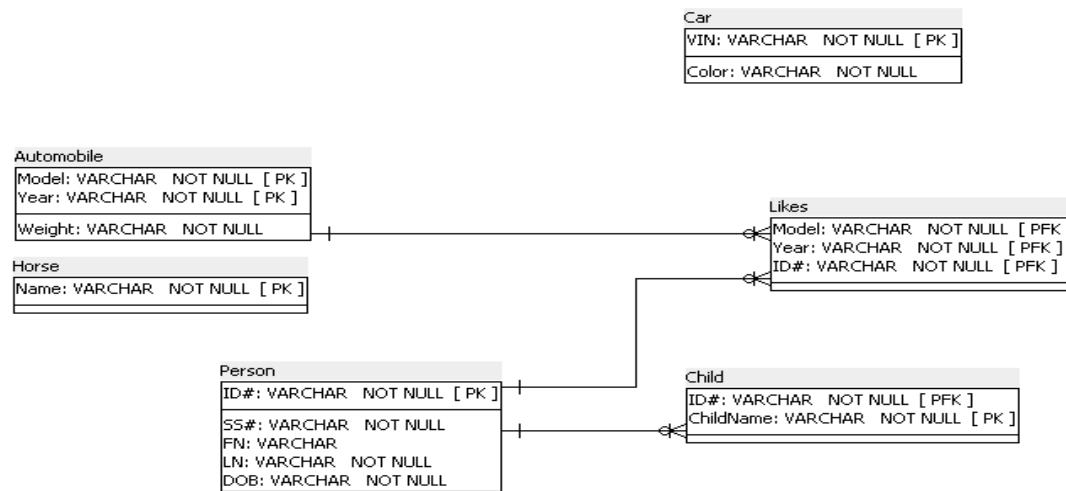
# Relational Database So Far



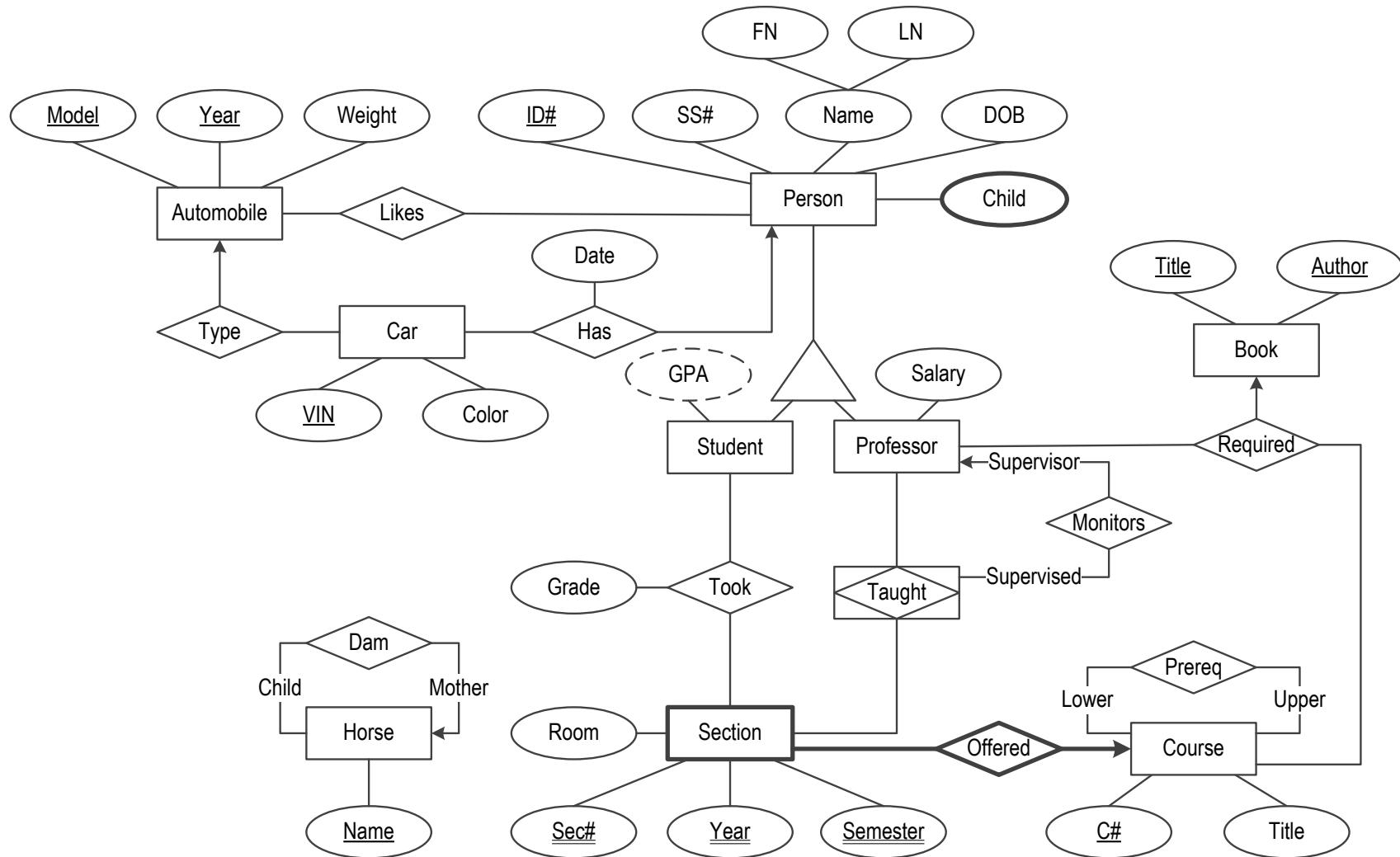
# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



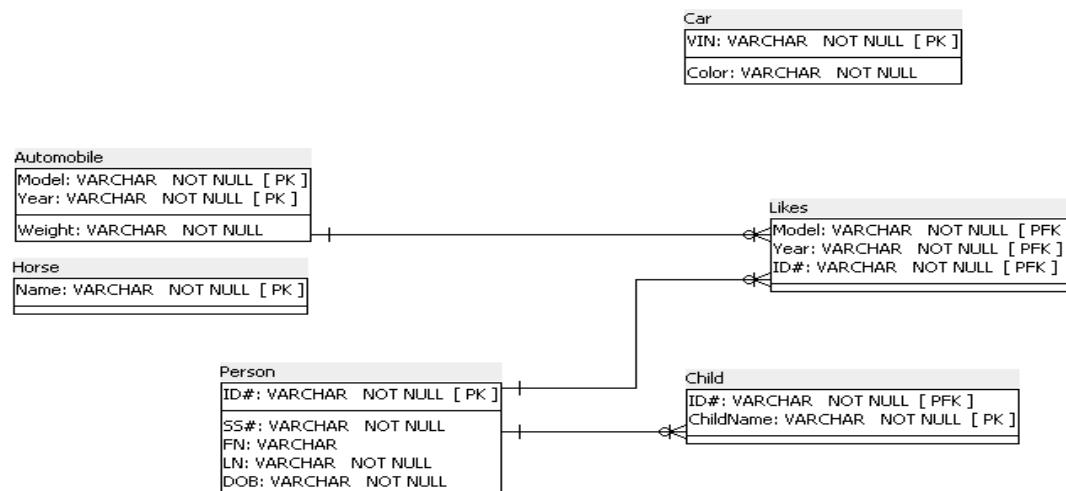
# Relational Database So Far



# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It

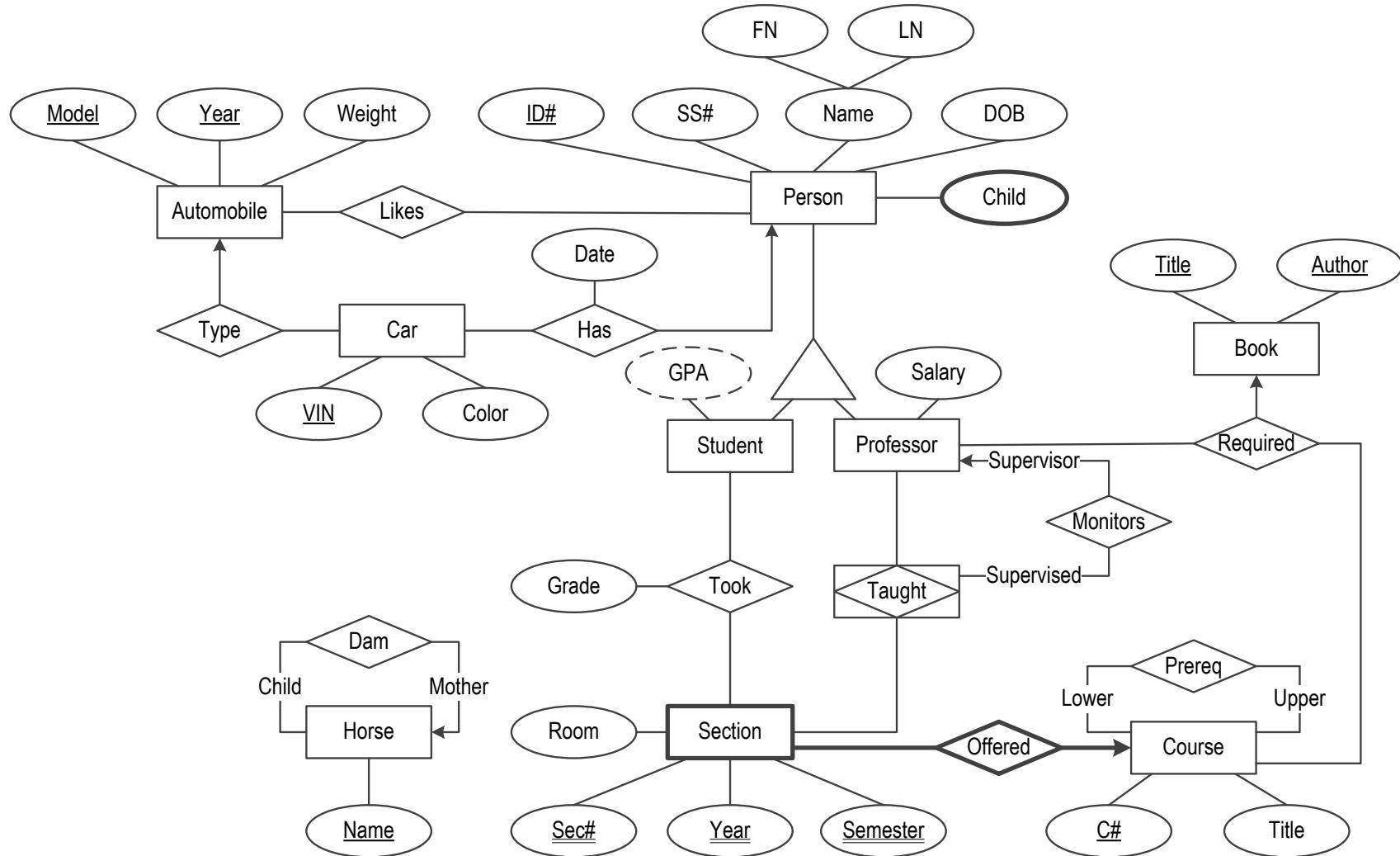


# Relational Database So Far

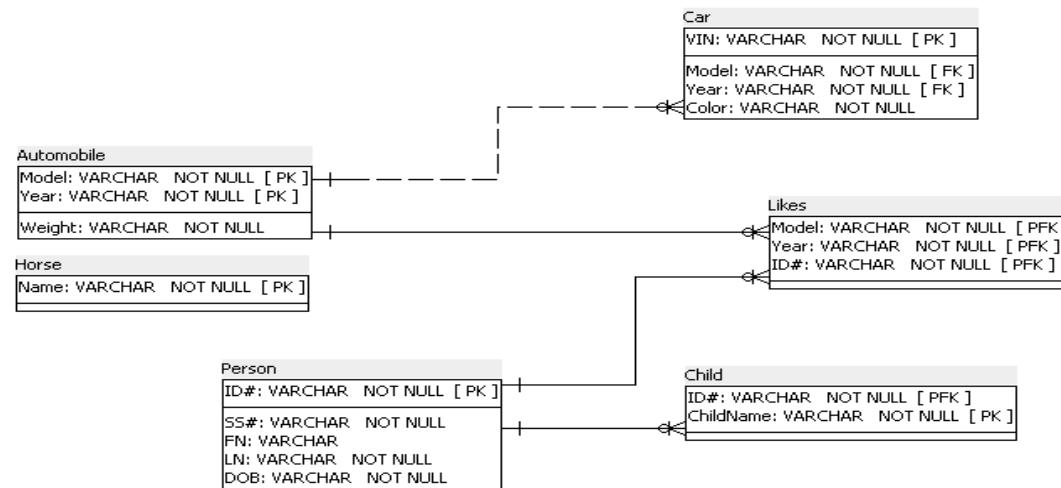


- ***Note: the following is bad/incorrect,***  
replacing one line by two lines
  - » Foreign Key (Model) References Automobile
  - » Foreign Key (Year) References Automobile
- There are 2 induced binary many-to-one relationships
  - » From Likes to Person
  - » From Likes to Automobile

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It

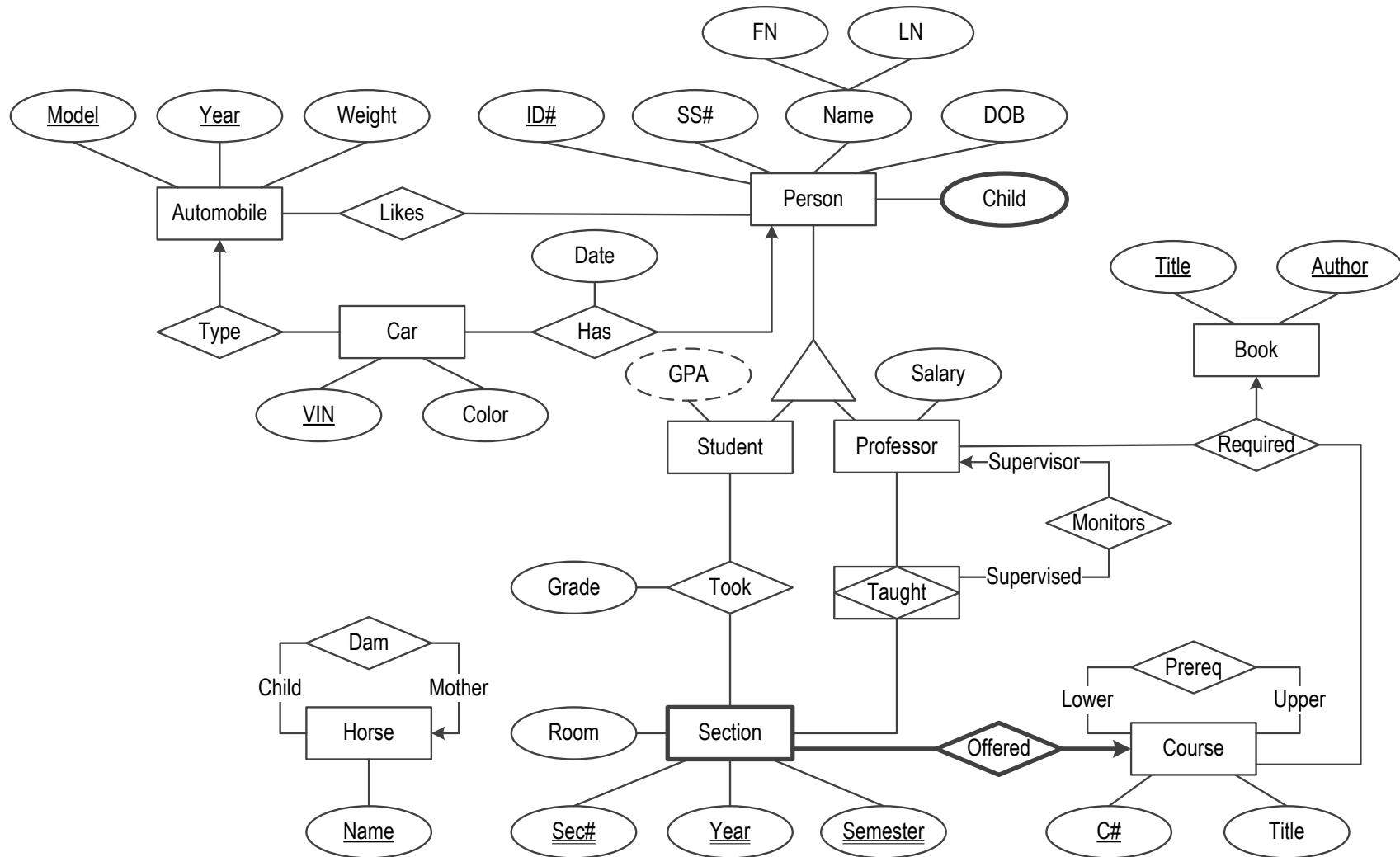


# Relational Database So Far

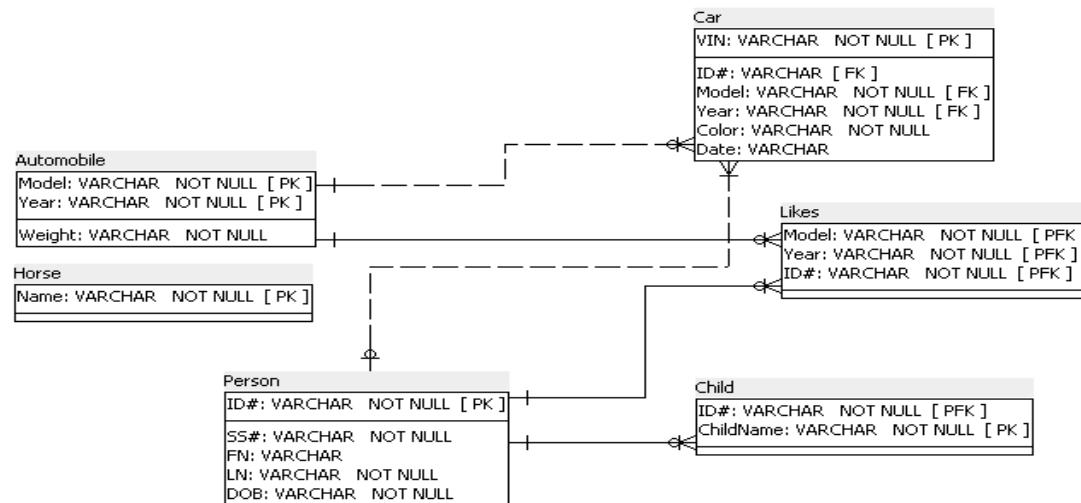


- Type is a binary many-to-one relationship, so we do not add a table to store this relationship
- Note the dashed line
- This is the first time we see a relationship that is not identifying
- Car does not need to be related to Automobile to be identified

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



# Relational Database So Far



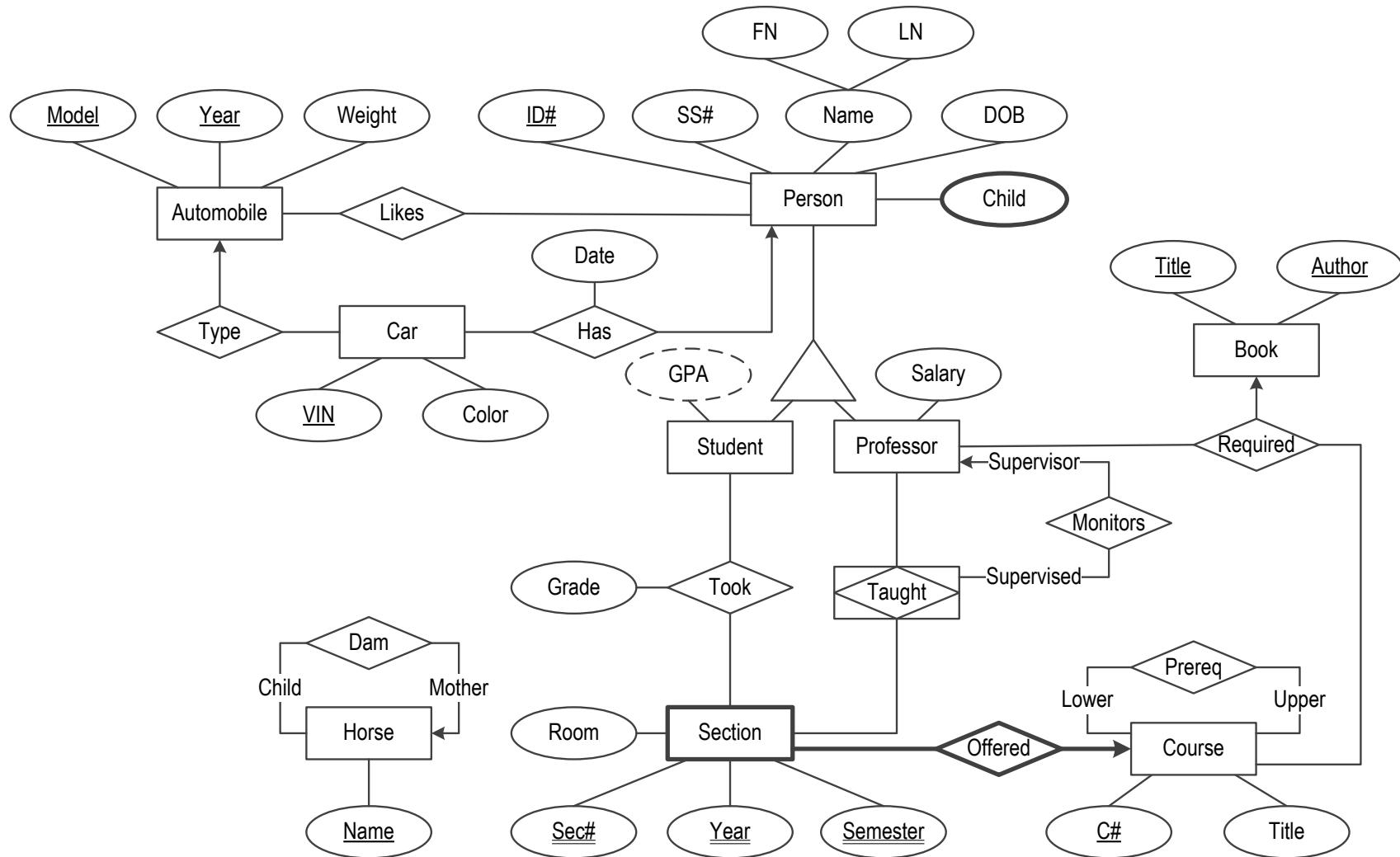
## Annotations

- Every ID# must appear in at least two distinct tuples of table Car

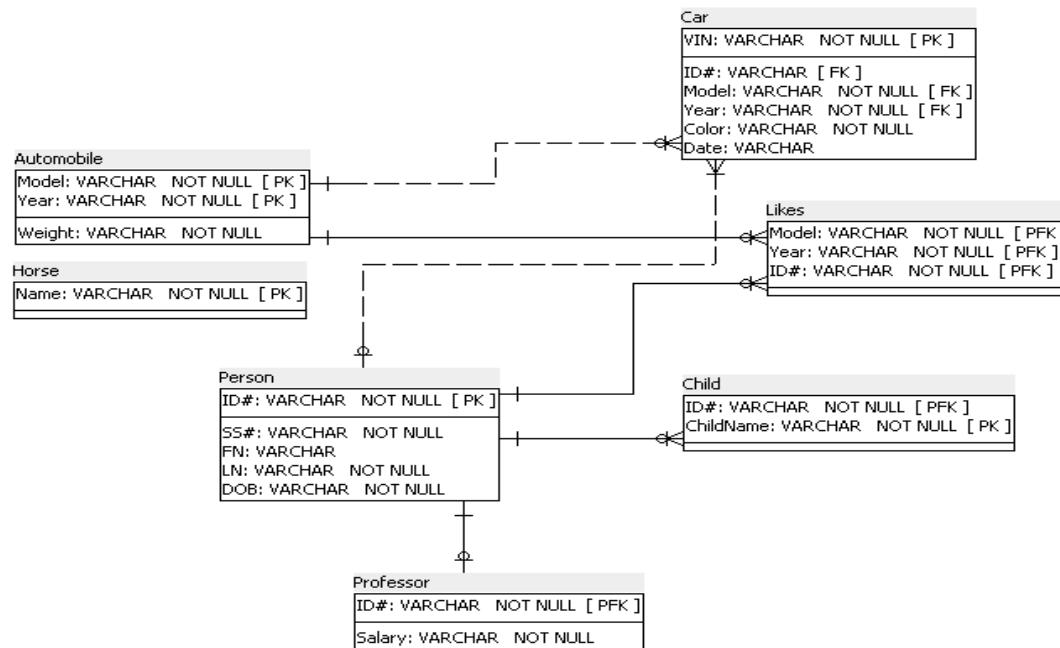
## Comments

- Our Crow's feet notation allows us to specify only the weaker condition: Every ID# must appear in at least one tuple of table Car
- Note that the Date is stored inside Car
- It could not have been stored inside Person
- Note that the number 2 does not impact on the structure of the design on the correctness of an instance
- We would have the same design if instead of 2 we had 500
- ***That's why the arrow notation was good enough and we did not use cardinality in our ER diagrams***

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



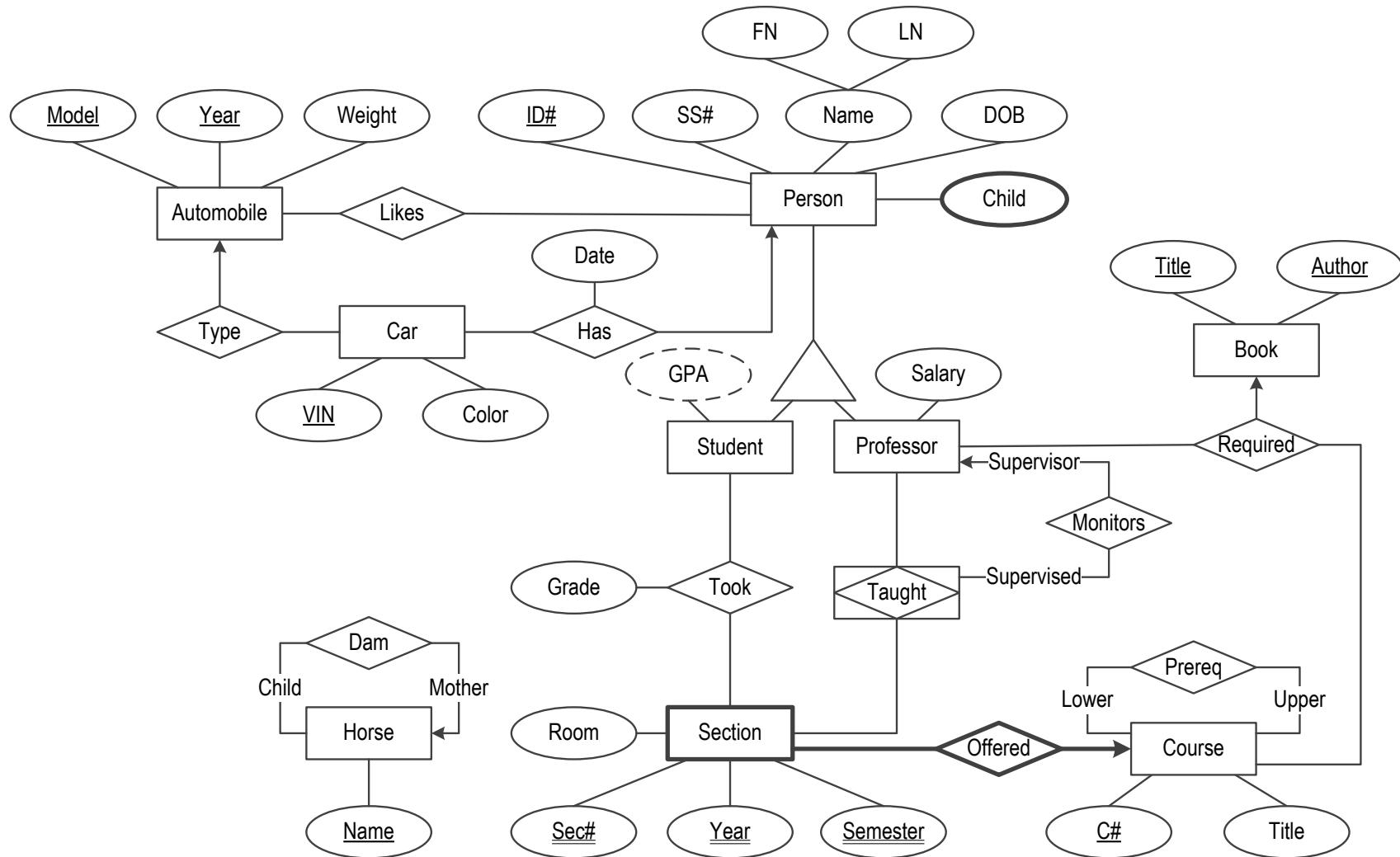
# Relational Database So Far



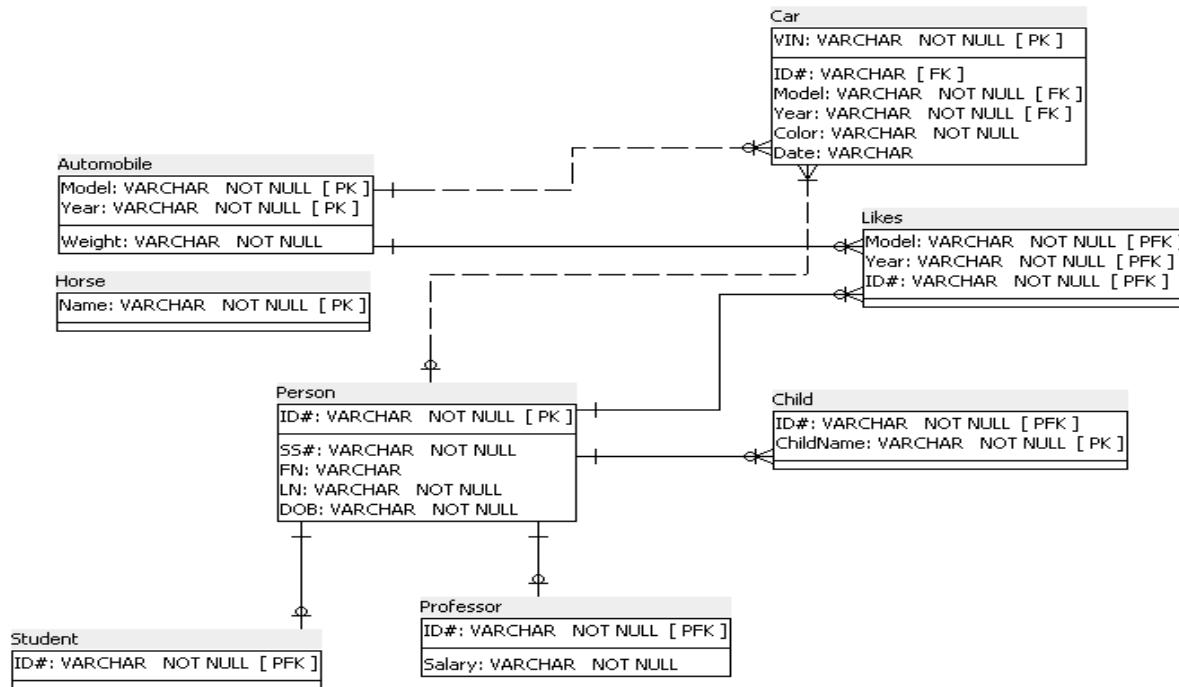
## Comments

- Note the ends of the line
- From Person you reach 0 or 1 Professors
- From Professor you reach 1 Person

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



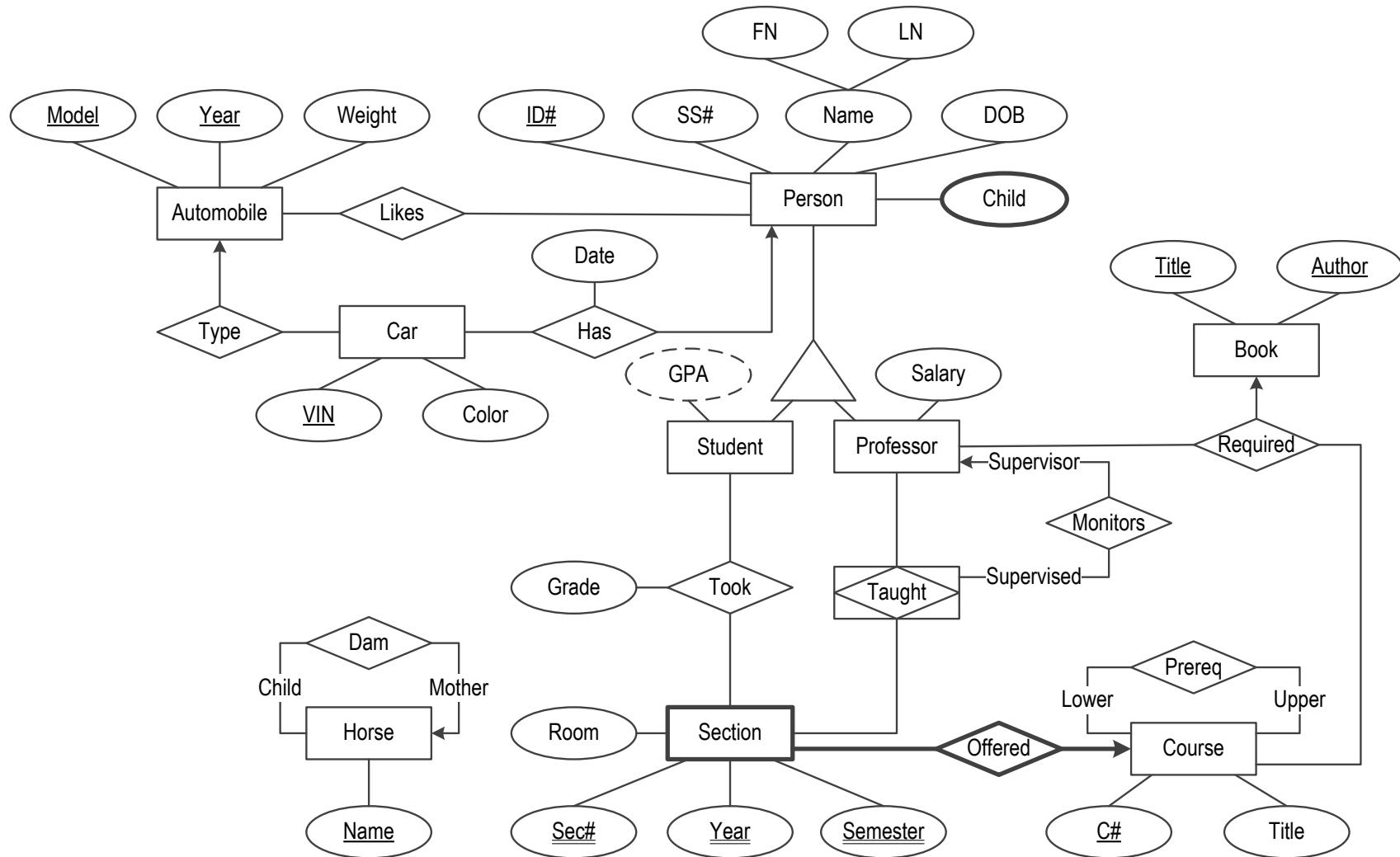
# Relational Database So Far



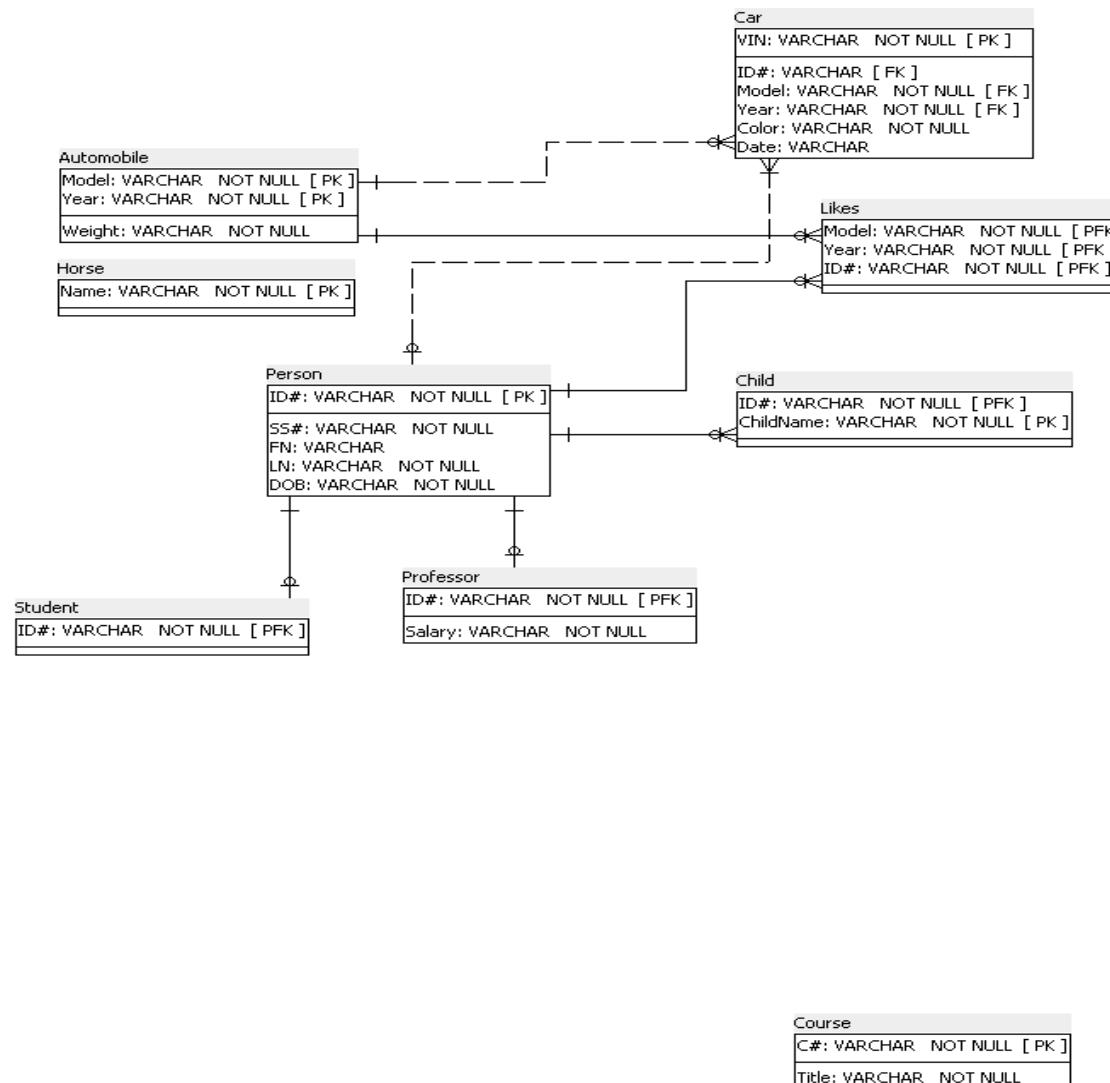
## Comment on ISA

- We did not have these in our example, but they are important and that's how should be phrased in annotations in your homework
- If ISA is total, we will say: The union of the set of ID#s in Student and of the ID#s in Professor is equal to the set of ID#s in Person
- If ISA is disjoint, we will say: The intersection of the set of ID#s in Student and of the ID#s in Professor is empty

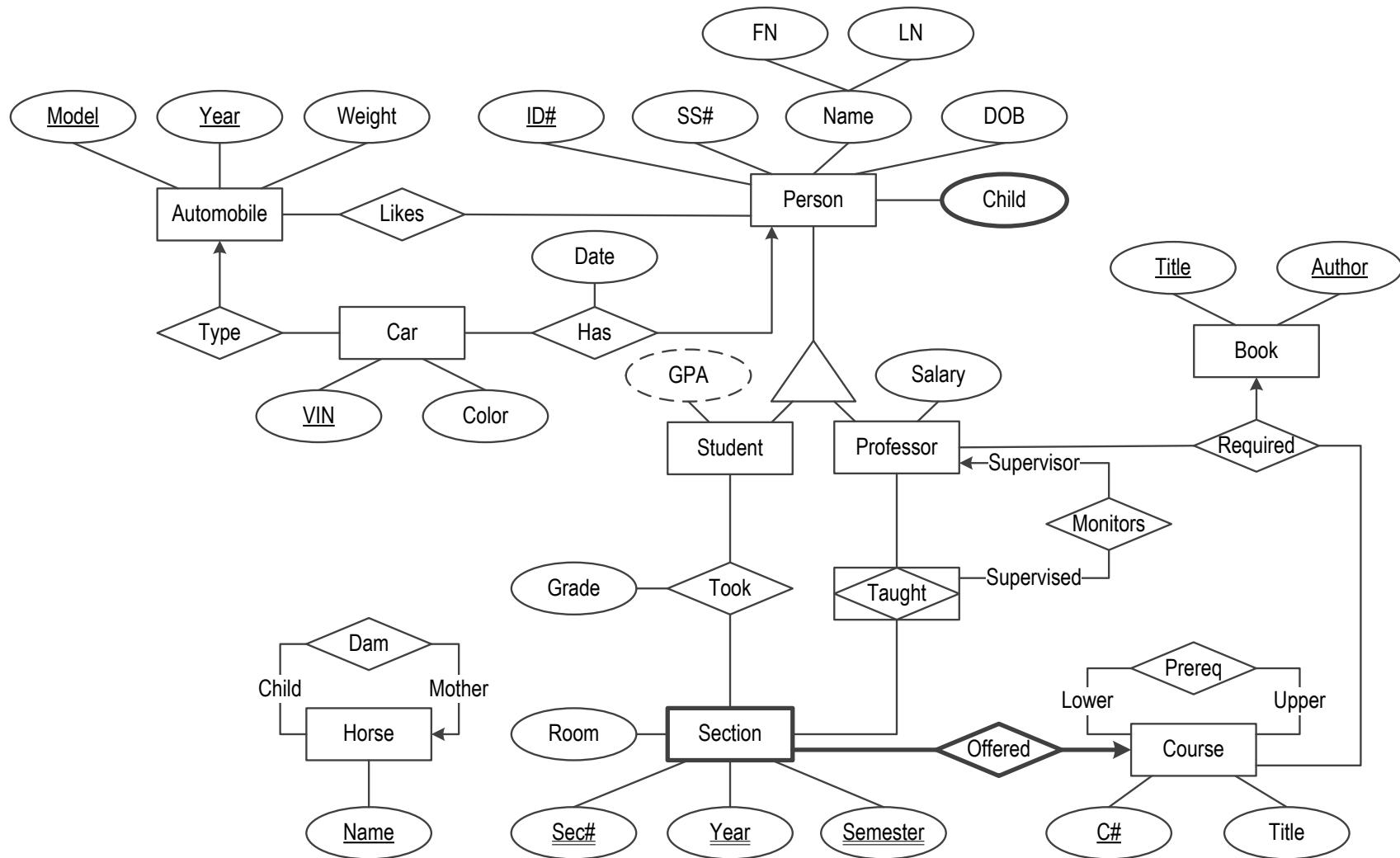
# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



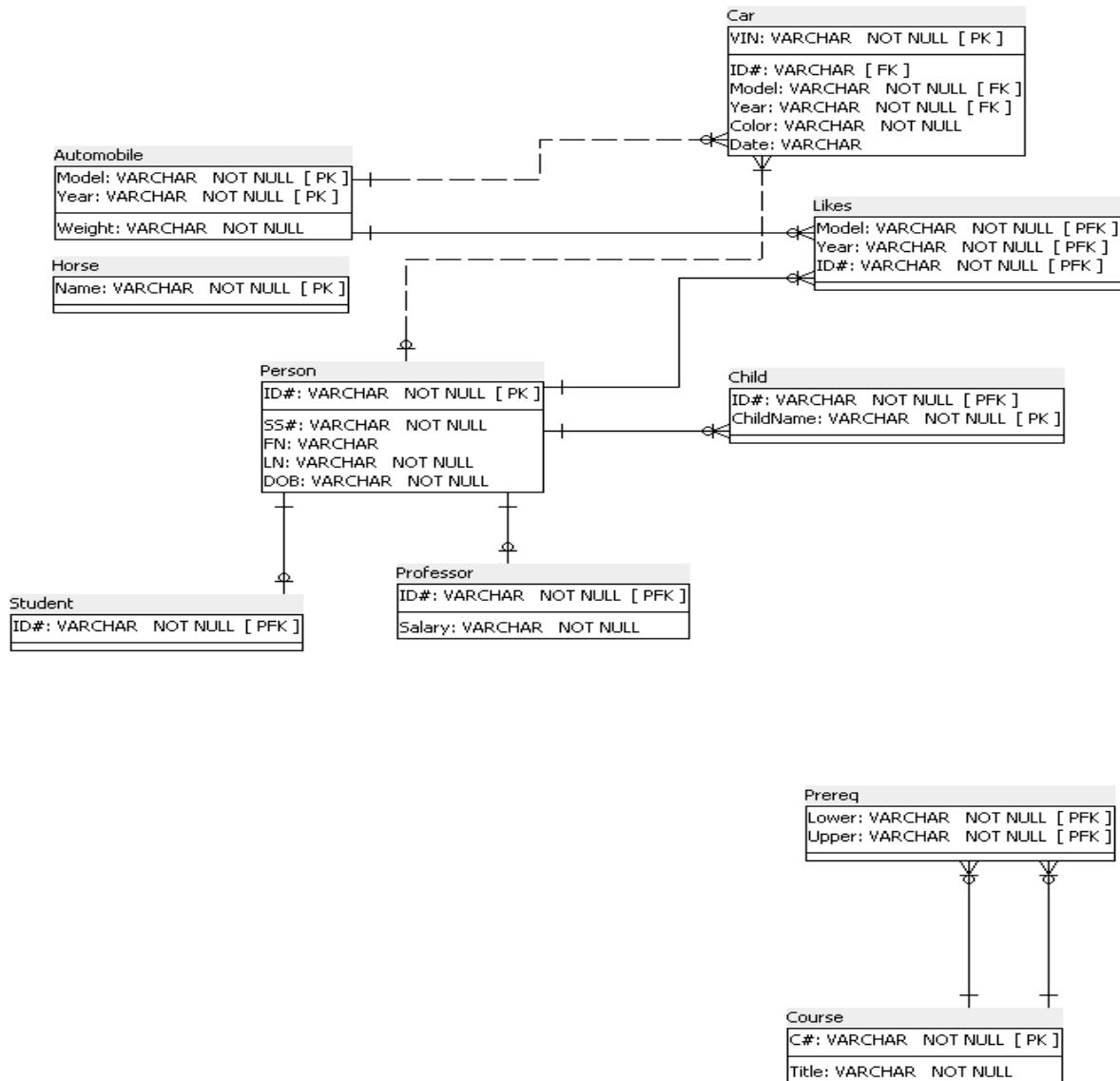
# Relational Database So Far



# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



# Relational Database So Far

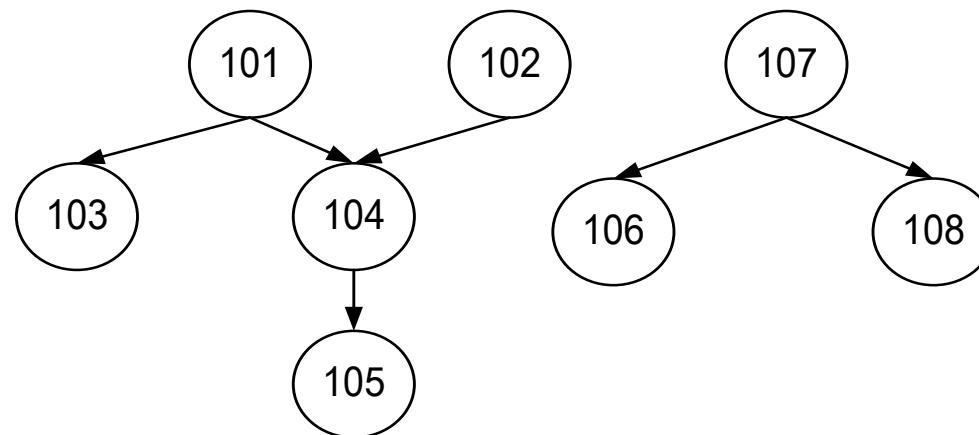


## Annotations

- Table Prereq stores/is a Directed Acyclic Graph (DAG)

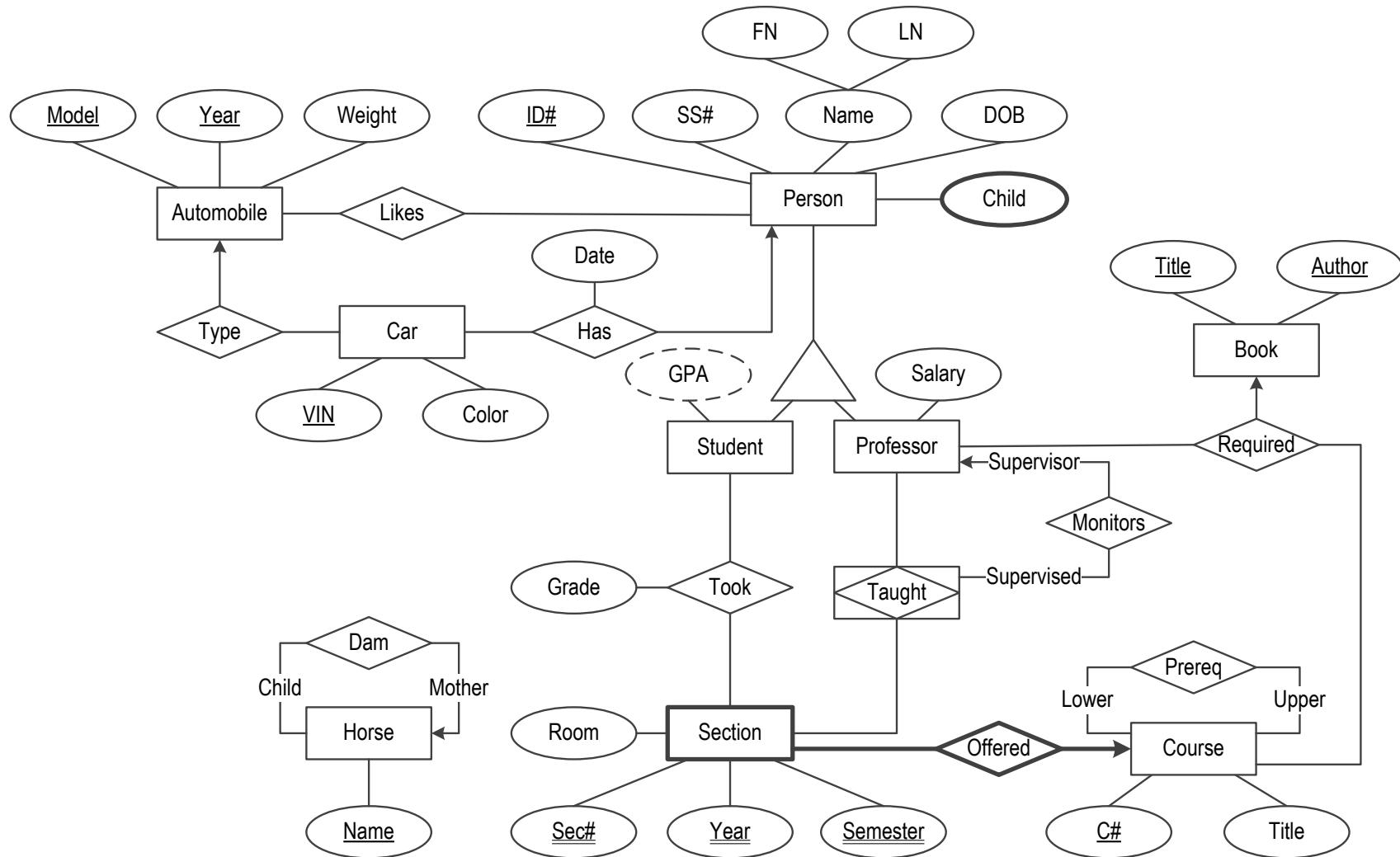
# Comments

- A binary relationship from a set to itself is also called recursive
- If we draw the DAG, the arcs will naturally point from Lower to Upper
- From roots/sources going down until they hit the leaves/sinks
- Example

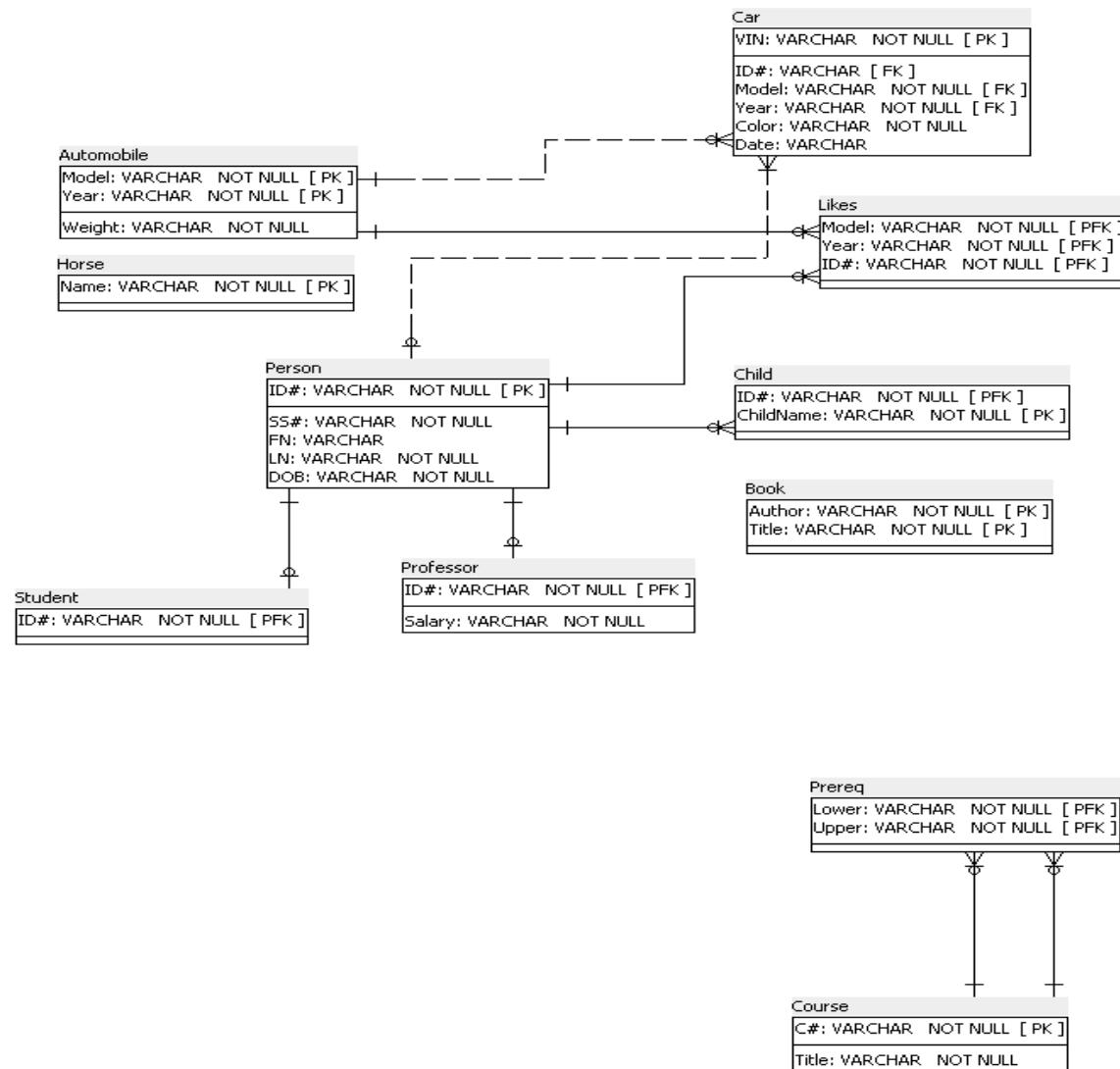


- But we could also draw in the opposite direction

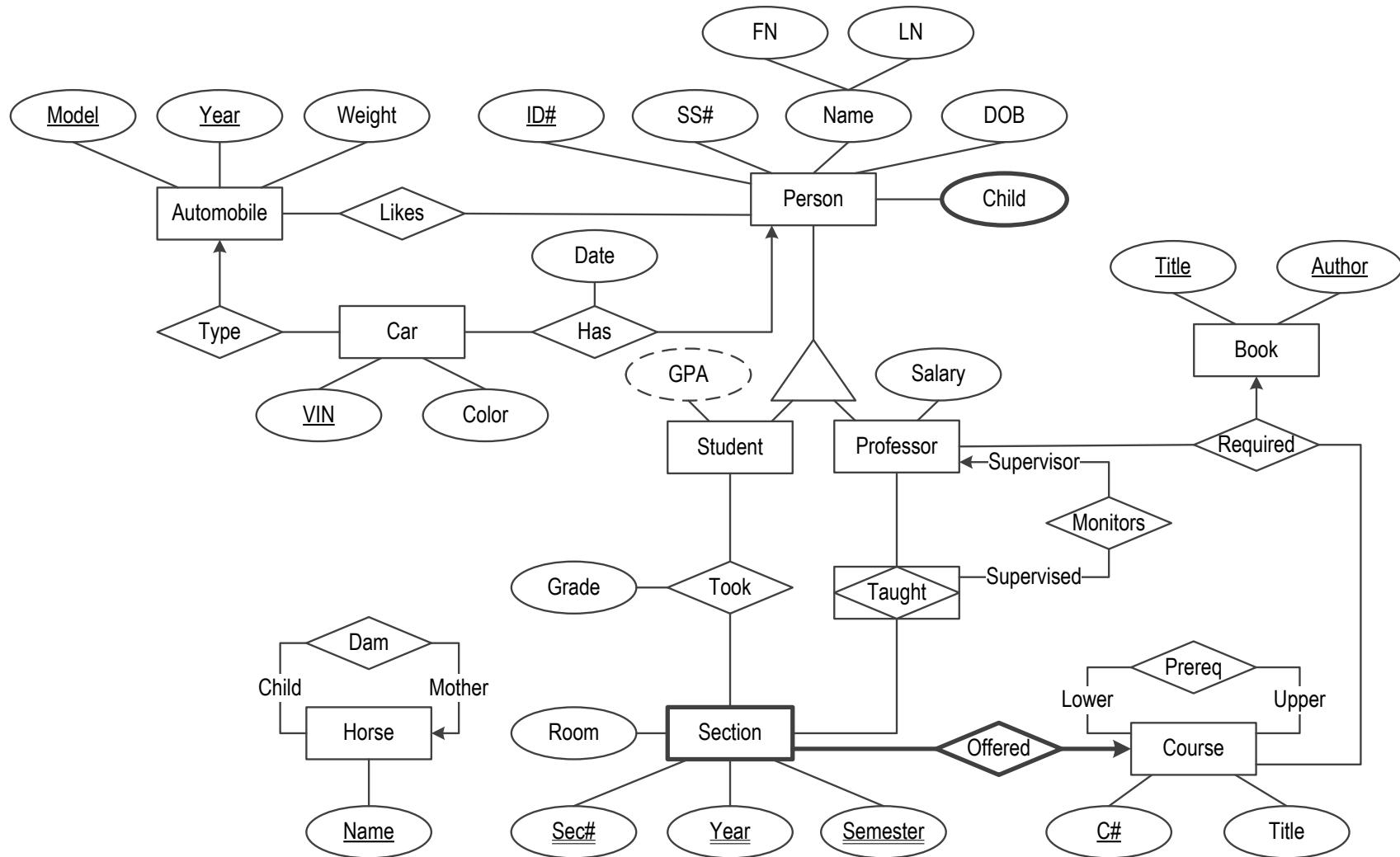
# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



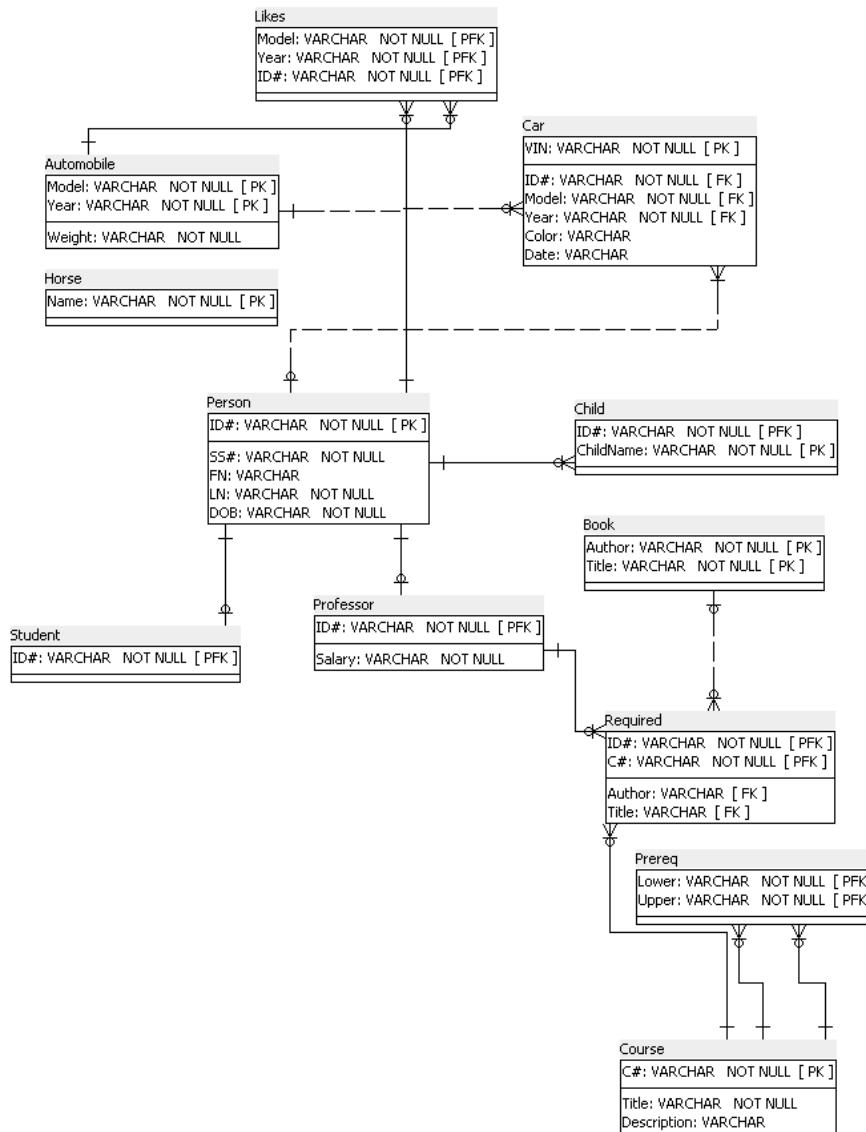
# Relational Database So Far



# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



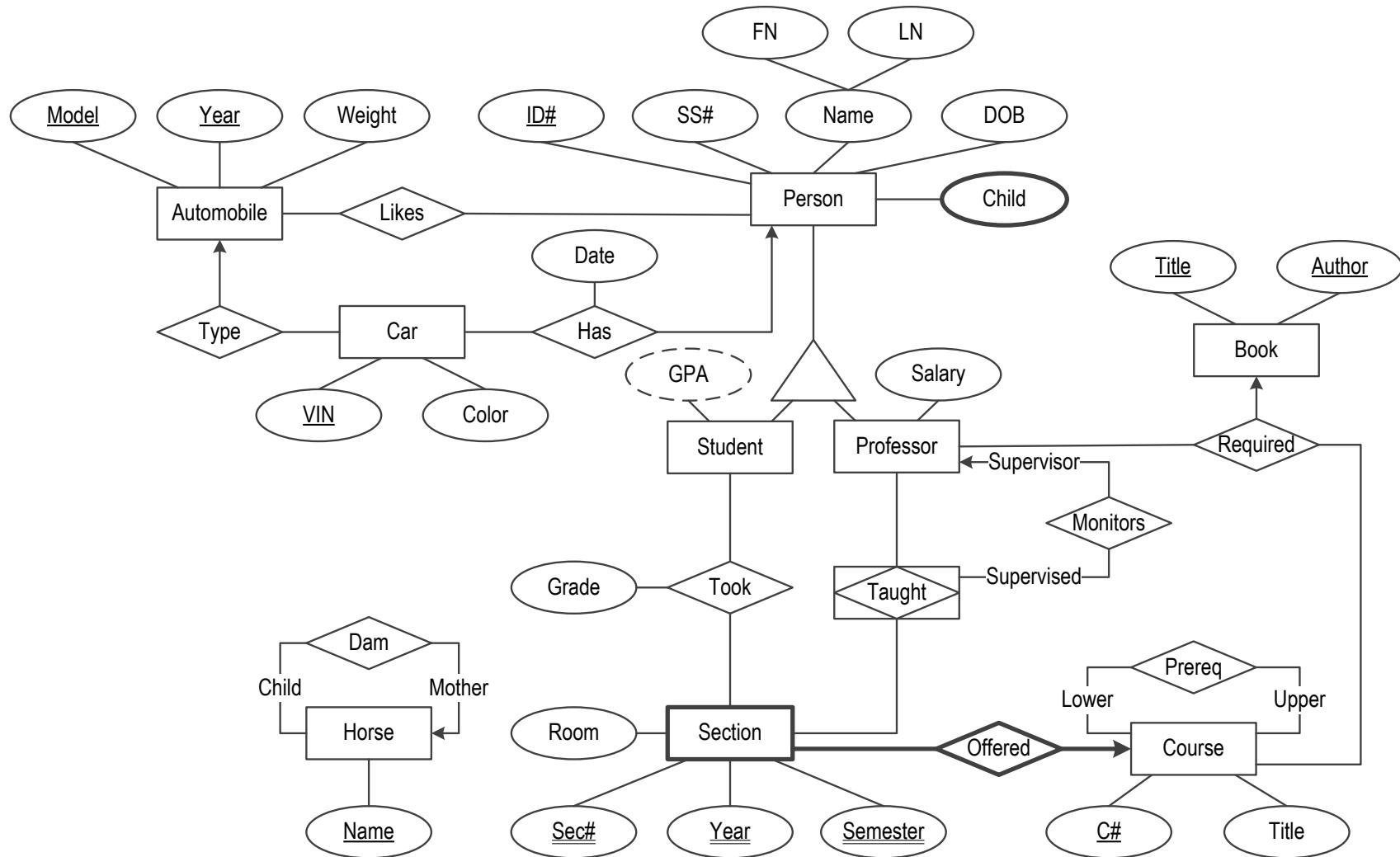
# Relational Database So Far



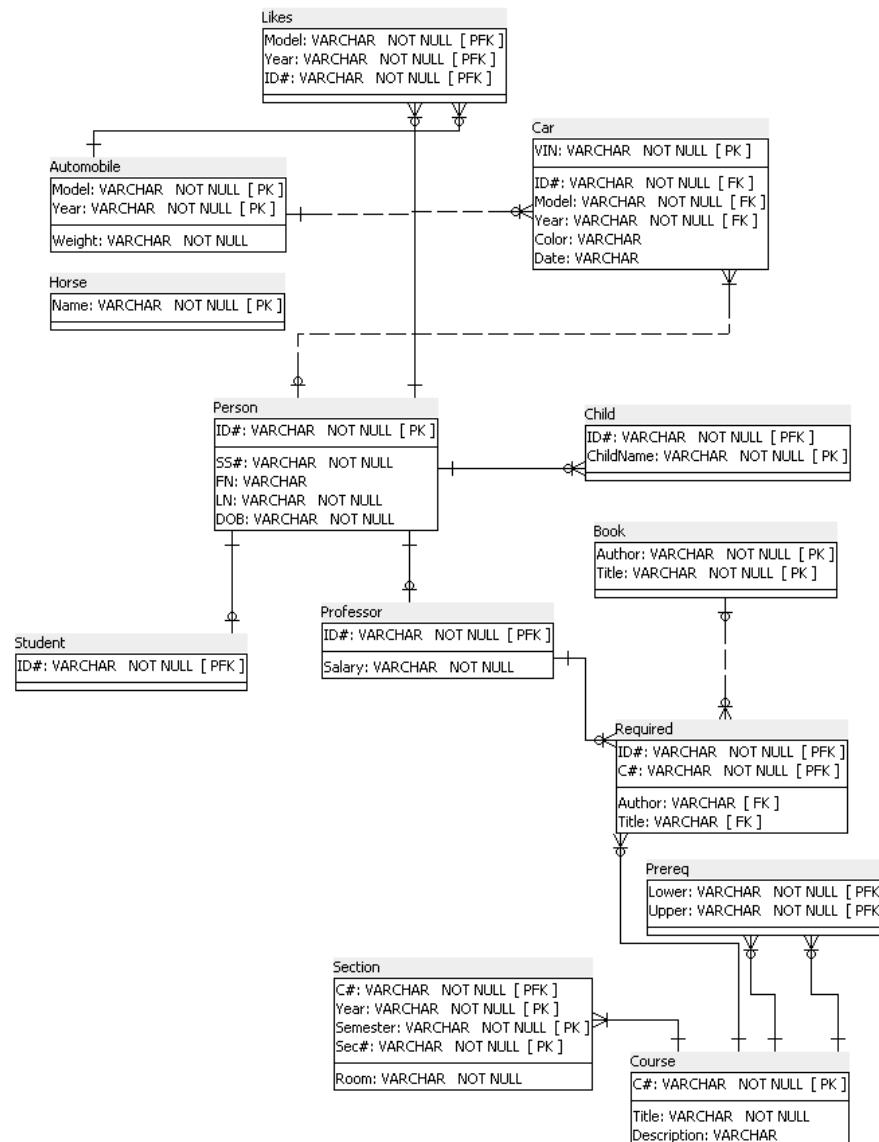
## Comments

- Note that the Foreign Key (Title, Author) in Required is not part of the Primary Key
- This shows that the ternary relationship stored in Required is somewhat restricted
- It is many-to-one from Course and Professor into Book
- It is a partial function of two variables
- We saw this before, so this is just a reminder

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



# Relational Database So Far



## Comments

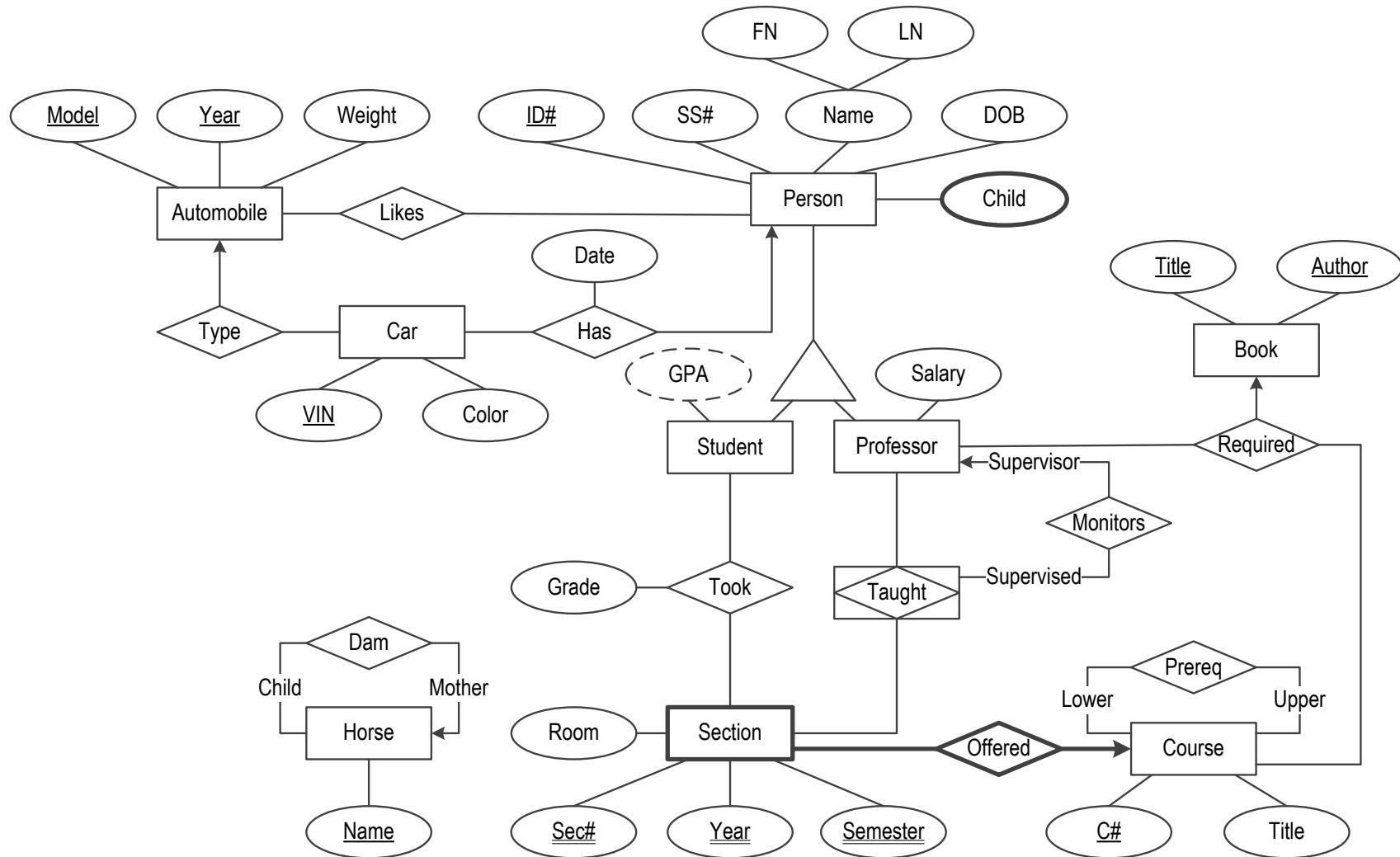
- One of our current sections (we have 3) is identified by the primary key: CSCI-GA.2433, 2021, Fall, 001
  - » Compare with what we had in the previous unit
- As Offered is binary many-to-one we do not need to have an “intermediate” table storing it
  - » In fact, it would not make any sense to have it here, because if we had it here, it would be just part of Section with one column repeated

A typical tuple would be primary keys from the Course table and from the Section table

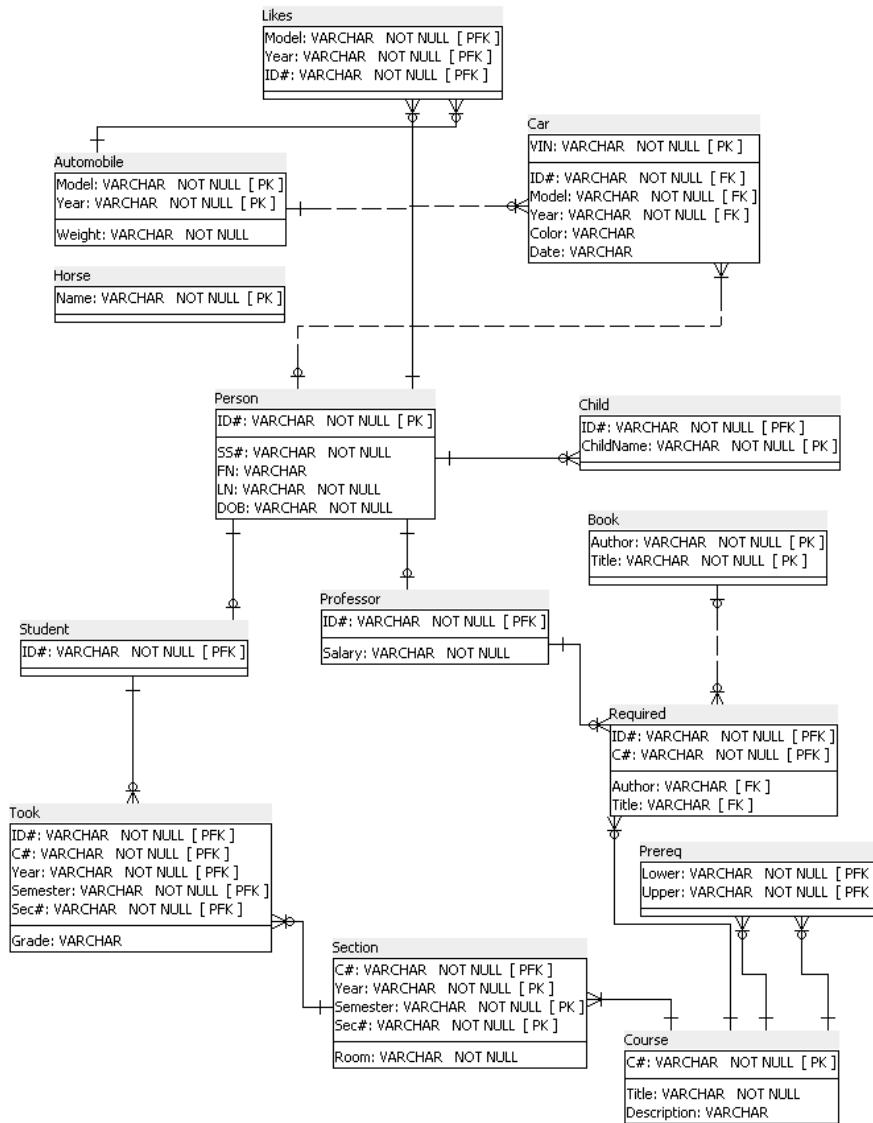
CSCI-GA.2433, CSCI-GA.2433, 2021, Fall, 001

- A more common case for a weak entity would be that the strong entity (here Course) would be related to 0, 1, or  $> 1$  weak entities (here Section)
- But in our application, we specifically said that each Course has at least 1 Section
- So, we have a special case: the strong entity is related to 1 or more than 1 weak entities

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



# Relational Database So Far

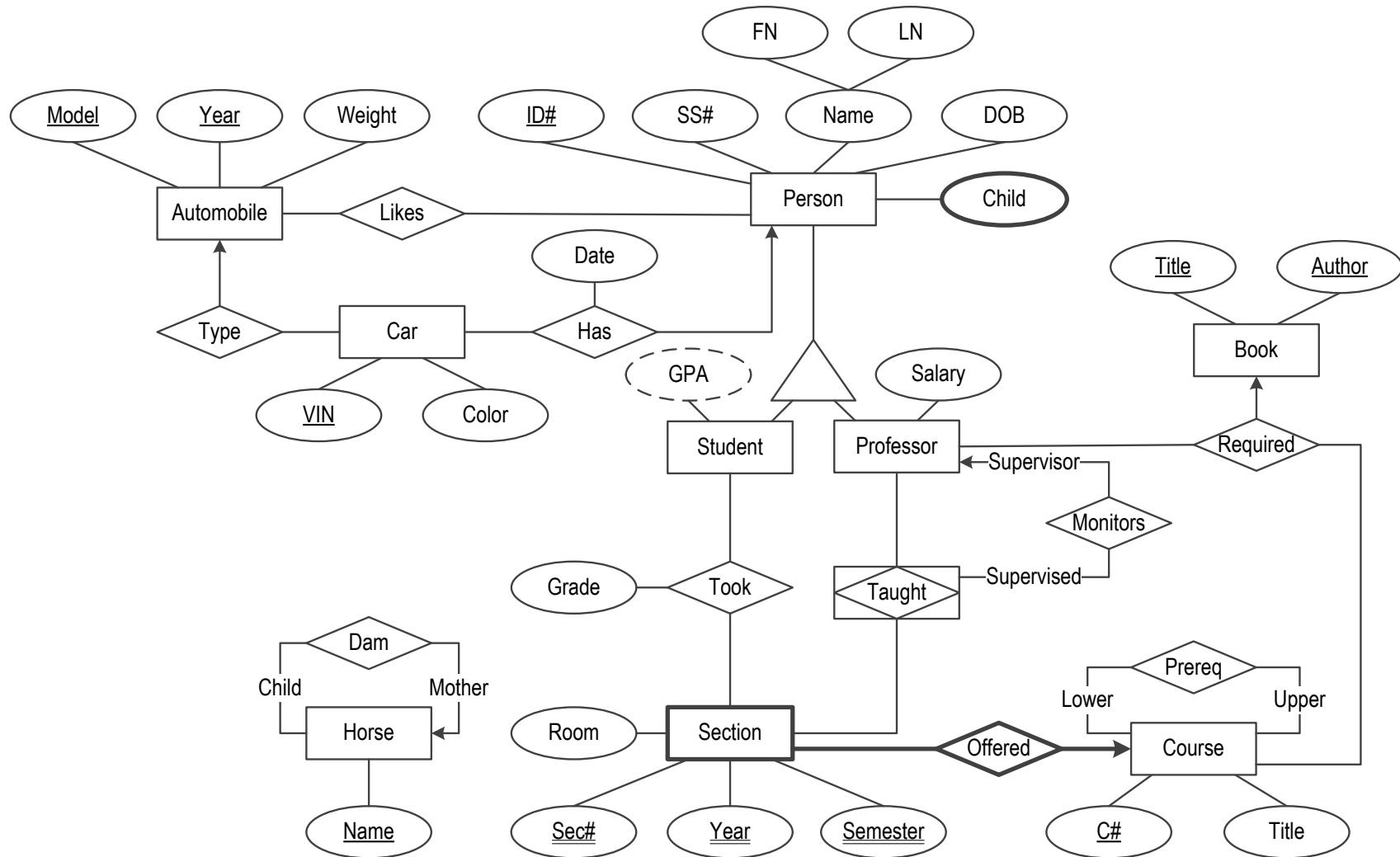


- GPA: Computed Attribute for Student by adding all the known numeric Grades, dividing by the number of Sections that the Student Took and in which it got a numeric Grade

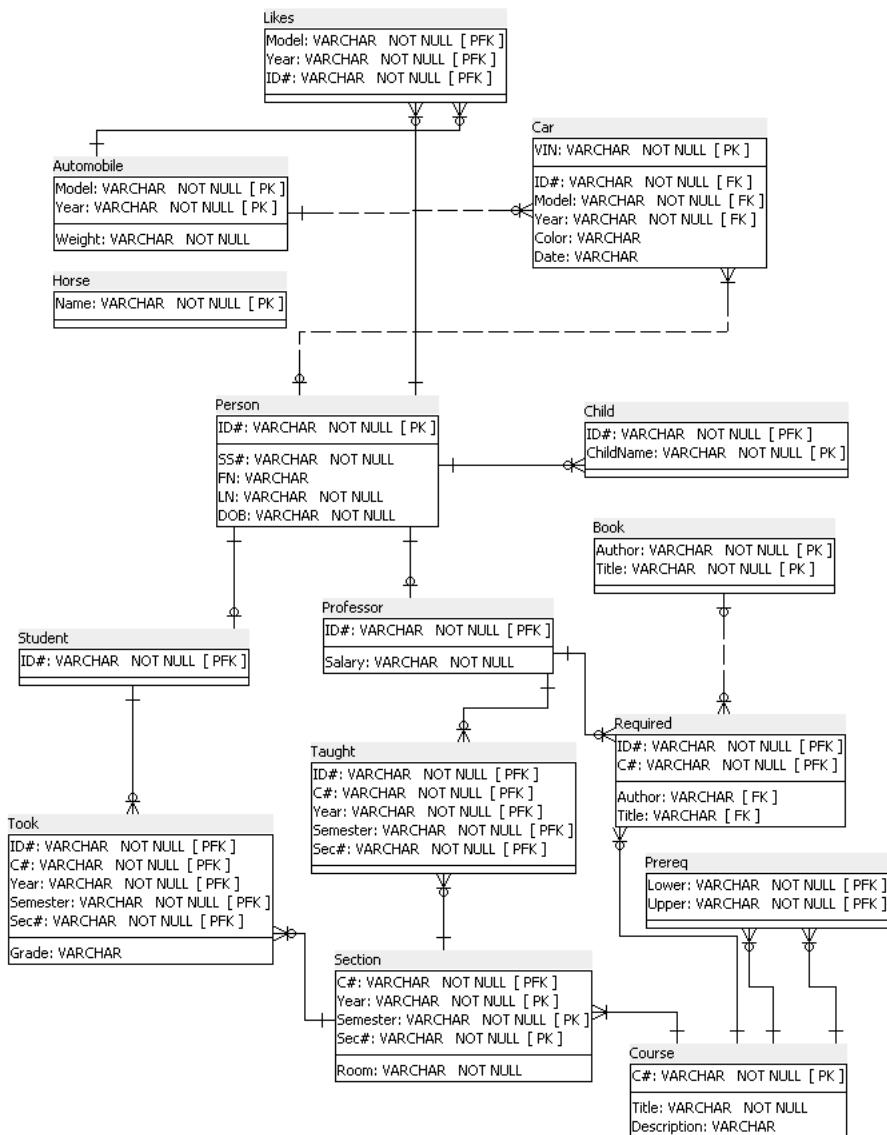
## Comments

- In some implementations a derived attribute is stored
- If this were the case, we would store GPA as an attribute in Student

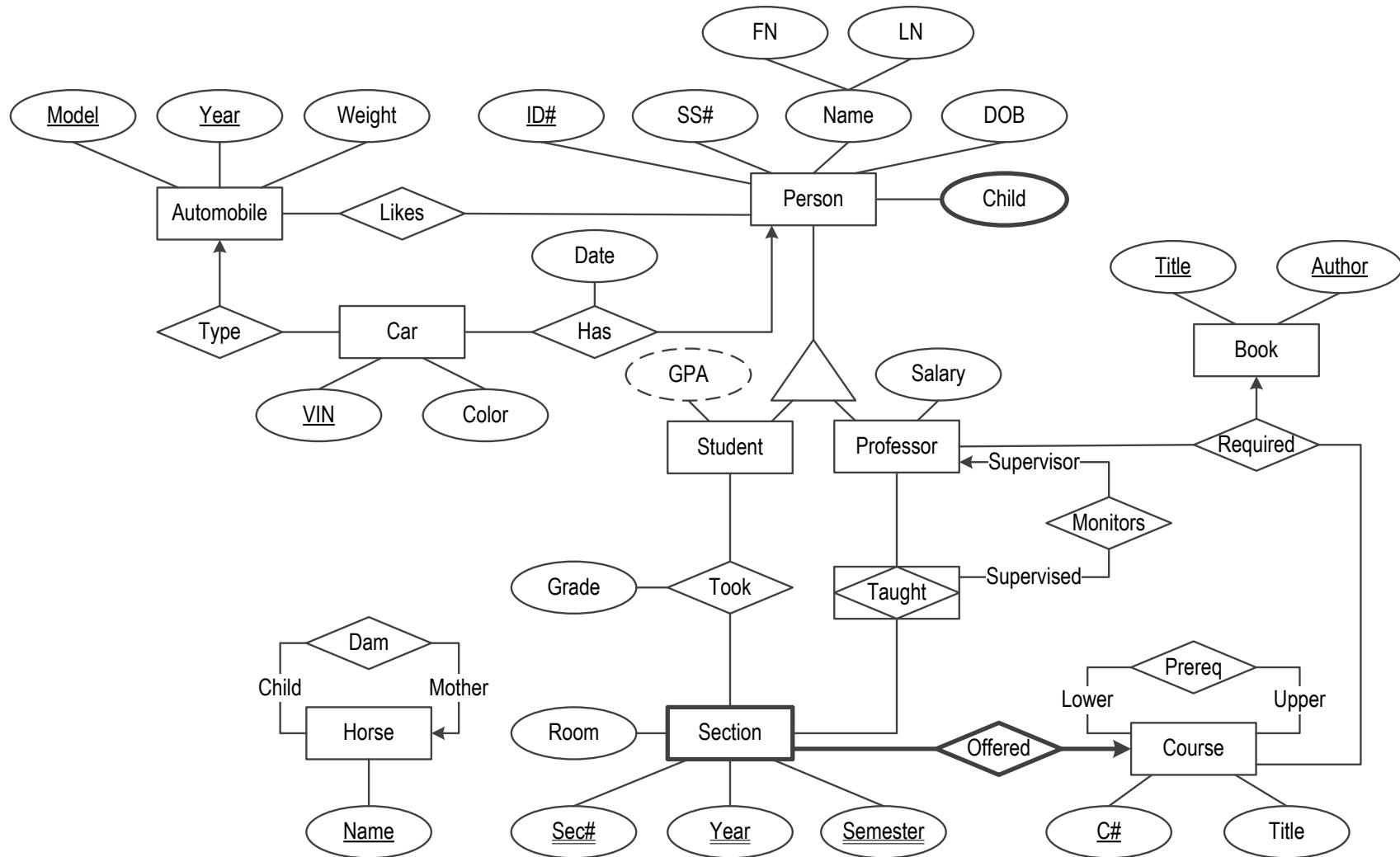
# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



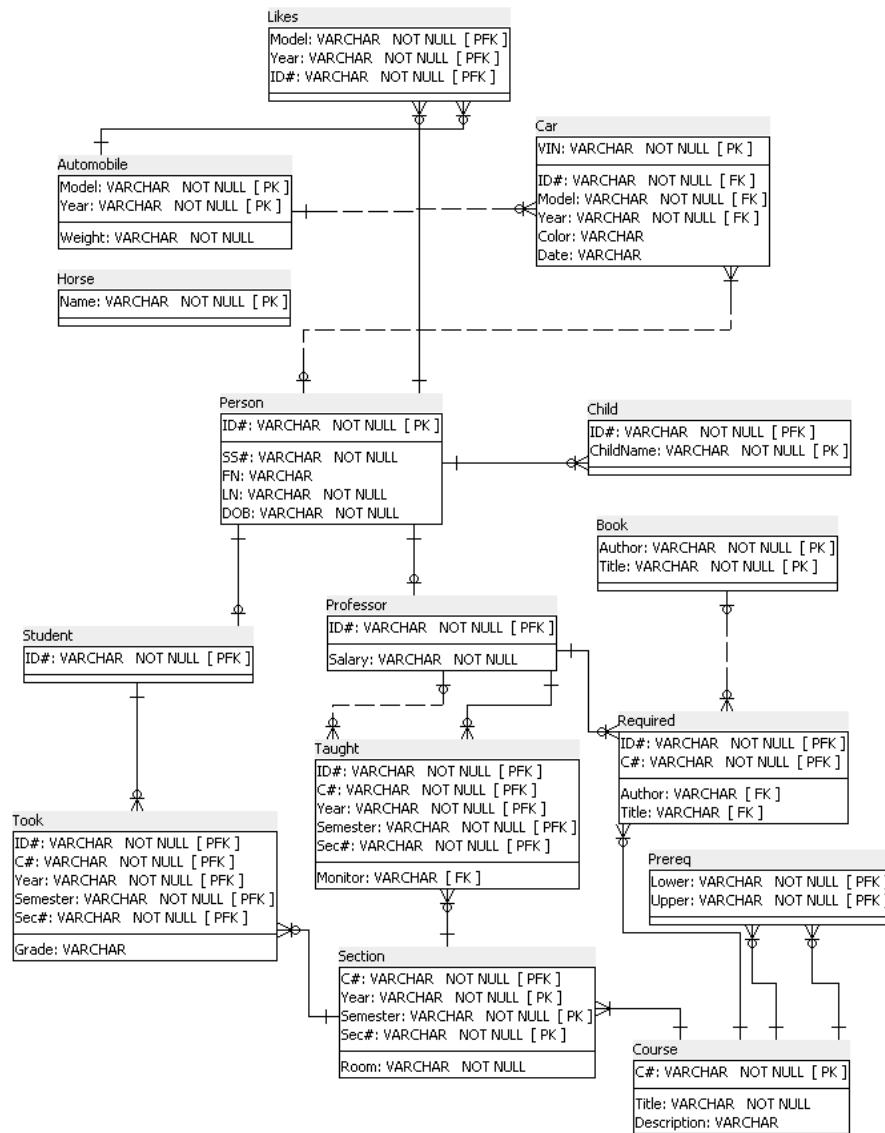
# Relational Database So Far



# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



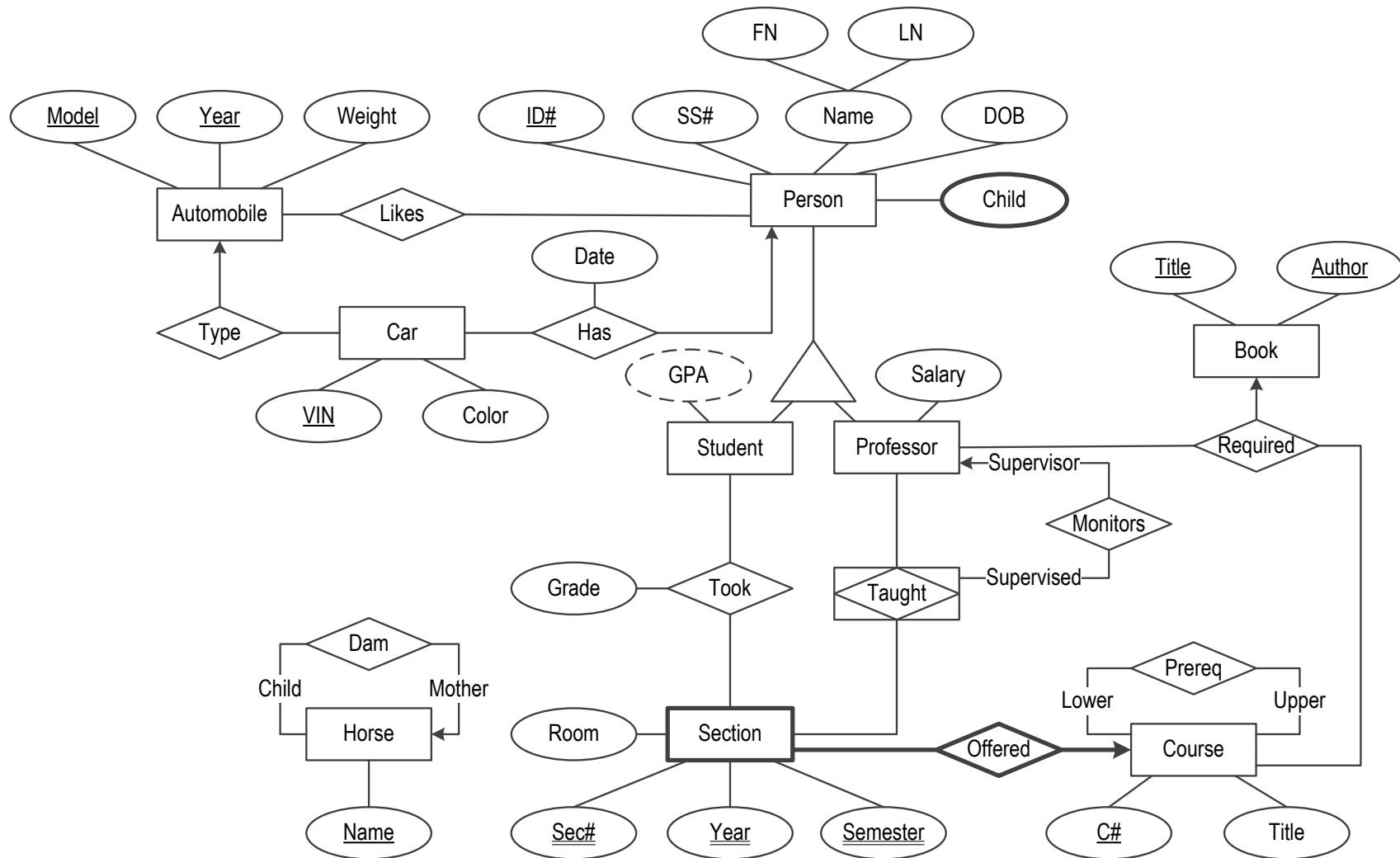
# Relational Database So Far



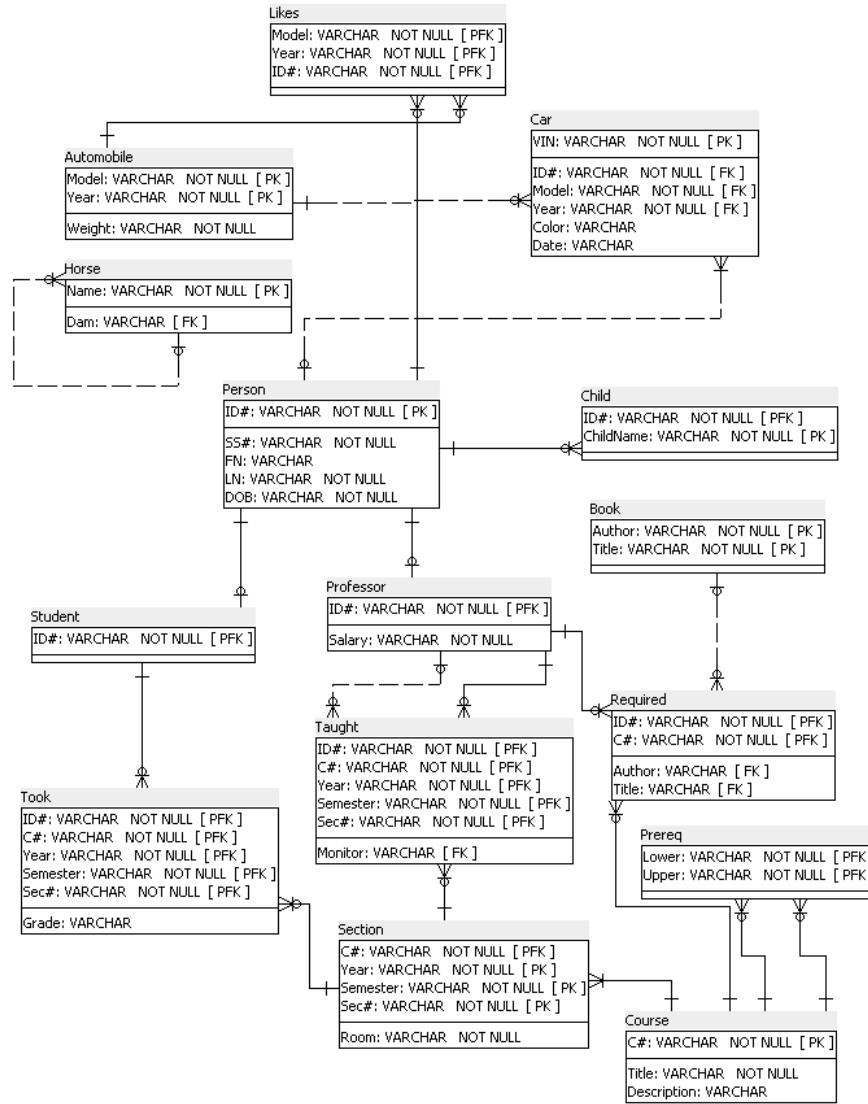
## Annotations

- In every tuple of table Taught, the values in columns ID# and Monitor must be different (if there is a Monitor)

# Reminder: Our ER Diagram So We Do Not Have To Go Back To Look At It



# Relational Database So Far

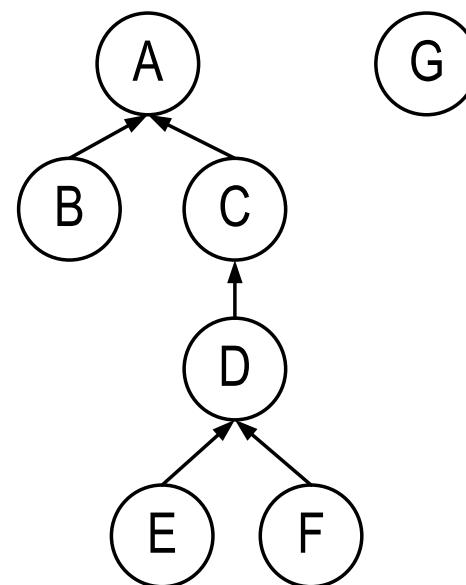


## Annotations

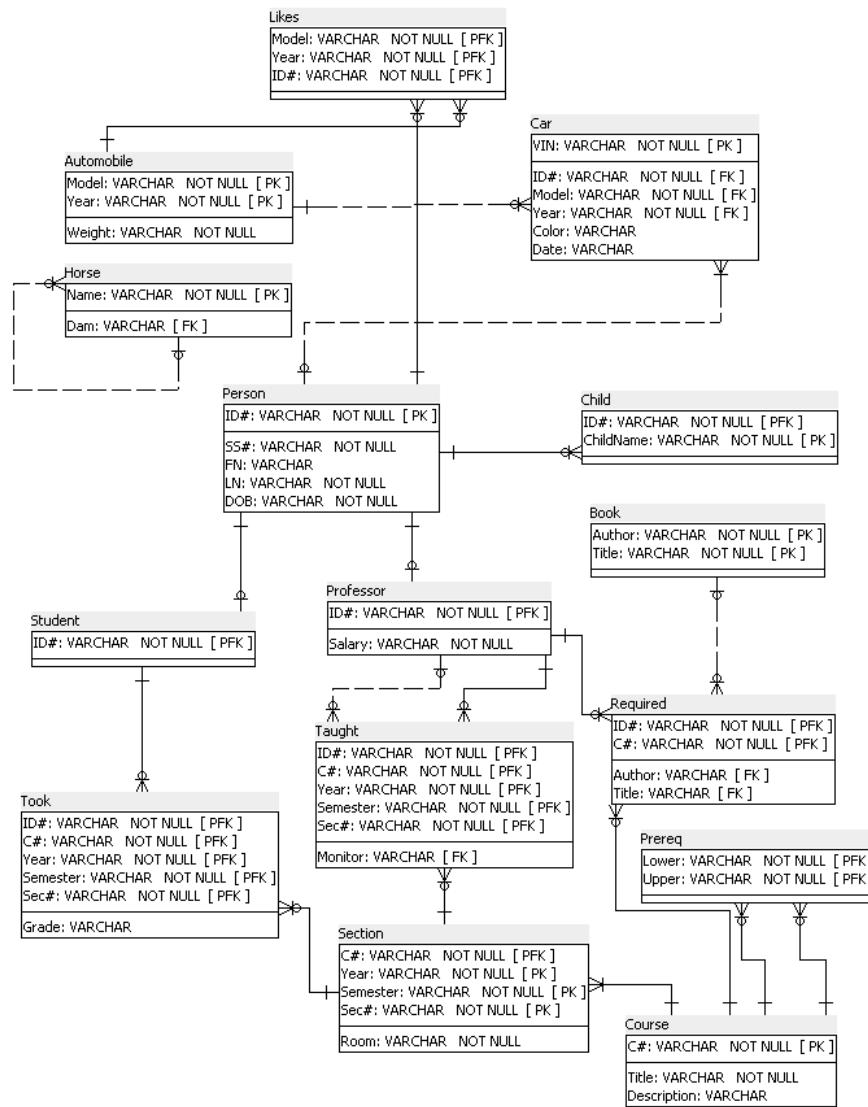
- Table Dam stores/is a forest of rooted trees

## Comments

- This is a binary many-to-one relationship from a set to itself
- If we drew the forest of rooted trees as we implement it here, the arcs would point from Child to Mother
  - » From leaves/sources going up until they hit the root/sink
  - » Because we have a function (from child to parent)
- Example



# We Are Done



# Annotations for the Relational Implementation (Abbreviated: Do Not Abbreviate in Homework)

- SS# is UNIQUE
- Every ID# must appear in at least two distinct tuples of Car
- Prereq is a Directed Acyclic Graph (DAG)
- GPA: Computed Attribute for Student by adding all the known numeric Grades, dividing by the number of Sections that the Student Took and in which it got a numeric Grade
- In every tuple of Taught, ID# and Monitor must be different (if there is a Monitor)
- Dam is a forest of rooted trees

## Comments

- Good design: few annotations



# Using Visio

- Visio can be used to designing/specifying relational databases
- You can look at a tutorial, to get familiar with the mechanics of Visio
- This is greatly oversimplified, but a good start
  - » <http://www.youtube.com/watch?v=1BYt3wmkgXE> but foreign keys are not explained
  - » <http://www.youtube.com/watch?v=r0x8ZMyPoj4&NR=1> but this third part
    - Is misleading in the context of relational databases, due to the handling of many-to-many relationships and
    - The use of the second page, all the pages in a single Visio drawing refer to a **single** ER diagram, so each ER diagram needs its own Visio drawing/file



# Specifying A Relational Implementation

- It is possible to use Visio or ErWin to specify our relational implementation
  - » Visio has an “enterprise” version to generate database specifications from the diagram to SQL DDL
- We will just focus on the first task
- The second can be done automatically so we do not need to look at it here



# Specifying A Relational Implementation (more on Visio)

- A drawing in Visio is not an Entity Relationship Diagram tool despite such terminology in Visio
  - » In fact, this is good, as ***it produces a relational schema***, which is what we actually need, but this is a lower-level construct than ER diagrams
- ***It focuses on tables and the implicit many-to-one binary relationships induced by foreign key constraints***
- ***Table***
  - » A rectangle with three vertical subrectangles: name, list of attributes in the primary key, list of attributes not in the primary key
  - » Required attributes are in bold
  - » Attributes in the primary key and foreign keys are labeled as such
- ***Relationship***
  - » A many-to-one binary (or perhaps one-to-one, which is a special case) relationship induced by a foreign key constraint is explicitly drawn by means of a segment with an arrow head  
We will have alternative notations later

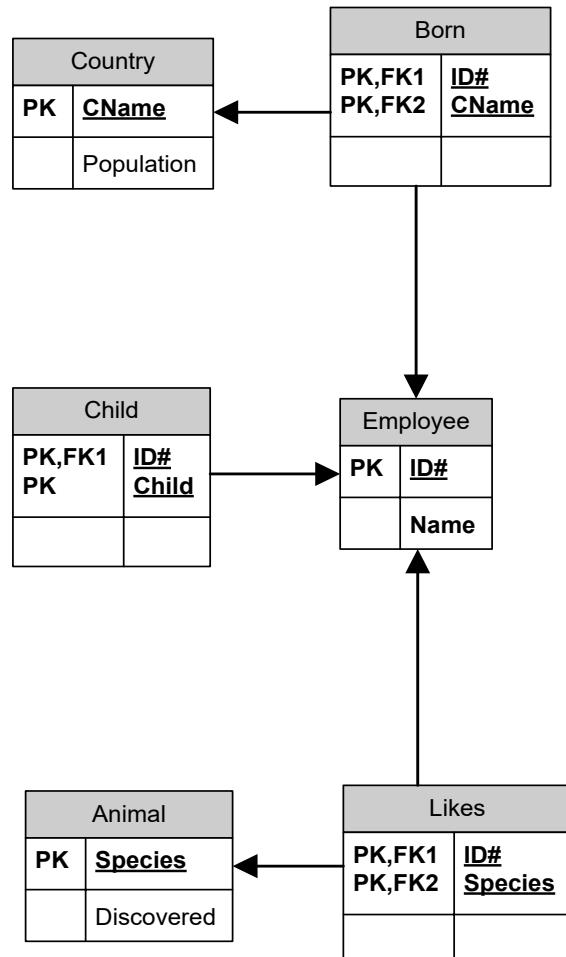


# Relational Implementation For The Example

Country	<u>Cname</u>	Population	n	ID#	CName
	US			1	US
	IN	1150		2	IN
	CN	1330		5	IN
	RU			6	CN

Child	<u>ID#</u>	<u>Child</u>	Employee	<u>ID#</u>	Name
	1	Erica		1	Alice
	1	Frank		2	Bob
	2	Bob		4	Carol
	2	Frank		5	David
	6	Frank		6	Bob

Animal	<u>Species</u>	Discovered	Likes	<u>ID#</u>	<u>Species</u>
	Horse	Asia		1	Horse
	Wolf	Asia		1	Cat
	Cat	Africa		2	Cat
	Yak	Asia		6	Yak
	Zebra	Africa			



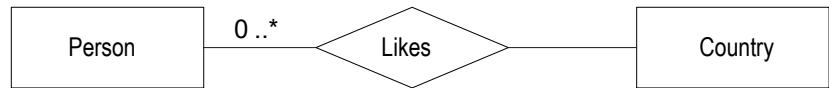


# Cardinality Constraints

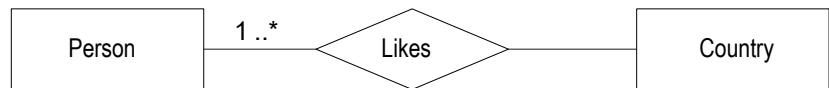
- The statement that a relationship is many-to-one as opposed to be a “standard” many-to-many relationship is really a cardinality constraint
- We will look at a relationships Likes between Person and Country and four cases of cardinality constraints on how many Countries a Person may like
  - » No constraint
  - » At least one
  - » At most one
  - » Exactly one
- For the first two, Likes is many-to-many
- For the last two, Likes is many-to-one
- Intuitively, Likes is many to one if for every Person, when you see which Countries this Person Likes, you get 0 or 1
- If you always get 1, this is a total function, otherwise this is a partial function



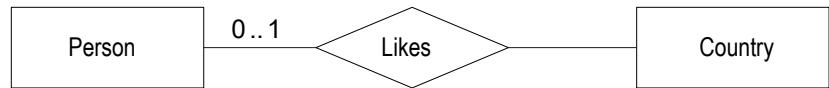
# Specifying These Constraints (Revisited)



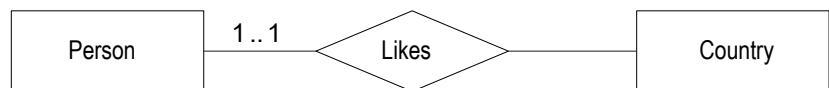
Every Person likes 0 or more Countries



Every Person likes 1 or more Countries



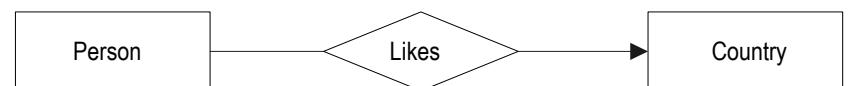
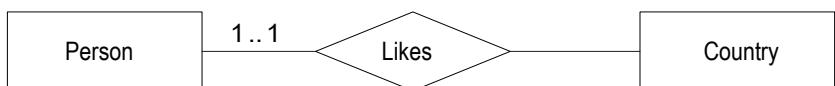
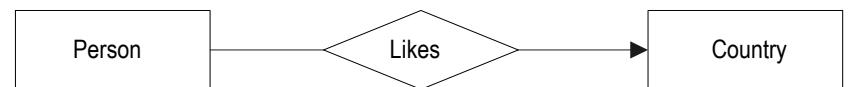
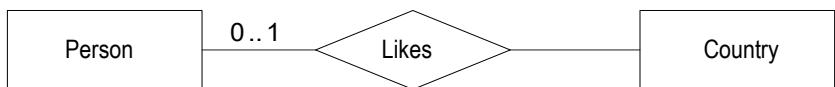
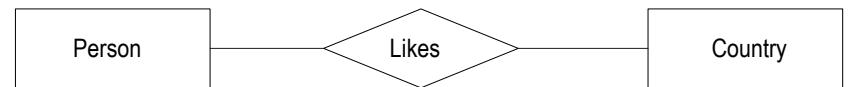
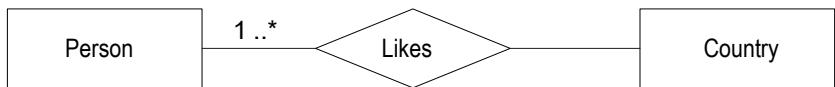
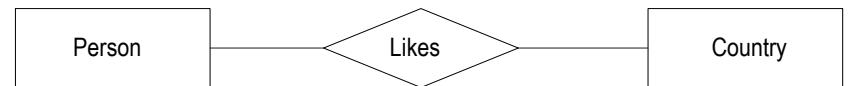
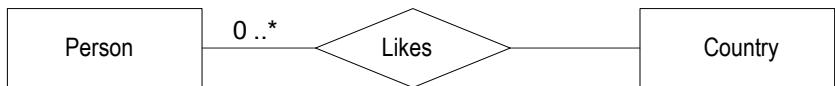
Every Person likes 0 or 1 Countries



Every Person likes 1 Country

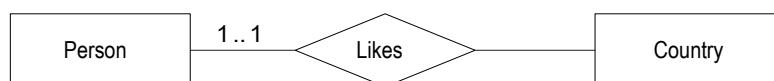
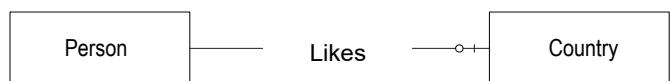
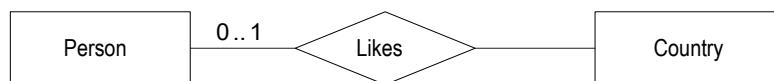
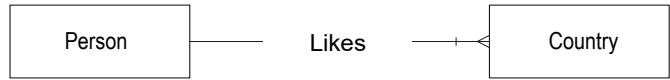
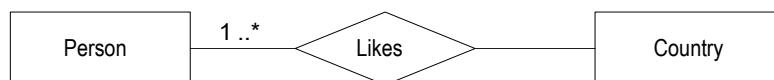
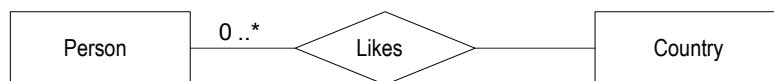


# Arrow Notation Cannot Distinguish Some Cases





- ***Note: different sides of the relationship are labeled in the two notations!***





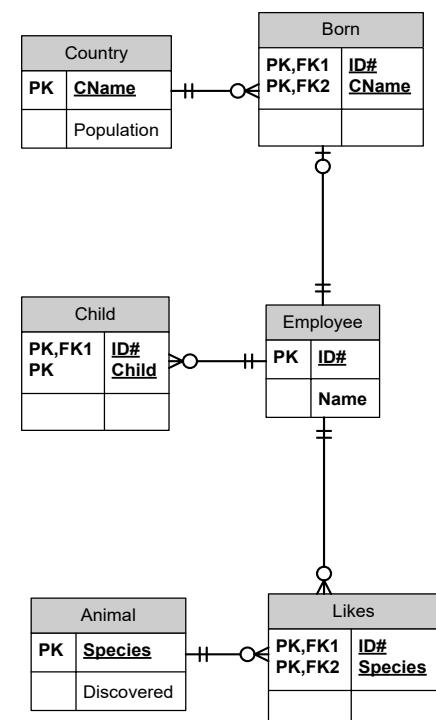
- In general, cardinalities of both sides of the relationship may need to be specified
- We did only one, because it is sufficient to understand the notation
- We now return to the relational implementation of our example
- Visio and ErWin can use the Crow's Feet notation



# Relational Implementation For The Example

Country	<u>Cname</u>	Population	Born	ID#	CName
	US			1	US
	IN	1150		2	IN
	CN	1330		5	IN
	RU			6	CN

Child	<u>ID#</u>	<u>Child</u>	Employee	<u>ID#</u>	Name
	1	Erica		1	Alice
	1	Frank		2	Bob
	2	Bob		4	Carol
	2	Frank		5	David
	6	Frank		6	Bob



Animal	<u>Species</u>	Discovered	Likes	<u>ID#</u>	<u>Species</u>
	Horse	Asia		1	Horse
	Wolf	Asia		1	Cat
	Cat	Africa		2	Cat
	Yak	Asia		6	Yak
	Zebra	Africa			
	Zebra	Africa			



## Ends Of Lines (1/2)

- For every Country:  $0..*$  of Born
  - » 0 or more Employees were born there
- For every Born:  $1..1$  of Country
  - » Because each row in Born has exactly 1 value of Country
- For every Employee:  $0..1$  of Born
  - » Because an Employee was born in at most 1 Country
- For every Born:  $1..1$  of Employee
  - » Because each row in Born has exactly 1 value of Employee
- For every Employee:  $0..*$  Child
  - » 0 or more of Child for an Employee
- For every Child:  $1..1$  Employee
  - » Because every Child is assigned to exactly one Employee



## Ends Of Lines (2/2)

- For every Employee: 0..\* of Likes
  - » Employee can Like 0 or more Species
- For every Likes: 1..1 of Employee
  - » Because each row in Likes has exactly 1 value of Employee
- For every Animal: 0..1 of Likes
  - » Because a Species can be Liked by 0 or more Employees
- For every Likes: 1..1 of Species
  - » Because each row in Likes has exactly 1 value of Species



# Born Versus Likes (1/2)

<u>Country</u>	<u>Cname</u>	<u>Population</u>
	US	
	IN	1150
	CN	1330
	RU	

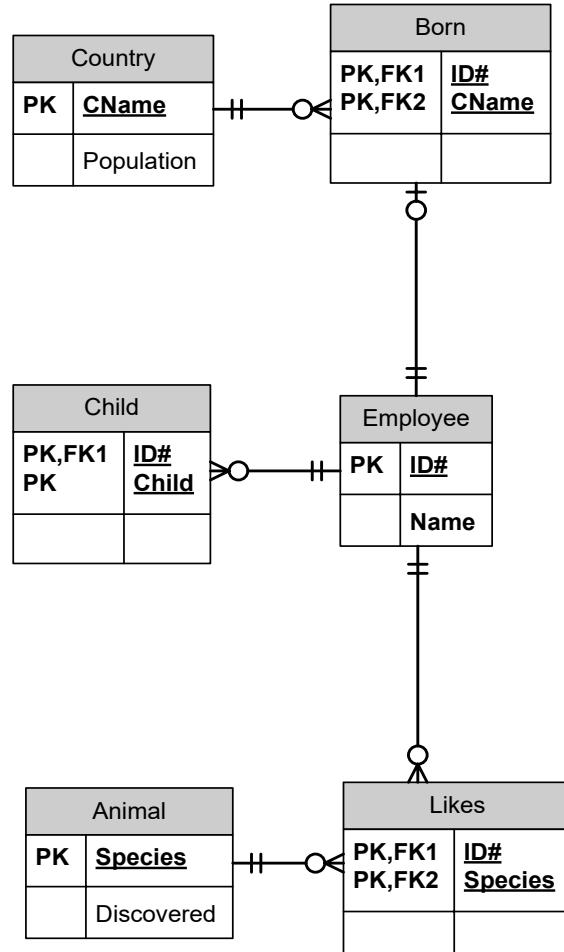
<u>Born</u>	<u>ID#</u>	<u>CName</u>
	1	US
	2	IN
	5	IN
	6	CN

<u>Child</u>	<u>ID#</u>	<u>Child</u>
	1	Erica
	1	Frank
	2	Bob
	2	Frank
	6	Frank

<u>Employee</u>	<u>ID#</u>	<u>Name</u>
	1	Alice
	2	Bob
	4	Carol
	5	David
	6	Bob

<u>Animal</u>	<u>Species</u>	<u>Discovered</u>
	Horse	Asia
	Wolf	Asia
	Cat	Africa
	Yak	Asia
	Zebra	Africa

<u>Likes</u>	<u>ID#</u>	<u>Species</u>
	1	Horse
	1	Cat
	2	Cat
	6	Yak





## Born Versus Likes (2/2)

- Note that the many-to-one relationships are not of the same type in both cases
- The relationship between Likes and Employee indicates that when you start from a row of Employee you end up in between 0 and unbounded number of rows of Likes: no restriction  
An employee can like any number of animals
- The relationship between Born and Employee indicates that when you start from a row of Employee you end up in between 0 and 1 rows of Born  
An employee can be born in at most one country and therefore from a row of Employee you end up in between 0 and 1 rows of Born: a restriction

***Born is really a (partial) one-to-one relationship***

Such relationships are considered “strange”



## Treating Born Differently From Likes

- The above discussion implies that for every row in Employee there is at most one “relevant” row of Born
- Therefore, the “extra” information about an employee that is currently stored in Born can be added to Employee
- Born can be removed from the design
- This sounds very formal, but intuitively very clear as we can see from an alternative design



# Alternative For Born

## ■ Replace

Employee	ID#	Name
1		Alice
2		Bob
4		Carol
5		David
6		Bob

Born	ID#	Cname
	1	US
	2	IN
	5	IN
	6	CN

by

Employee	ID#	Name	Cname
1		Alice	US
2		Bob	IN
4		Carol	
5		David	IN
6		Bob	CN

# Alternative Relational Implementation For The Example

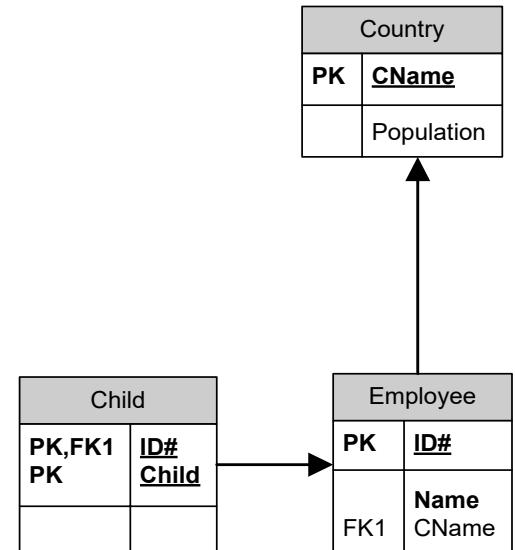


<u>Country</u>	<u>CName</u>	Population
	US	
	IN	1150
	CN	1330
	RU	

Country	
PK	CName
	Population

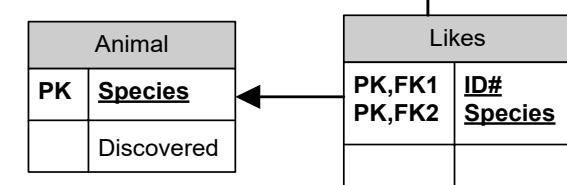
<u>Child</u>	<u>ID#</u>	<u>Child</u>
1		Erica
1		Frank
2		Bob
2		Frank
6		Frank

<u>Employee</u>	<u>ID#</u>	Name	CName
	1	Alice	US
	2	Bob	IN
	4	Carol	
	5	David	IN
	6	Bob	CN



<u>Animal</u>	<u>Species</u>	Discovered
Horse		Asia
Wolf		Asia
Cat		Africa
Yak		Asia
Zebra		Africa

<u>Likes</u>	<u>ID#</u>	<u>Species</u>
	1	Horse
	1	Cat
	2	Cat
	6	Yak





# Alternative Relational Implementation For The Example

Country	CName	Population
	US	
	IN	1150
	CN	1330
	RU	

Country	
PK	CName
	Population



Child	ID#	Child	Employee	ID#	Name	CName
	1	Erica		1	Alice	US
	1	Frank		2	Bob	IN
	2	Bob		4	Carol	
	2	Frank		5	David	IN
	6	Frank		6	Bob	CN

Child	
PK,FK1 PK	ID# Child

Employee	
PK	ID#
	Name CName



Animal	Species	Discovered
	Horse	Asia
	Wolf	Asia
	Cat	Africa
	Yak	Asia
	Zebra	Africa

Likes	ID#	Species
	1	Horse
	1	Cat
	2	Cat
	6	Yak

Animal	
PK	Species
	Discovered

Likes	
PK,FK1 PK,FK2	ID# Species





## Pattern Of Lines (1/2)

- The line between Animal and Likes is ***solid*** because the primary key of the “many side”, Likes, includes the primary key of the “one side”, Animal, so it “cannot exist” without it
- The line between Employee and Likes is ***solid*** because the primary key of the “many side”, Likes, includes the primary key of the “one side”, Employee, so it “cannot exist” without it
- The line between Employee and Child is ***solid*** because the primary key of the “many side”, Child, includes the primary key of the “one side”, Employee, so it “cannot exist” without it
- The line between Country and Employee is ***dashed*** because the primary key of the “many side”, Employee, does not include the primary key of the “one side”, Country, so it “can exist” without it

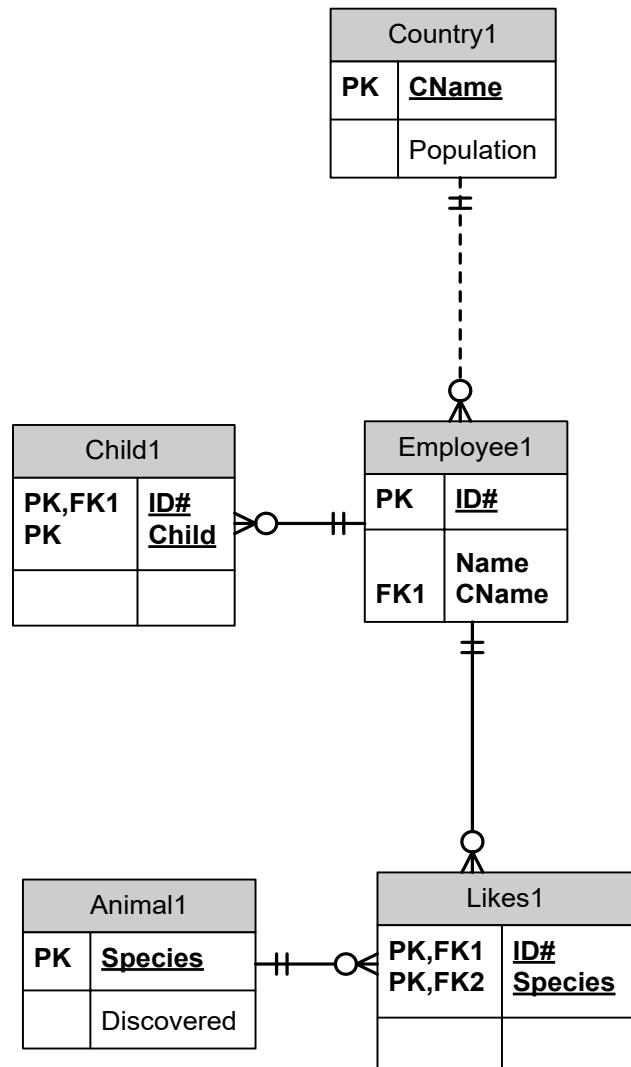


## Pattern Of Lines (2/2)

- ***This is not a question of the ends of lines “forcing” the pattern of lines***
- In the next slide, we see a slight modification of our example in which all lines have the same pair of endings
- We required that for each Employee the Country of Birth is known
- Nevertheless, as Cname is not part of the primary key of Country, the line is dashed
- For technical reasons, the tables have slightly different names, but this has nothing to do with our point



# Example





# Which Implementation To Use For Born?

- We cannot give a general rule
- The first implementation uses more tables
- The second implementation may introduce NULLs (empty values), which we do not like
- For the purpose of the class we will always use the second version, to have better exercises
- So do this for all the homeworks and tests, when relevant



# To Remember!

- Structurally, a relational database consists of
  1. A ***set of tables***
  2. A ***set of many-to-one binary relationships***  
between them, induced by foreign key constraints  
In other words; ***a set of functions (in general partial), each from a table into a table***
- When designing a relational database, you ***must*** specify both (or you will produce a bad specification)
  - » Technically, tables are enough, but this a very bad practice as you do not specify the relationships between tables



# Very Bad Diagram

- Tables are listed with attributes, specifying only which are in the primary key
  - » Even the primary keys are not strictly required
- Foreign key constraints are not specified
  - » So the DB system does not know what to enforce

Country	
PK	<u>CName</u>
	Population

Child	
PK	<u>ID#</u>
PK	<u>Child</u>

Employee	
PK	<u>ID#</u>
	Name
	CName

Animal	
PK	<u>Species</u>
	Discovered

Likes	
PK	<u>ID#</u>
PK	<u>Species</u>



# Terrible Diagram

- Even primary keys are not specified

Country	
	CName Population

Child	
	ID# Child

Employee	
	ID# Name CName

Animal	
	Species Discovered

Likes	
	ID# Species

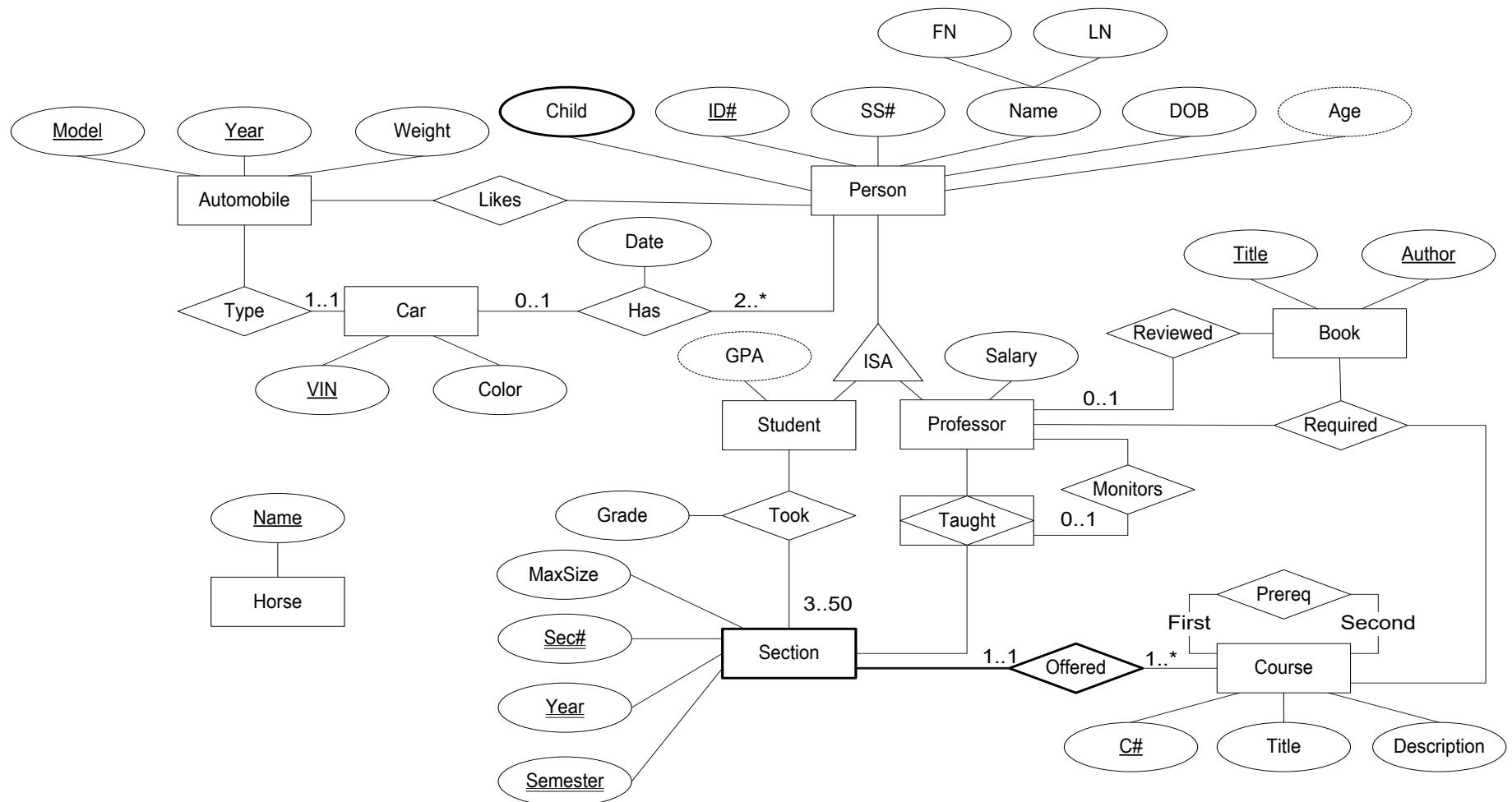


# Another Example of ER Diagram To Relational Database Conversion

- We now convert a big ER diagram into a relational database
- We specify
  - » Attributes that must not be NULL
  - » Primary keys
  - » Keys (beyond primary)
  - » Foreign keys and what they reference
  - » Cardinality constraints
  - » Some additional “stubs”
- We both give a narrative description, similar to actual SQL DDL (so we are learning about actual relational databases) and Visio/Erwin diagrams
- We should specify domains also, but we would not learn anything from this here, so we do not do it
- ***We go bottom up, in the same order as the one we used in constructing the ER diagram***

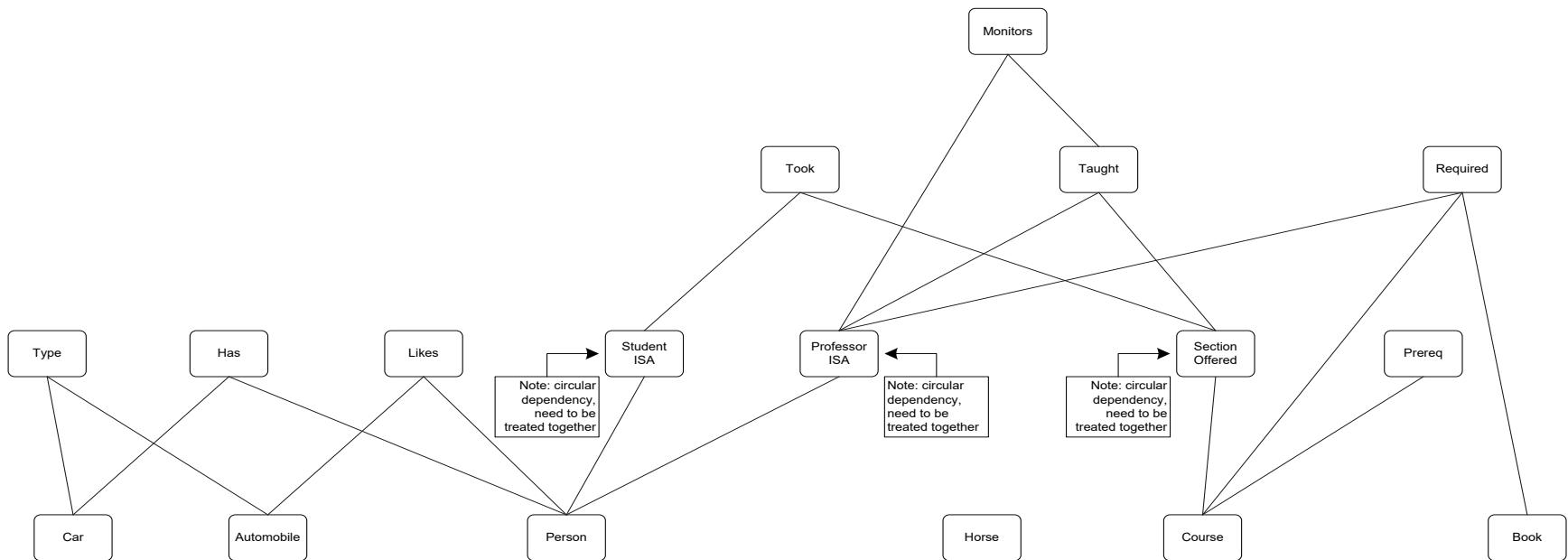


# Our ER Diagram



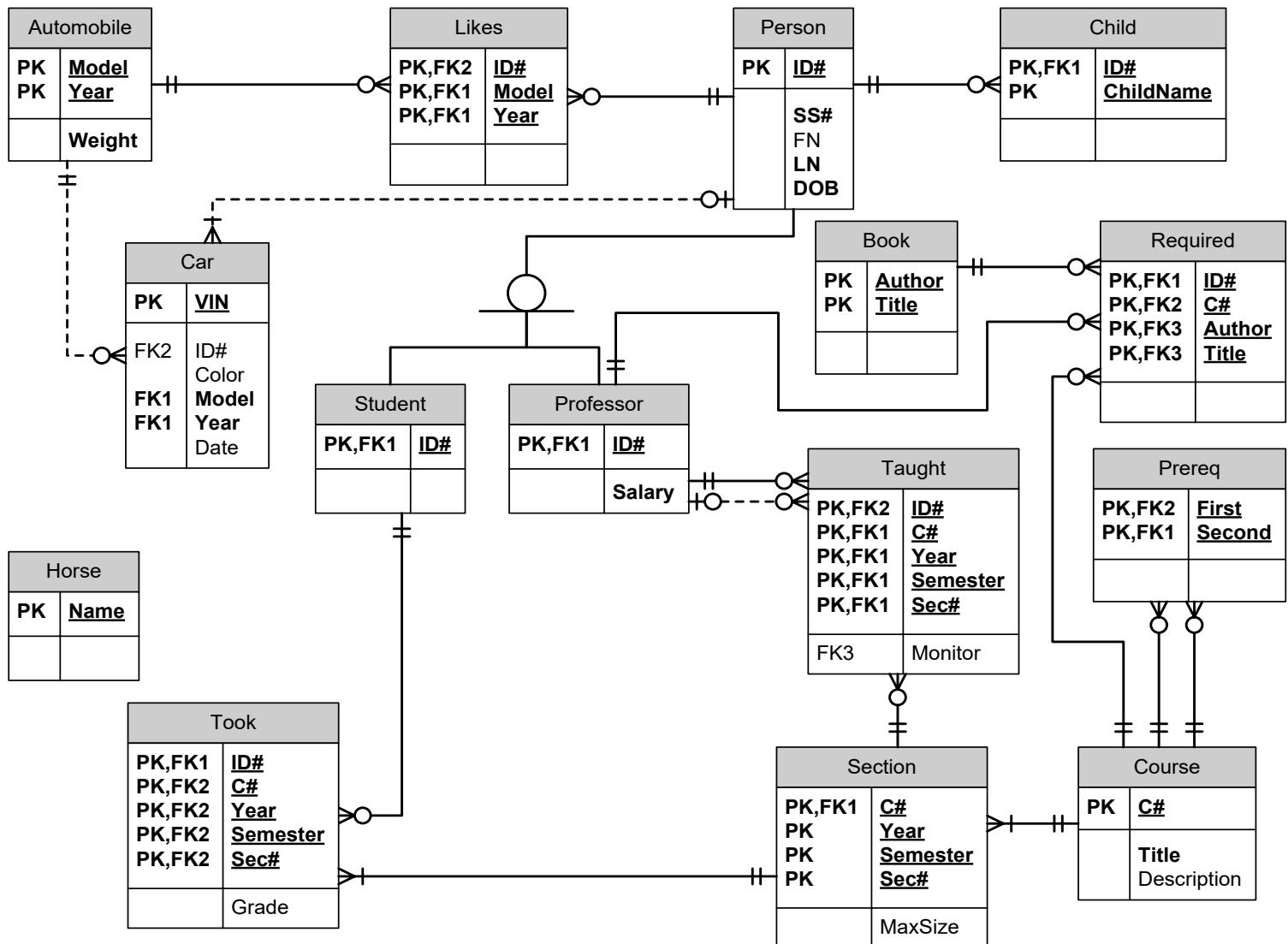


# Hierarchy For Our ER Diagram





# We Will Produce





## Horse (1/2)

- Define Table Horse (  
Name NOT NULL,  
Primary Key (Name));
- This represents the simplest possible relational database
  - » One table with one attribute



# Horse (2/2)

Horse	
PK	<u>Name</u>



- Person has some interesting attributes
- Multivalued attribute: we will create another table
- Derived attribute: we do not create a column for it, it will be computed as needed
- Composite attribute: we “flatten” it



## Person (2/3)

- Define Table Person ( ID# NOT NULL,  
SS# NOT NULL,  
FN,  
LN NOT NULL,  
DOB NOT NULL,  
Primary Key (ID#),  
Candidate Key (SS#),  
Age (computed by procedure ...) );
- In SQL DDL, the keyword UNIQUE is used instead of Candidate Key, but “Candidate Key” is better for reminding us what this could be
- Age would likely not be stored but defined in some view



# Person (3/3)

Person	
PK	<u>ID#</u>
	SS#
	FN
	LN
	DOB

Horse	
PK	<u>Name</u>



- Define Table Child (  
ID# NOT NULL,  
ChildName NOT NULL,  
Primary Key (ID#,ChildName),  
Foreign Key (ID#) References Person );
- This lists all pairs (ID# of person, a child's name)
  - » We have chosen a more descriptive attribute name than the one in the ER diagram for children's names
- Note
  - » A person may have several children, each with a different name
  - » Two different persons may have children with the same name
- Because of this, no single attribute can serve as primary key of Child

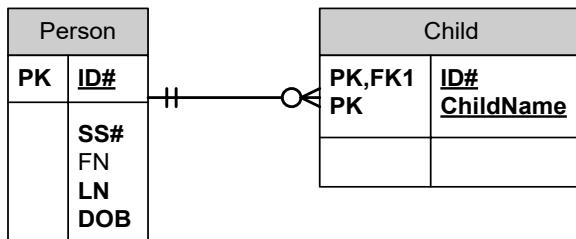


## Person And Child (1/2)

- Note that some attributes are not bold, such as FN here
- This means that FN could be NULL (in this context, meaning empty)
- Note the induced many-to-one relationship
- We need to make sure we understand what the line ends indicate
  - » A person may have 0 or more children (unbounded)
  - » A child has exactly 1 person to whom it is attached
- We need to pay attention to such matters, though we are generally not going to be listing them here  
But you should look at all lines and understand the ends and the patterns (solid or dashed)



## Person And Child (2/2)



Horse	
PK	Name



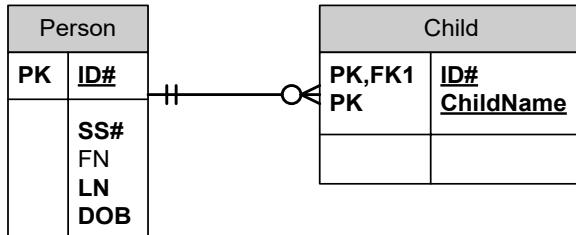
## Automobile (1/2)

- Define Table Automobile (Model NOT NULL, Year NOT NULL, Weight NOT NULL, Primary Key (Model,Year) );



# Automobile (2/2)

Automobile	
PK	<u>Model</u>
PK	Year
	Weight



Horse	
PK	<u>Name</u>

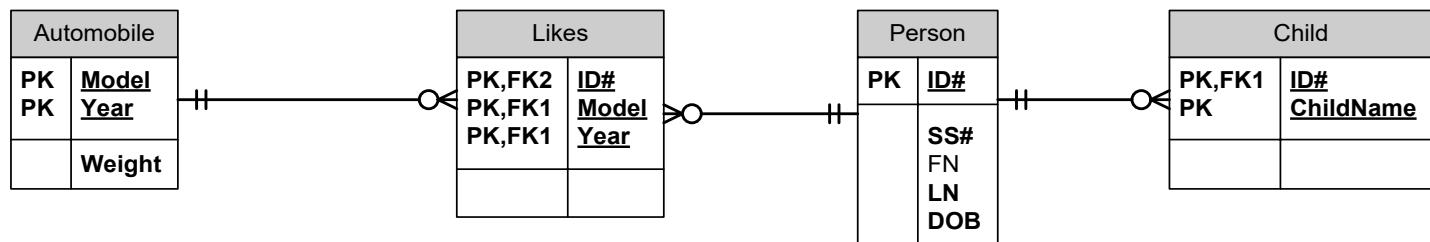


## Likes (1/2)

- Define Table Likes ( ID# NOT NULL,  
Model NOT NULL,  
Year NOT NULL,  
Primary Key (ID#,Model,Year),  
Foreign Key (ID#) References Person,  
Foreign Key (Model,Year) References Automobile );
  
- There are induced binary many-to-one relationships between
  - » Likes and Person
  - » Likes and Automobile



## Likes (2/2)



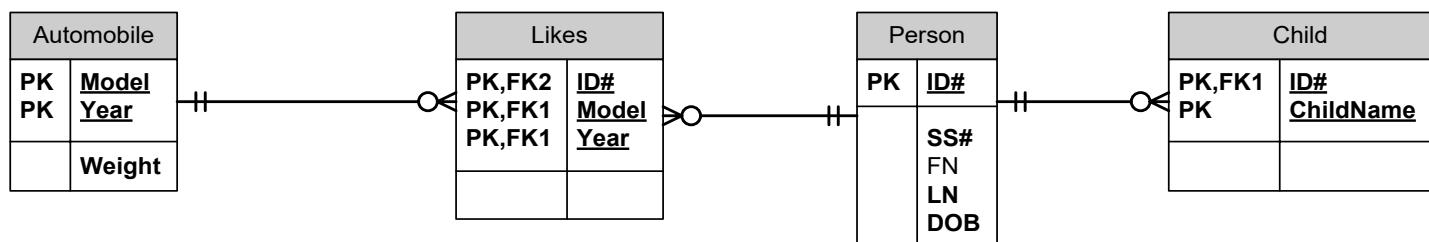
Horse	
<u>PK</u>	<u>Name</u>



- Define Table Car (  
VIN NOT NULL,  
Color,  
Primary Key (VIN) );



# Car (2/2)



Car	
PK	VIN
	Color

Horse	
PK	Name



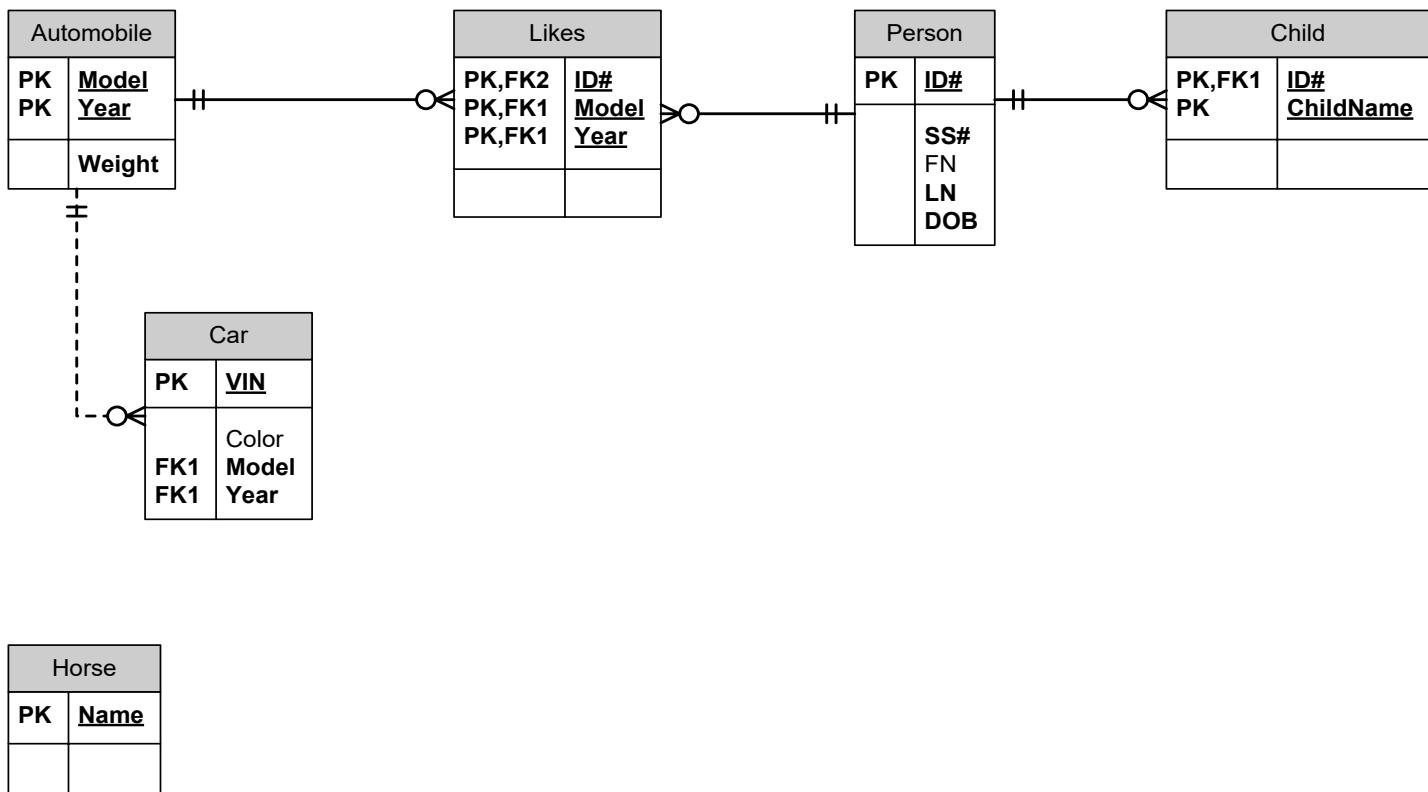
- There is no need for a table for Type as Type is a binary many-to-one relationship
- It is essentially “stored” in the “many” side, that is in Car



- Define Table Car (VIN NOT NULL,  
Color,  
Model NOT NULL,  
Year NOT NULL,  
Weight NOT NULL,  
Primary Key (VIN),  
Foreign Key (Model,Year) References  
Automobile );



# Type





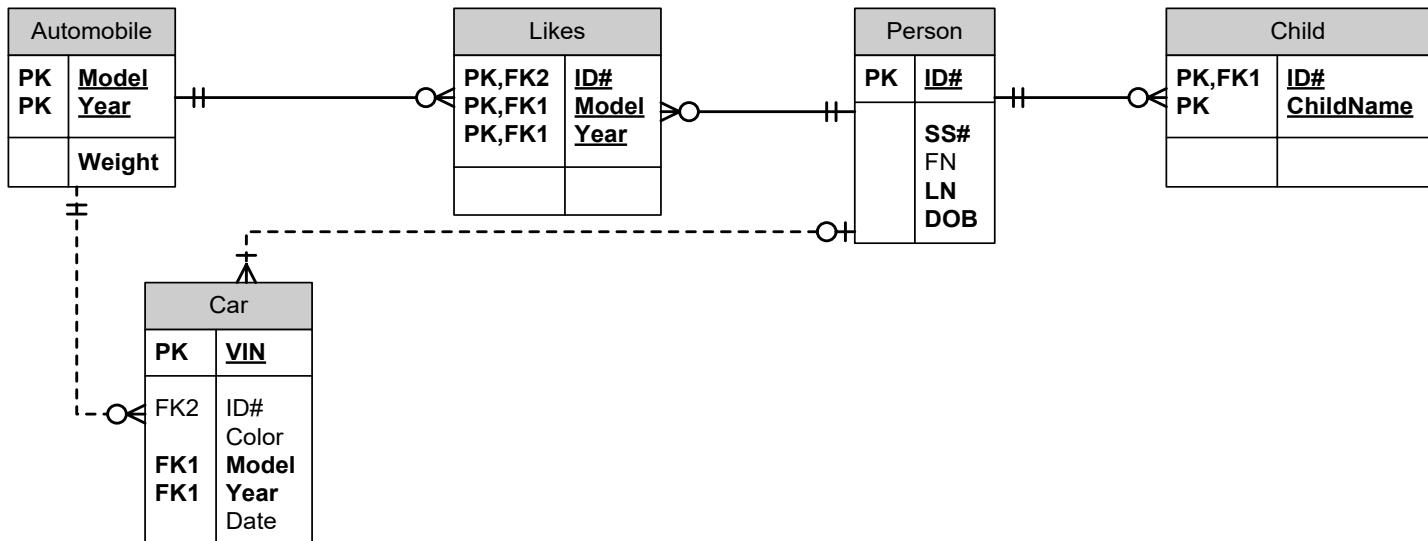
- As Has is a binary many-to-one relationship, the attribute of this relationship, Date, is stored in the “many” side, Car
  - There is no need for a table for Has as Has is a binary many-to-one relationship
  - It is essentially “stored” in the “many” side, that is in Car
  - We can only specify that a Person has at least 1 Car with the notation we currently use
  - The CHECK condition is specified using appropriate SQL constraint syntax
- This can actually be done in Visio/Erwin also



- Define Table Car (VIN NOT NULL,  
Color,  
Model NOT NULL,  
Year NOT NULL,  
Weight NOT NULL,  
ID#,  
Primary Key (VIN),  
Foreign Key (Model,Year) References  
Automobile  
Foreign Key (ID#) References Person );



# Has



Horse	
PK	Name



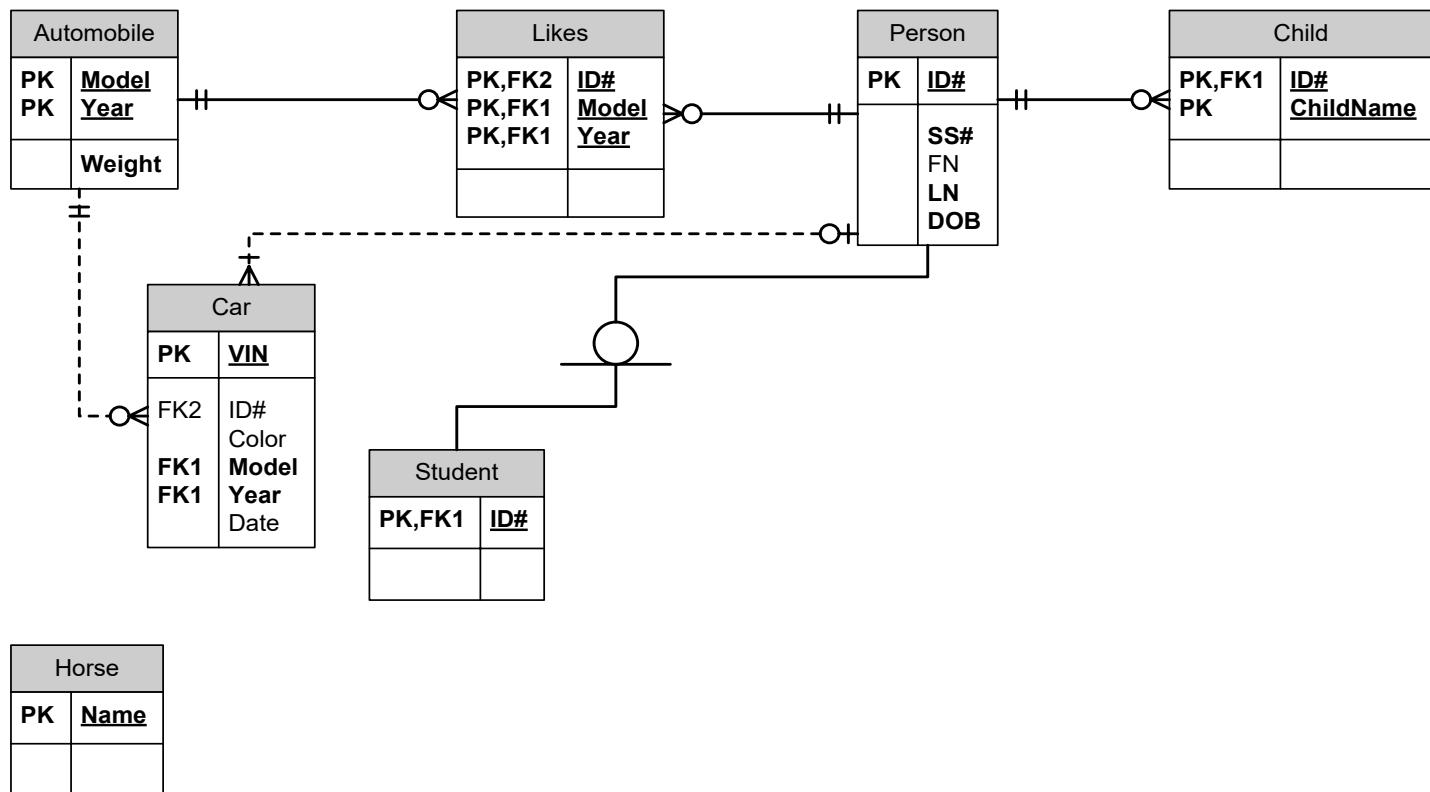
- We do not define a table for ISA
- This/these relationship/s is/are “embedded” in Student and Professor



- Define Table Student ( ID# NOT NULL,  
Primary Key (ID#),  
Foreign Key (ID#) References Person,  
GPA (computed by procedure ...) );
- Note, how ISA, the class/subclass  
(set/subset) relations, is modeled by  
Visio/Erwin



# Student And ISA

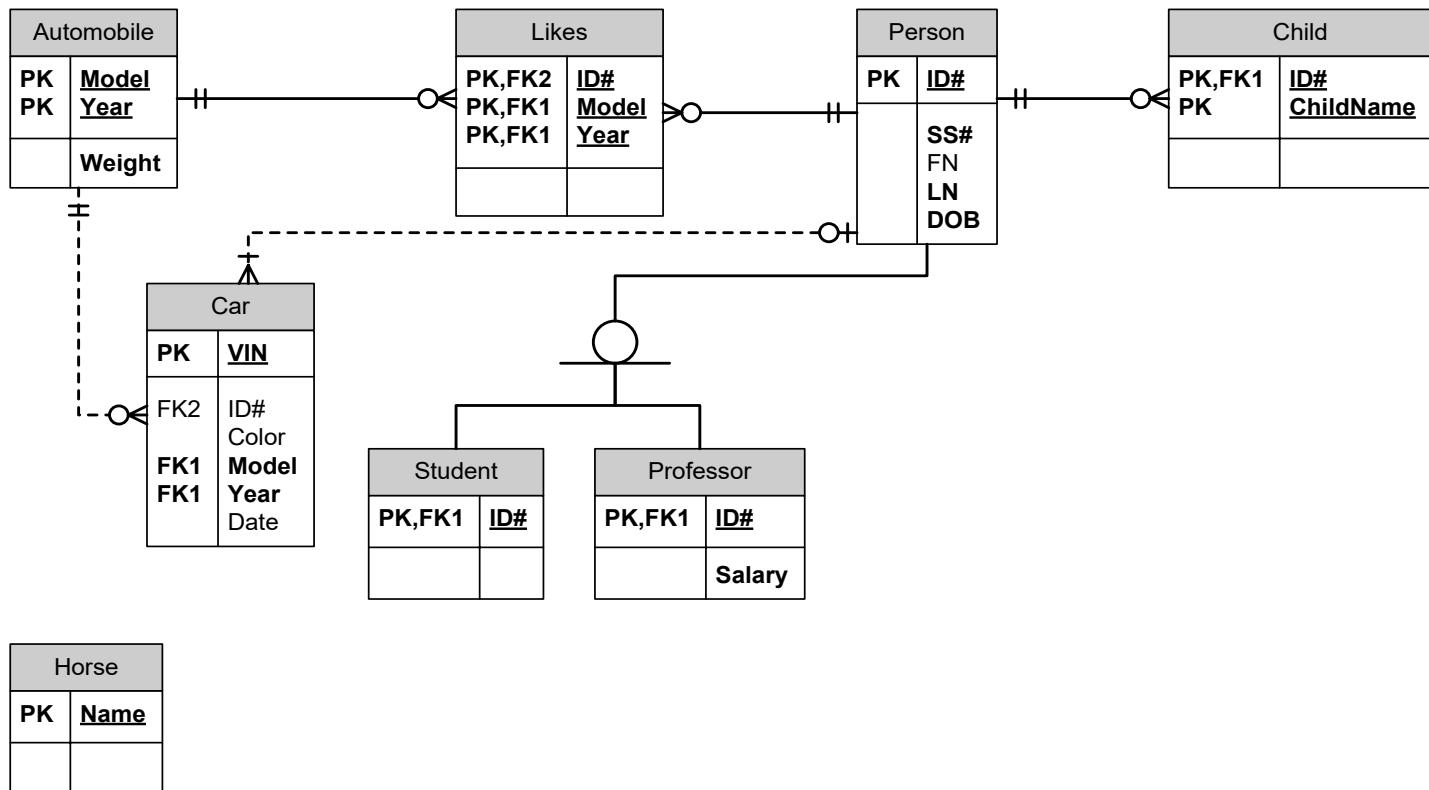




- Define Table Professor ( ID# NOT NULL,  
Salary NOT NULL,  
Primary Key (ID#),  
Foreign Key (ID#) References Person );



# Professor And ISA

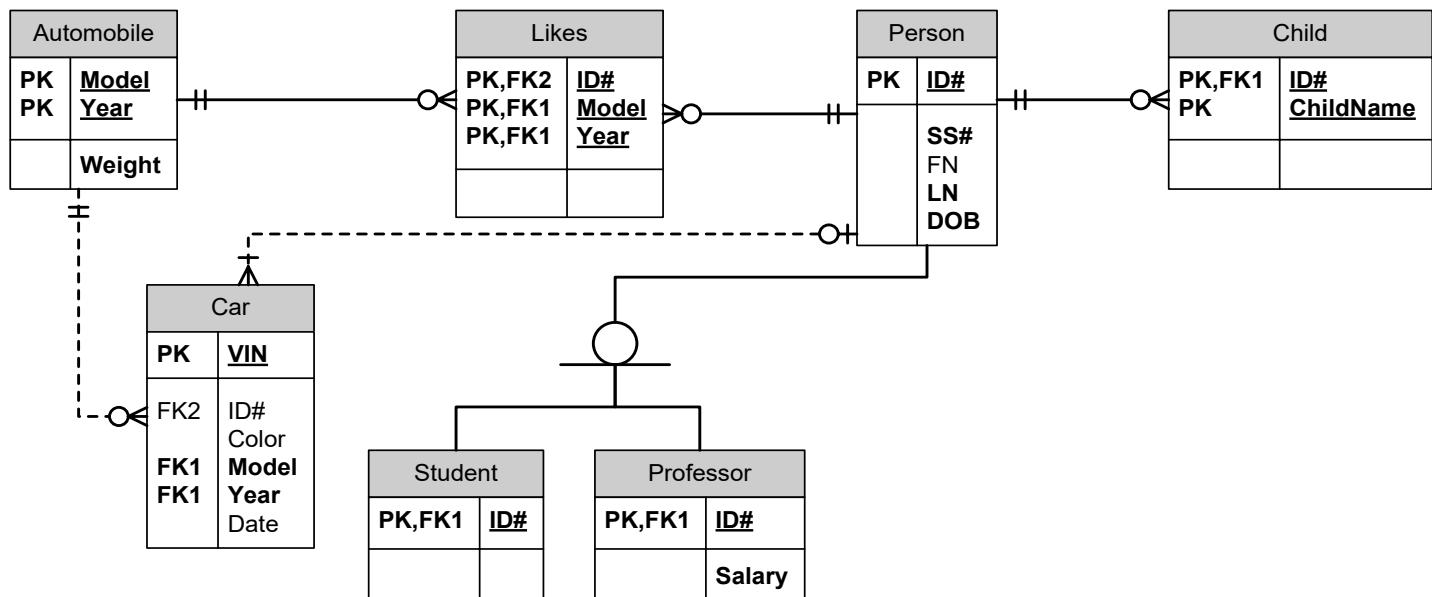




- Define Table Course (C# NOT NULL,  
Title NOT NULL,  
Description,  
Primary Key (C#) );



# Course (2/2)



Horse	
PK	Name

Course	
PK	C#
	Title Description



- Define Table Prereq (First NOT NULL,  
Second NOT NULL,  
Primary Key (First,Second),  
Foreign Key (First) References Course,  
Foreign Key (Second) References Course );

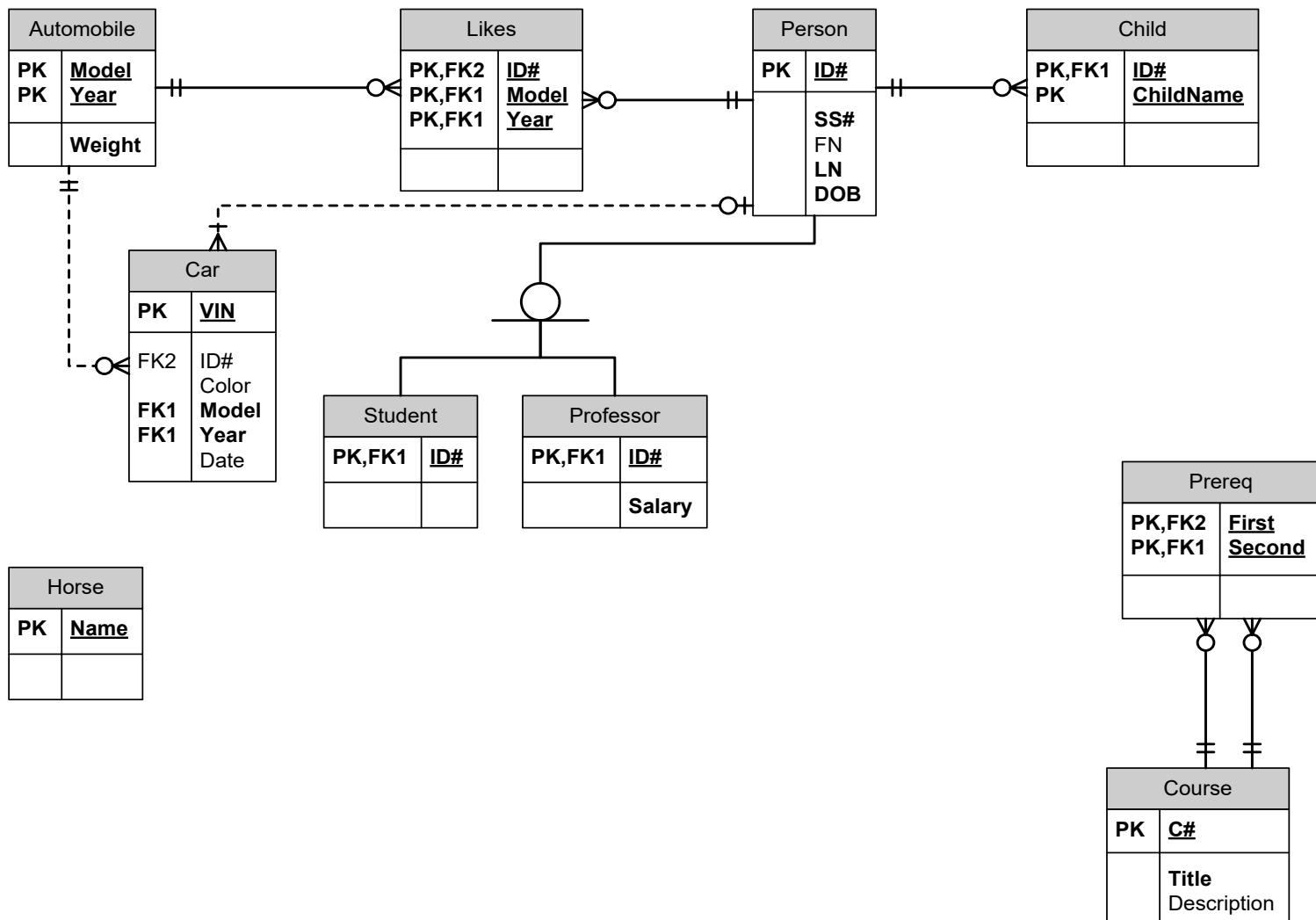


## Prerequisite (2/3)

- This is our first example of a table modeling a recursive relationship, between an entity set and itself
- We decide to name the table Prereq, as this is shorter than Prerequisite
- Note that it is perfectly clear and acceptable to refer here to C# by new names: First and Second
  - » Similarly, to using ChildName in the Child table
- We should add some constraint to indicate that this (directed graph) should be acyclic
  - » Maybe other conditions, based on numbering conventions specifying course levels



# Prerequisite (3/3)

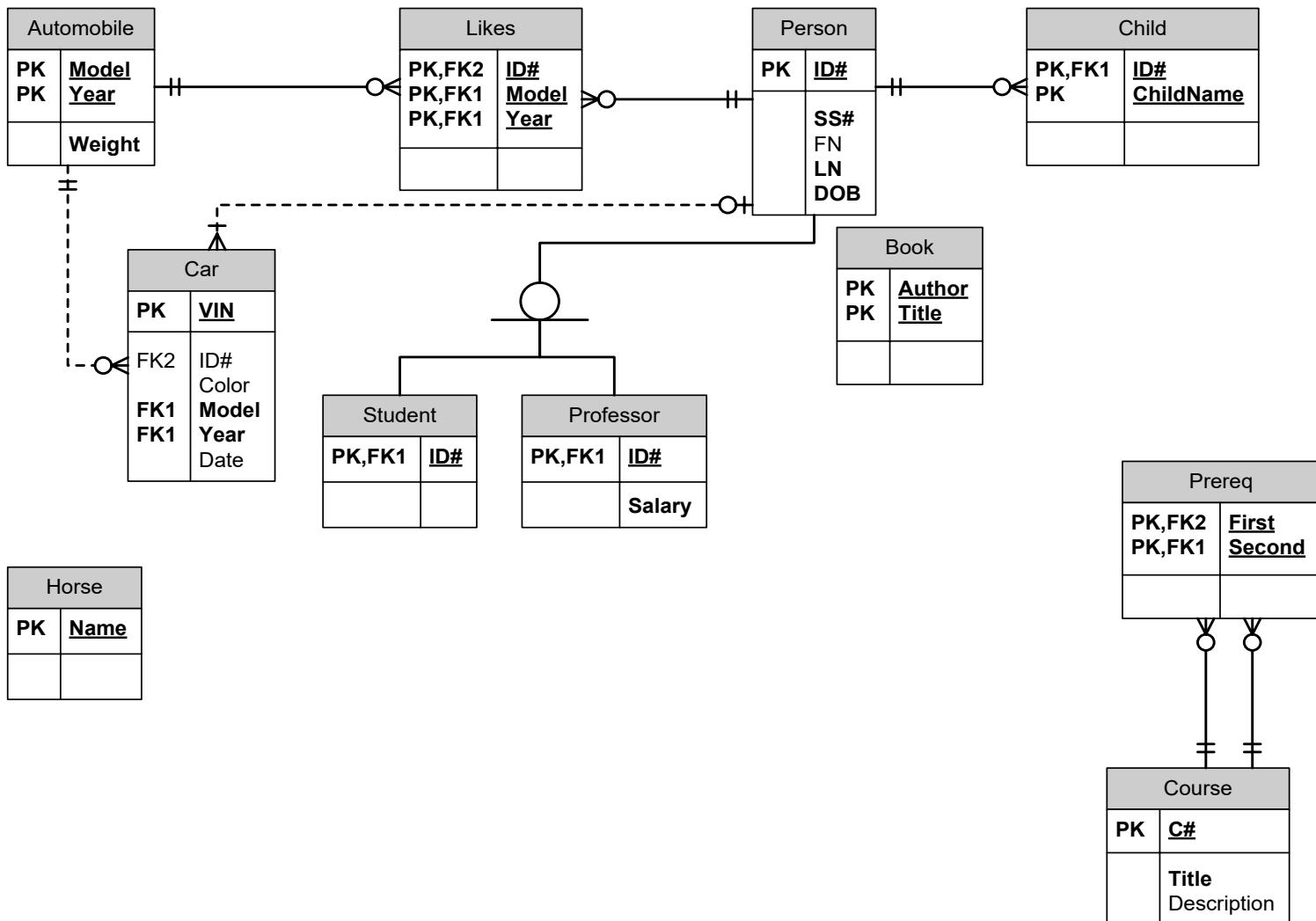




- Define Table Book (Author NOT NULL,  
Title NOT NULL,  
Primary Key (Author, Title) );



# Book (2/2)





## Required (1/3)

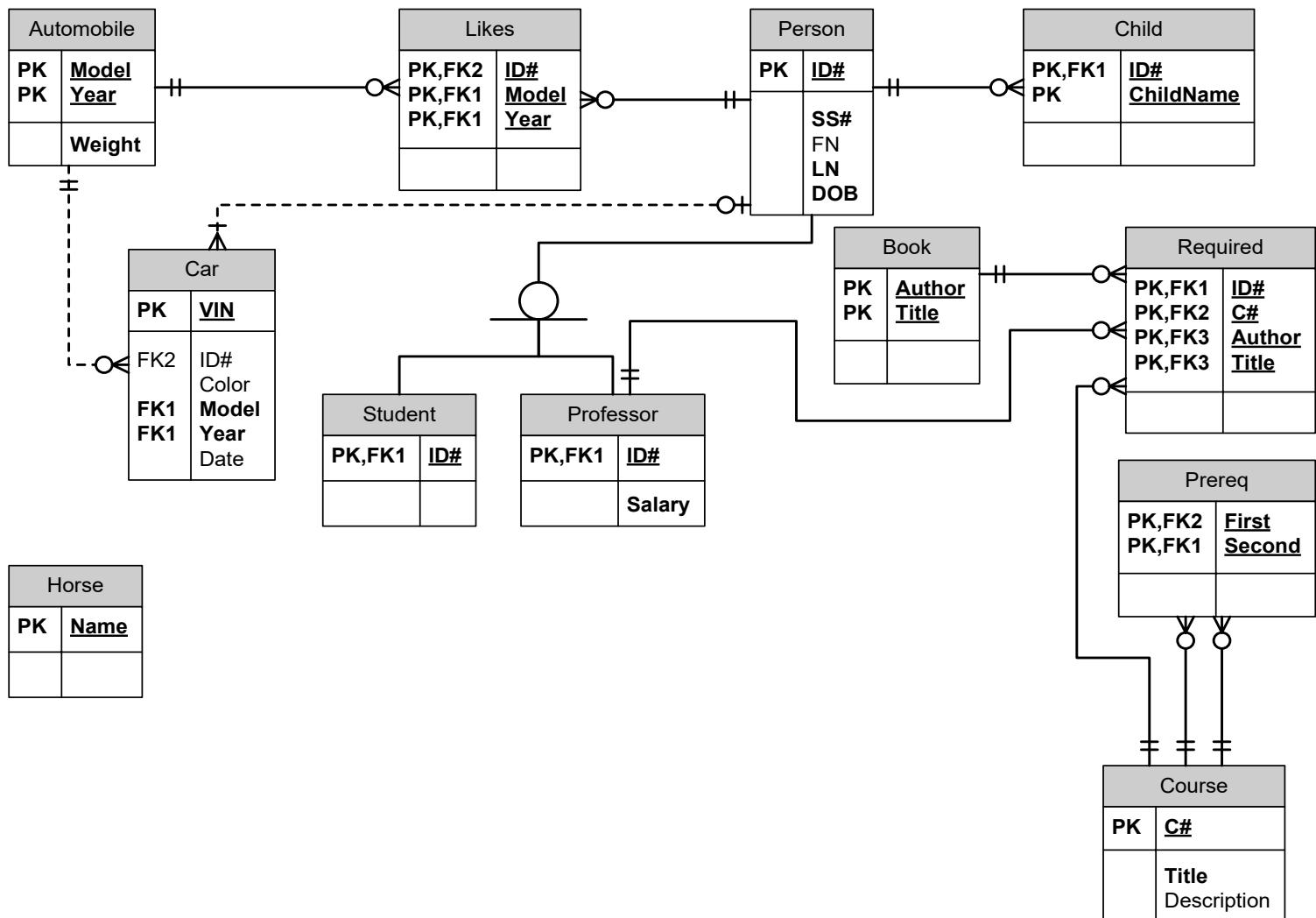
- Define Table Required ( ID# NOT NULL,  
C# NOT NULL,  
Author NOT NULL,  
Title NOT NULL,  
Primary Key (ID#,C#,Author,Title),  
Foreign Key (ID#) References Professor,  
Foreign Key (C#) References Course,  
Foreign Key (Author,Title) References Book );
  - Why is it **bad** to have
    - Foreign Key (ID#) References Person,
    - instead of
    - Foreign Key (ID#) References Professor?
- Because only a Professor can Require a Book



- This is our first example of a table modeling a relationship that is not binary
- Relationship Required was ternary: it involved three entity sets
- There is nothing unusual about handling it
- We still have as foreign keys the primary keys of the “participating” entities



# Required (3/3)





## Section (1/2)

- Define Table Section ( C# NOT NULL,  
Year NOT NULL,  
Semester NOT NULL,  
Sec# NOT NULL,  
MaxSize,  
Primary Key (C#,Year,Semester,Sec#),  
Foreign Key (C#) References Course );
- Note on the end of the edge between Course and Section, the Section end, on the drawing how the requirement of having at least one Section is modeled



## Section (2/2)

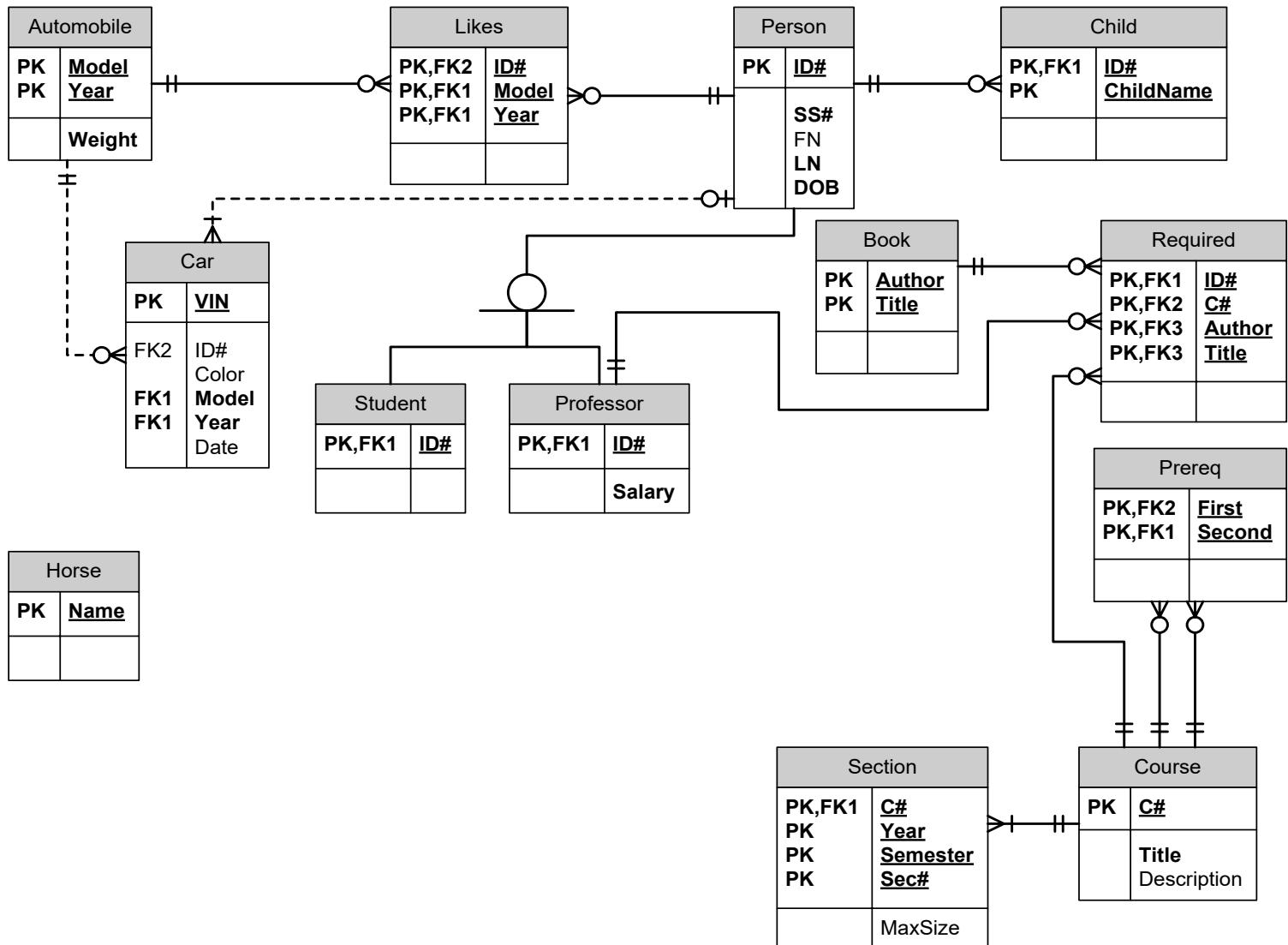
- Section is our first example of a weak entity



- We do not define a table for Offered
- Relationship Offered is implicit in the foreign key constraint



# Section + Offered





## Took (1/3)

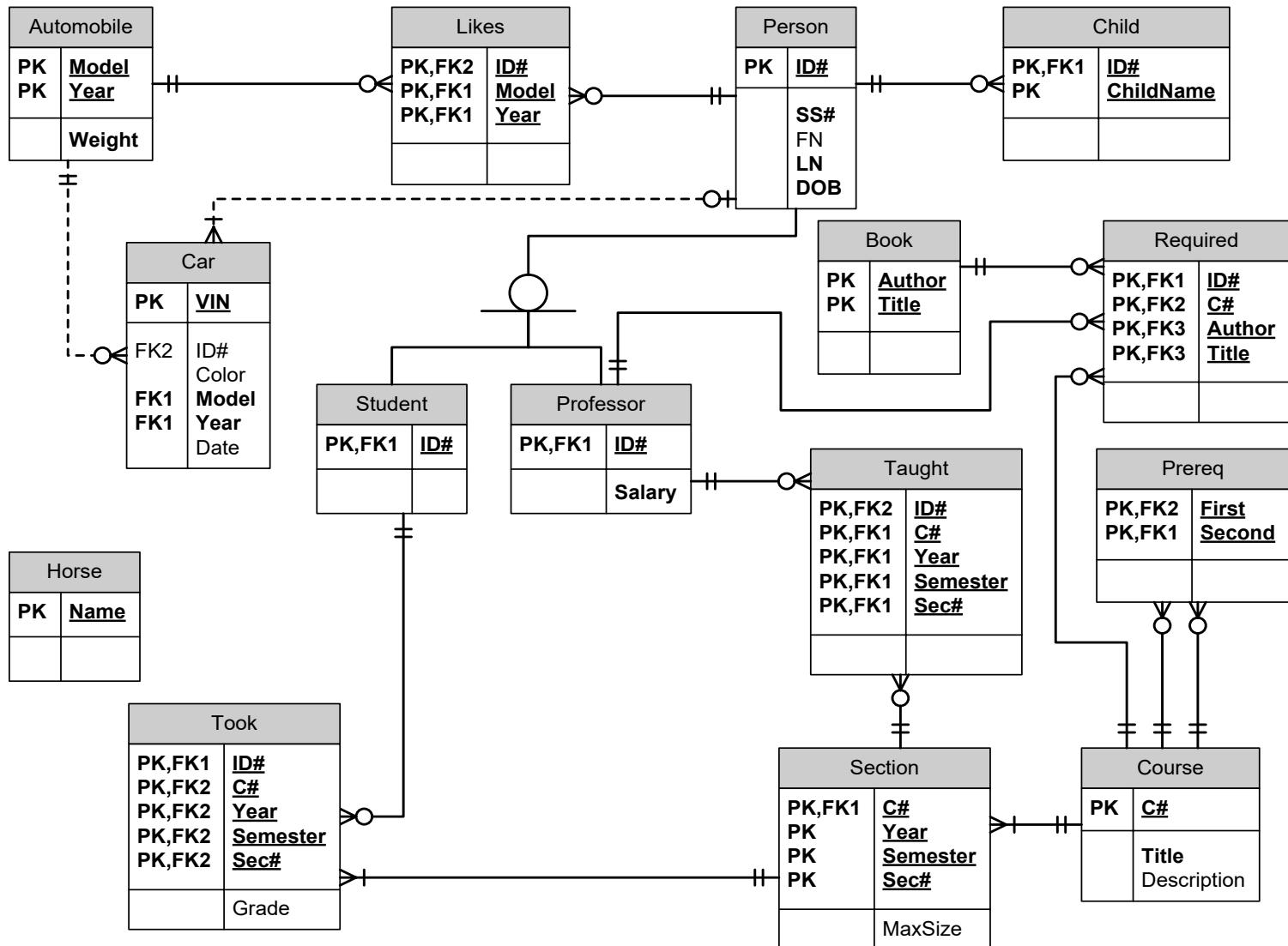
- Define Table Took (  
ID# NOT NULL,  
C# NOT NULL,  
Year NOT NULL,  
Semester NOT NULL,  
Sec# NOT NULL,  
Grade,  
Primary Key (ID#,C#,Year,Semester,Sec#),  
Foreign Key (ID#) References Student,  
Foreign Key (C#,Year,Semester, Sec#) References  
Section );
- Note on the end of the edge between Section and Took, the Took  
end, on the drawing how the requirement of having between 3 and  
50 students in a section is not fully modeled
- We can only show 1 or more using current notation



- Because Took is a many-to-many relationship we store its attribute, Grade, in its table
- We cannot store Grade in any of the two
  - » Section
  - » Student



# Took (3/3)



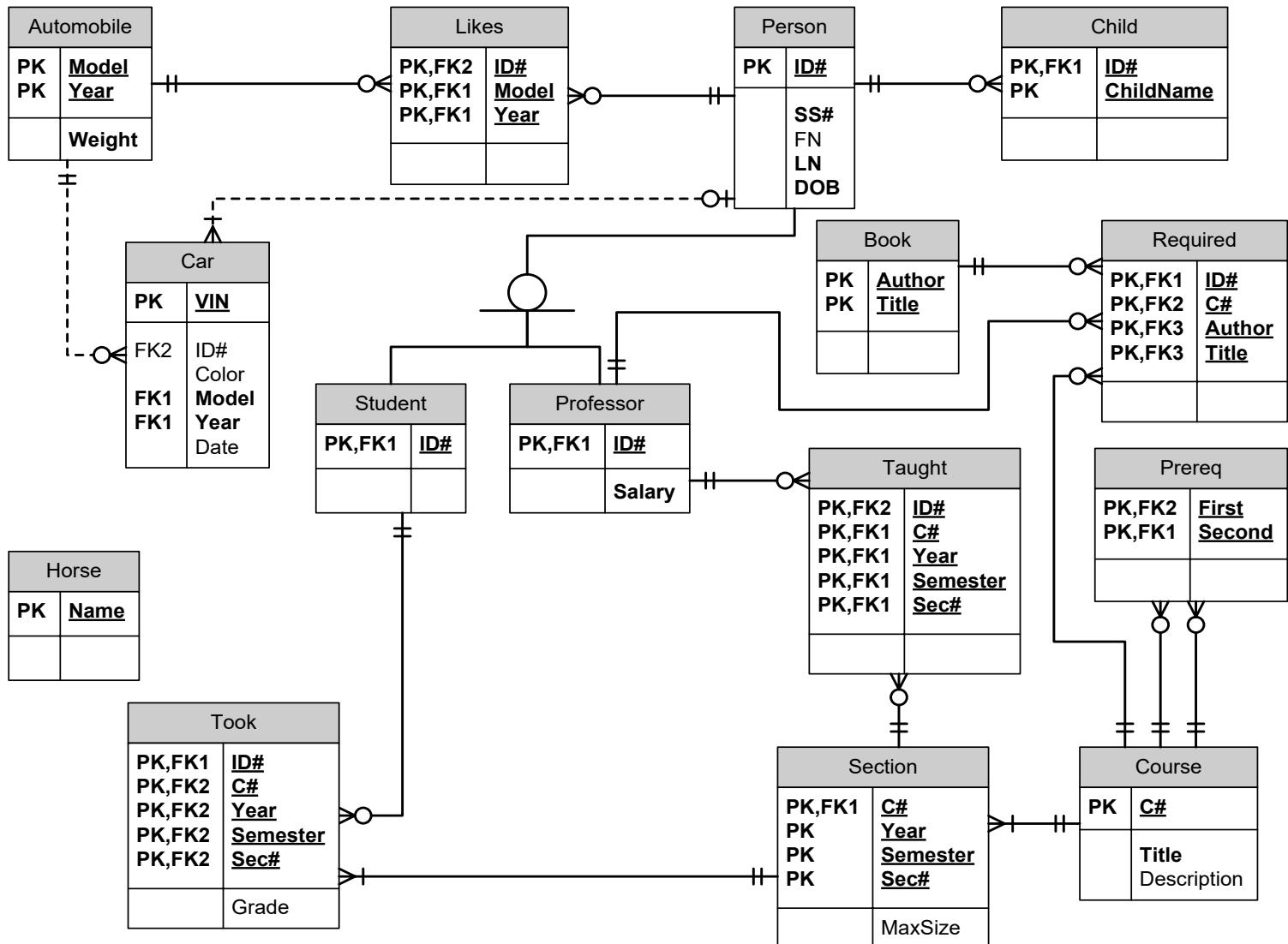


## Taught (1/2)

- Define Table Taught (  
ID# NOT NULL,  
C# NOT NULL,  
Year NOT NULL,  
Semester NOT NULL,  
Sec# NOT NULL,  
Primary Key (ID#,C#,Year,Semester,Sec#),  
Foreign Key (ID#), References Professor,  
Foreign Key (C#,Year,Semester,Sec#)  
References Section );



# Taught (2/2)



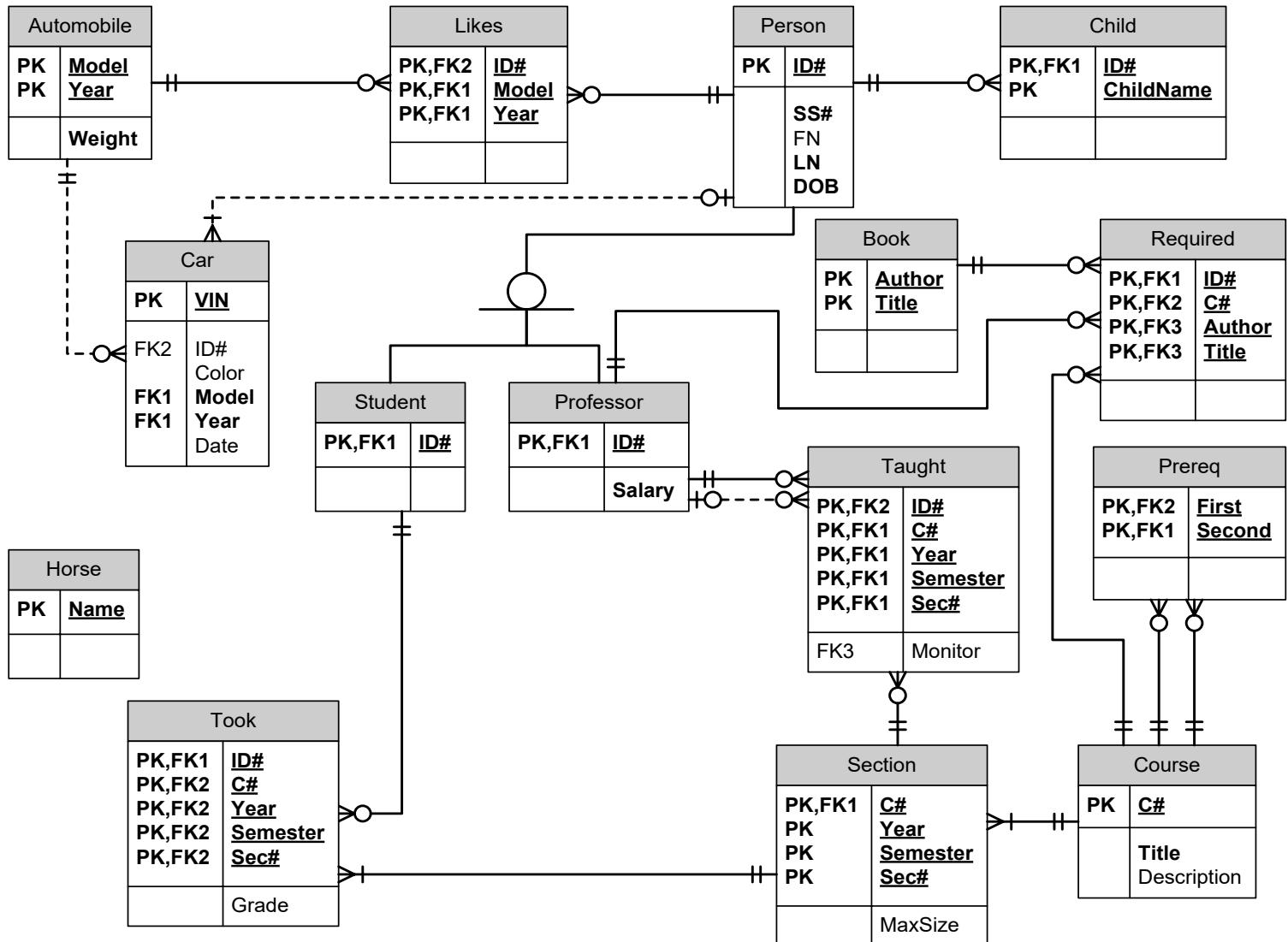


- This is our first example in which a table, Taught, that “came from” a relationship is treated as if it came from an entity and participates in a relationship with other tables
- Nothing special needs to be done to “convert” a table that models a relationship, to be also treated as a table modeling an entity
- In this case, Monitors is a binary many-to-one relationship, so we do not need to create a table for it, and it can be stored in the “many” side, Taught



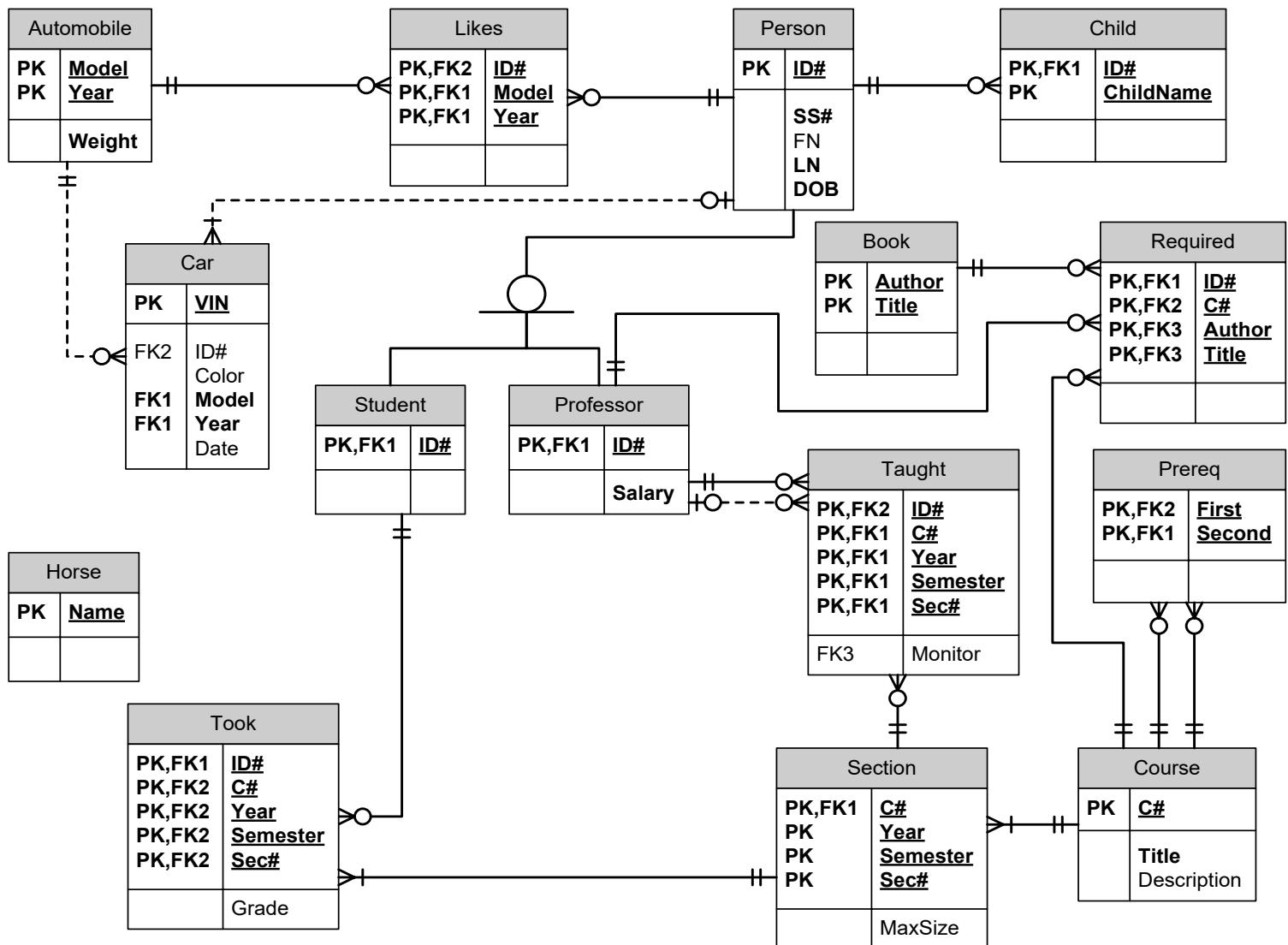
- Define Table Taught (  
ID# NOT NULL,  
C# NOT NULL,  
Year NOT NULL,  
Semester NOT NULL,  
Sec# NOT NULL,  
Monitor  
Primary Key (ID#,C#,Year,Semester,Sec#),  
Foreign Key (ID#), References Professor,  
Foreign Key (C#,Year,Semester,Sec#)  
References Section  
Foreign Key (Monitor) References Professor );

# Monitors



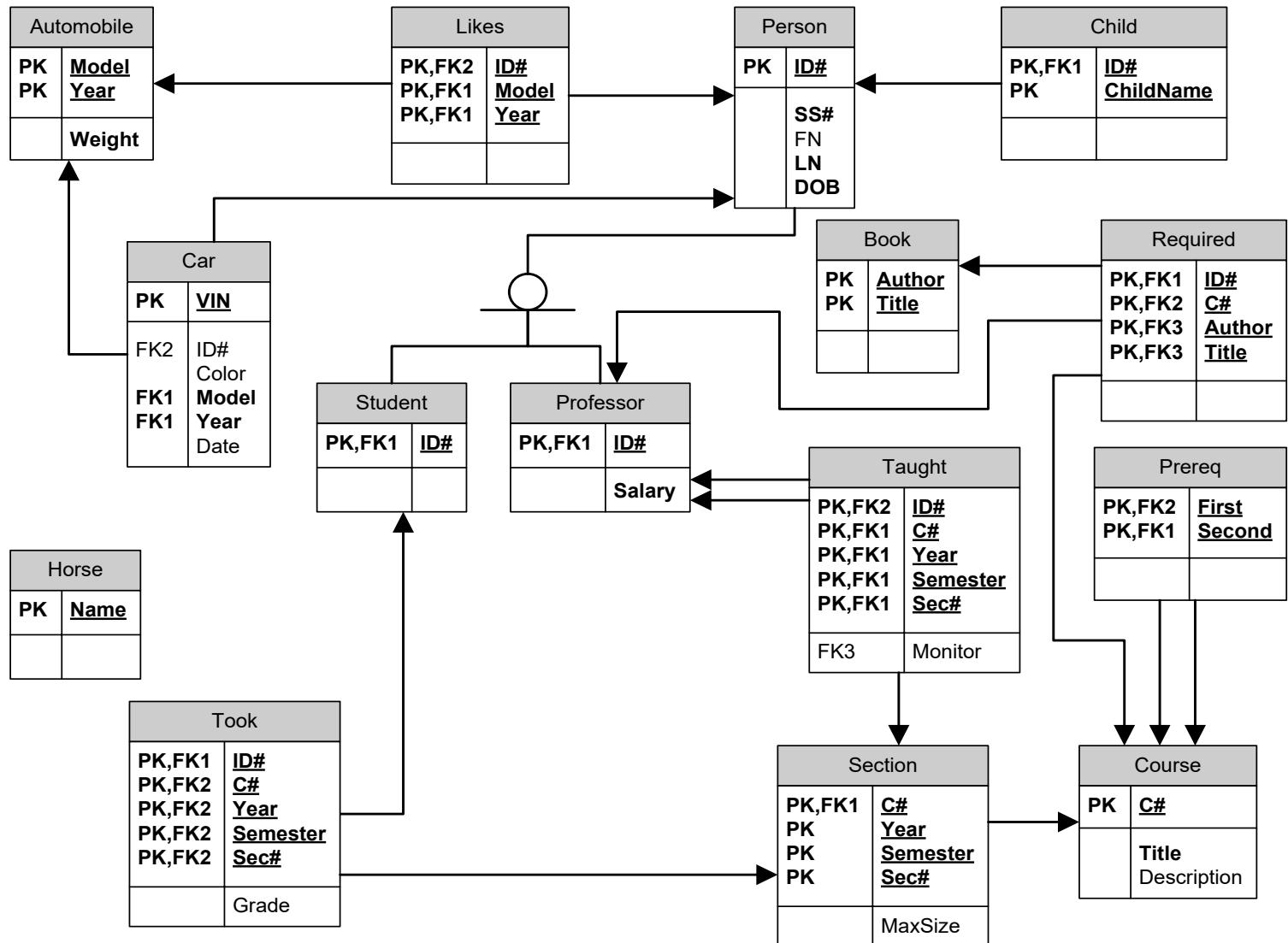


# We Are Done



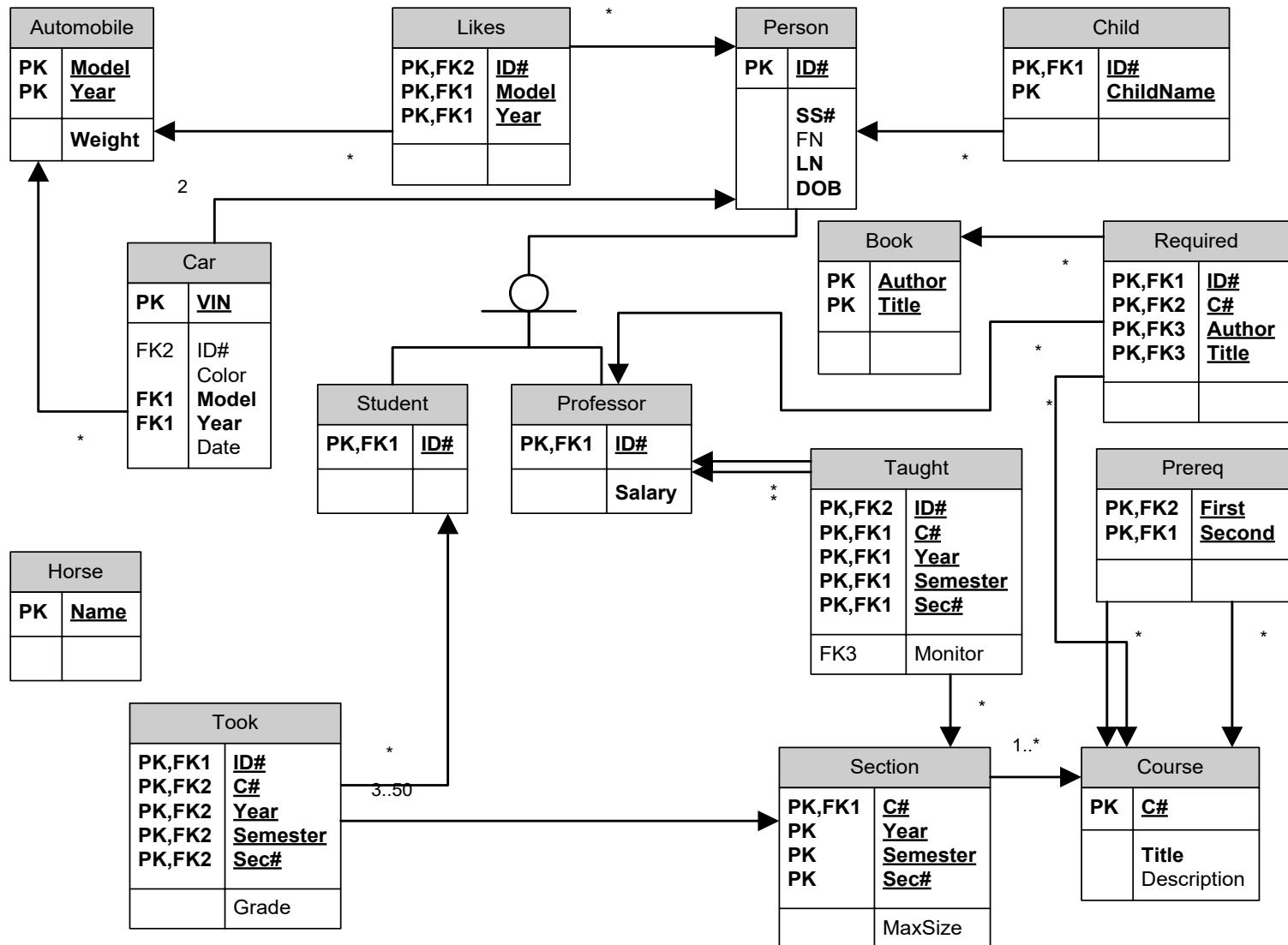


# Arrow Notation





# Arrows And Cardinality Notation



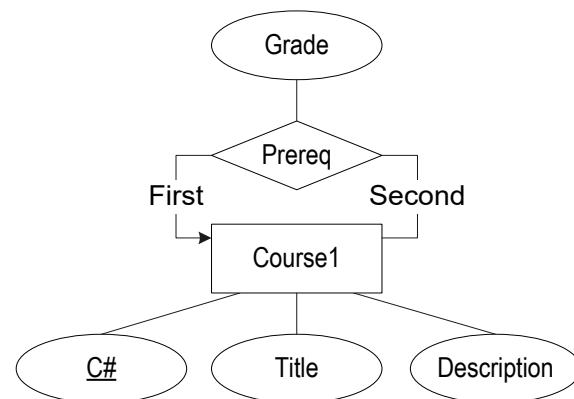
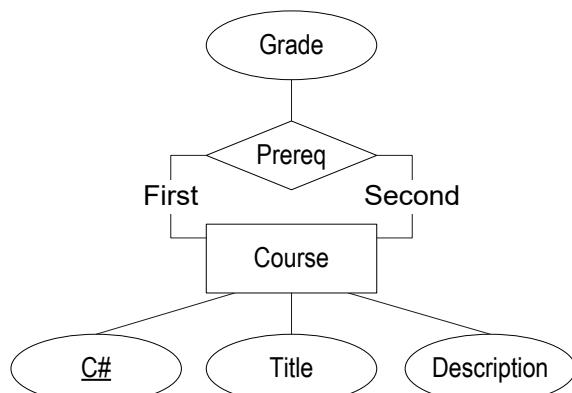


- We will discuss some additional, important, points
  - » Elaboration on recursive relationships
  - » Referential Integrity
  - » Temporal databases



# Recursive Relationships: Example (1/2)

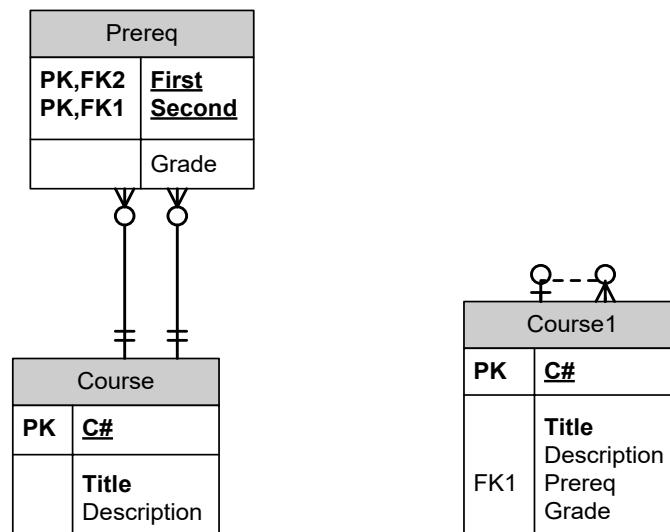
- Assume now that a prerequisite course, “First” course, must be taken with at least some Grade to count as a prerequisite
- This to make an example a little “richer”
- Two cases:
  - » A course may have any number of prerequisites  
Prereq is many-to-many
  - » A course may have at most one prerequisite  
Prereq is many to one (Second is the many side, a single First could be a prerequisite for many Second courses)





## Recursive Relationships: Example (2/2)

- Nothing special, we handle the second case of Prereq by storing it in the “many” side of the relationship
- So there are two additional attributes in Course1
  - » The prerequisite course, if any
  - » The required grade, if any





## Referential Integrity: Example (1/3)

Professor	ID#	Salary
	5	1
	7	2

Taught	ID#	C#	Year	Semester	Sec#	Monitor
5	G22.2433	2009	Spring	001	7	

- Assume that we have some professors in table Professor, with rows: 5,1 and 7,2
- There is a row in Taught  
5,G22.2433,2009,Spring,001,7
- This means that 5 teaches a specific section and 7 monitors this assignment



## Referential Integrity: Example (2/3)

Professor	ID#	Salary
	5	1
	7	2

Taught	ID#	C#	Year	Semester	Sec#	Monitors
	5	G22.2433	2009	Spring	001	7

- A user accesses the database and **attempts to delete row** (or all rows like this, recall that duplicates are permitted) 5,1 **from Professor**
- What should happen, as there is a row in Taught referencing this row in Professor?
- A user accesses the database and attempts to delete row 7,2 from Professor?
- What should happen, as there is a row in Taught referencing this row in Professor?



## Referential Integrity: Example (3/3)

- Part of specification of foreign key in Taught
- An action on Professor can be denied, or can trigger an action on Taught
- For example

- » ON DELETE NO ACTION

This means that the “needed” row in Professor cannot be deleted

Of course, it is possible to delete the row from Taught and then from the Professor (if no other row in any table in the database “needs” the row in Professor)

- » ON DELETE CASCADE

This means that if a row is deleted from Professor, all the rows in Taught referring to it are deleted too

- » ON DELETE SET NULL

This means, that the value referring to no-longer-existing professor is replaced by NULL

In our example, this is not possible for ID# as it is a part of the primary key of Taught, but is possible for Monitor



# Referential Integrity: Another Example

- Part of specification of foreign key in Professor
- An action on Person can be denied, or can trigger an action on Professor
- For example
  - » ON UPDATE CASCADE

This means that if the value of ID# in Person is changed, this value of ID# also propagates to Professor
- Could (and probably should) add to Taught and Required:
  - » ON UPDATE CASCADE
    - In appropriate attributes, so that the change of ID# in Professor also propagates to them
    - In Taught in both ID# and Monitor
    - In Required in ID#
- Excellent mechanism for centralized maintenance



- Of course, we may want to maintain historical data
- So, in practice one may have some indication that the professor no longer works, but still keep historical information about the past
- But we do not assume this for our example



## Summary: Strong Entity (1/2)

- Example: **Person**
- Create a table for the entity without multivalued and derived attributes, flattening composite attributes

The primary key of this table will consist of the attributes serving as primary key of the entity

Example table: Person

- If there is a derived attribute, describe how it is computed, but do not store it
- If there is a multivalued attribute, create a table for it consisting of it and attributes of the primary key of the entity; do not put it in the table for the entity

Example table: Child

The primary key of this table will consist of all its attributes



## Summary: Strong Entity (2/2)

- There could be an attribute that is composite with some components being multivalued and some derived
- And similar complexities
- Example, without drawing the appropriate entity using the ER model (this is getting too hairy)
  - » A person has many children (multivalued)
  - » Each child has both FirstName and MiddleName
  - » The child has DOB
  - » The child has Age
- Then the table for child will look like

Child	ID#	FirstName	MiddleName	DOB
	5432	Krishna	Satya	2006-11-05



## Summary: ISA And A Subclass

- Example: ***ISA*** and ***Professor***
- Do not do anything for ISA
- The class “above” ISA (here Person) has already been implemented as a table
- Create a table with all the attributes of the subclass (as for strong entity above) augmented with the primary key of the table “above” ISA, and no other attributes from it

The primary key is the same as the primary key of the table “above” ISA

Example table: Professor



# Summary: Weak Entity And Defining Relationship

- Example: **Offered** and **Section**
  - Do not do anything for the defining relationship, here Offered
  - Imagine that the weak entity is augmented by the primary key of the “stronger” table through which it is defined (the table for it has been created already)  
Treat the augmented weak entity the same way as a strong entity  
The primary key is the primary key of the “stronger” table augmented by the attributes in the discriminant of the weak entity (a discriminant may consist of more than one attribute)
- Example table: Section and Offered



# Summary: A Relationship That Is Not Binary Many-To-One

- Example **Took**

The tables for the participating entities have already been created

Create a table consisting of the primary keys of the participating tables and the attributes of the relationship itself

Of course, treat attributes of the relationship that are derived, multivalued, or composite, appropriately, not storing them, producing additional tables, flattening them

The primary key consists of all the attributes of the primary keys of the participating tables

Example table: Took



# Summary: A Relationship That Is Binary Many-To-One

- Example: **Has**

Do not create a table for this relationship

Put the attributes of the primary key of the “one” side and the attributes of the relationship itself into the table of the “many” side

Of course, treat attributes of the relation that are derived, multivalued, or composite, appropriately, not storing them, producing additional tables, flattening them, as the case may be

You may decide to treat such a relationship the way you treat a relationship that is not binary many to one (but not in our class)

If the relationship is one-to-one, choose which side to treat as if it were “many”

Example table: Has



## Summary: Treating A Relationship As An Entity

- Example: **Taught** (before it was modified by removing Approved)
    - We have a table for that was created when we treated it as a relationship
    - We do not need to do anything else to this table
- Example table: Taught

# Agenda

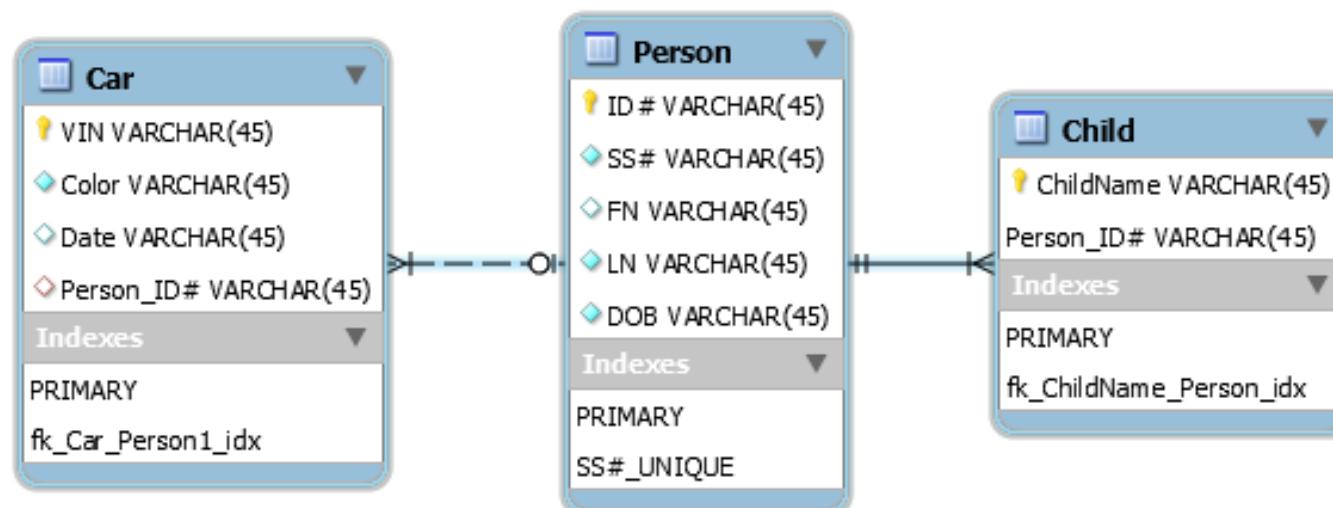
- 
- 1 Session Overview
  - 2 ER and EER to Relational Mapping
  - 3 Database Design Methodology and UML
  - 4 Mapping Relational Design to ER/EER Case Study
  - 5 Comparing Notations and Other Topics
  - 6 Summary and Conclusion

## Comparing Notations

- It is useful for you to go over the full SQL Power Architect implementation to see the different ways the binary many-to-one mappings are represented
  - » We have covered that earlier, but it's good to practice to do that

# MySQL Workbench

- A system tailored to MySQL with the capabilities to
  - » Design a database (works for some other DBMSs)
  - » Forward engineer a diagram/design into SQL DDL (we will see later what SQL DDL looks like more precisely)
  - » Reverse engineer a MySQL database to obtain a diagram/design
- A fragment of our database in MySQL Workbench (similarly, in SQL Power Architect)



- Next: The fragment's forward-engineered SQL DDL

# Forwarded-Engineered Database

```
-- MySQL Workbench Forward Engineering
```

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
```

```
-- Schema mydb
```

```
-- Schema mydb
```

```
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;
```

# Forwarded-Engineered Database

```
-- -----  
-- Table `mydb`.`Person`  
-- -----  
  
CREATE TABLE IF NOT EXISTS `mydb`.`Person` (  
  `ID#` VARCHAR(45) NOT NULL,  
  `SS#` VARCHAR(45) NOT NULL,  
  `FN` VARCHAR(45) NULL,  
  `LN` VARCHAR(45) NOT NULL,  
  `DOB` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`ID#`),  
  UNIQUE INDEX `SS#_UNIQUE` (`SS#` ASC) VISIBLE)  
ENGINE = InnoDB;
```

# Forwarded-Engineered Database

```
-- -----  
-- Table `mydb`.`Child`  
-- -----  
  
CREATE TABLE IF NOT EXISTS `mydb`.`Child` (  
  `ID#` INT NOT NULL,  
  `ChildName` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`ID#`, `ChildName`),  
  CONSTRAINT `fk01`  
    FOREIGN KEY (`ID#`)  
    REFERENCES `mydb`.`Person` (`ID#`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```

# Forwarded-Engineered Database

```
-- -----  
-- Table `mydb`.`Child`  
-- -----  
  
CREATE TABLE IF NOT EXISTS `mydb`.`Child` (  
  `ChildName` VARCHAR(45) NOT NULL,  
  `Person_ID#` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`ChildName`, `Person_ID#`),  
  INDEX `fk_ChildName_Person_idx` (`Person_ID#` ASC) VISIBLE,  
  CONSTRAINT `fk_ChildName_Person`  
    FOREIGN KEY (`Person_ID#`)  
    REFERENCES `mydb`.`Person` (`ID#`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```

# Forwarded-Engineered Database

```
-- -----
-- Table `mydb`.`Car`  
-----  
  
CREATE TABLE IF NOT EXISTS `mydb`.`Car` (  
  `VIN` VARCHAR(45) NOT NULL,  
  `Color` VARCHAR(45) NOT NULL,  
  `Date` VARCHAR(45) NULL,  
  `Person_ID#` VARCHAR(45) NULL,  
  PRIMARY KEY (`VIN`),  
  INDEX `fk_Car_Person1_idx` (`Person_ID#` ASC) VISIBLE,  
  CONSTRAINT `fk_Car_Person1`  
    FOREIGN KEY (`Person_ID#`)  
    REFERENCES `mydb`.`Person` (`ID#`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```

# Forwarded-Engineered Database

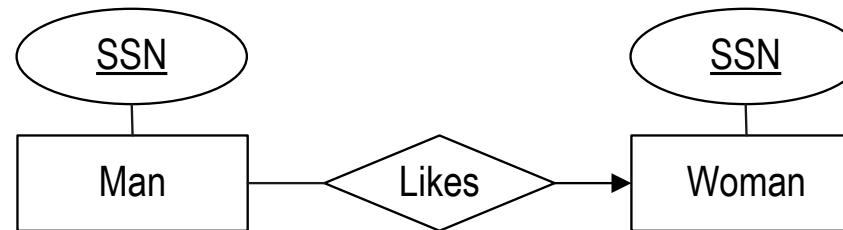
```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

## Sometimes It Is Necessary To Rename Attributes

- In practice, you may have constraints/policies on the permitted names of attributes
- So, you may want, in practice, to change the names of the attributes from what they appear as in an ER diagram
- We will not do it here, and please do not do that in your work as it will just cause more work for you and us
- But sometimes we **must** rename attributes, as we will see next

# Sometimes It Is Necessary To Rename Attributes

- Consider the ER diagram



- Likes is binary many-to-one, so we store it inside Man
- As we already have attribute SSN in Man (Man's SSN), woman's attribute inside Man must be something different than SSN
- So, we can have

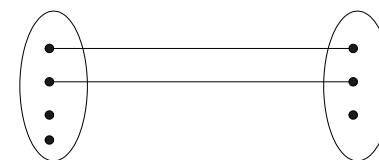


# We Have Seen This Before

- 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)
- In other words, R is one to one, if and only if
  - » R is many to one from A to B, and
  - » R is many to one from B to A



# ISA (We Saw This Before)

- ***Student is a subset of Person*** and inherits the primary key of Person, NetID, as ***both***
  - » Its own primary key, and
  - » As a foreign key referencing Person through the NetID attribute in Person
- There is no need to do anything else

Person	<u>NetID</u>	B	Student	<u>NetID</u>	D
	g	2		g	27
	i	56		h	6
	h	2			



## Likes: One-To-One Case

- From Man we can reach 0 or 1 Women
- From Woman we can reach 0 or 1 Men



- This is a one-to-one relationship (one-to-one partial function)
  - » We can say: a partial function from Man to Woman
  - » We can say: a partial function from Woman to Man
- We discussed this case in the context of focusing only on the “arrowheads” in an uncluttered drawing
- But to actually implement this we need to do more

# Instance

- ***We have two entity sets, neither a subset of the other one***
- We have one relationship: Likes
- We have one sets of edges (ignore possible arrow heads)
- An instance tells us who Likes whom
- When we start with an individual Man, we reach at most one Woman
- When we start with an individual Woman, we reach at most one Man

Man	<u>ManID</u>	Name	Woman	<u>WomanID</u>	Name
	g			K	
	h			L	
	i			M	
				N	

# Instance

- The instances below violate the one-to-one definition
- These are really two binary many-to-one relationships and not a single one-to-one relationship
- But our relationship was one-to-one

Man	<u>ManID</u>	Woman	<u>WomanID</u>
	g		K
	h		L
	i		M
			N

Man	<u>ManID</u>	Woman	<u>WomanID</u>
	g		K
	h		L
	i		M
			N

# Relational Implementations

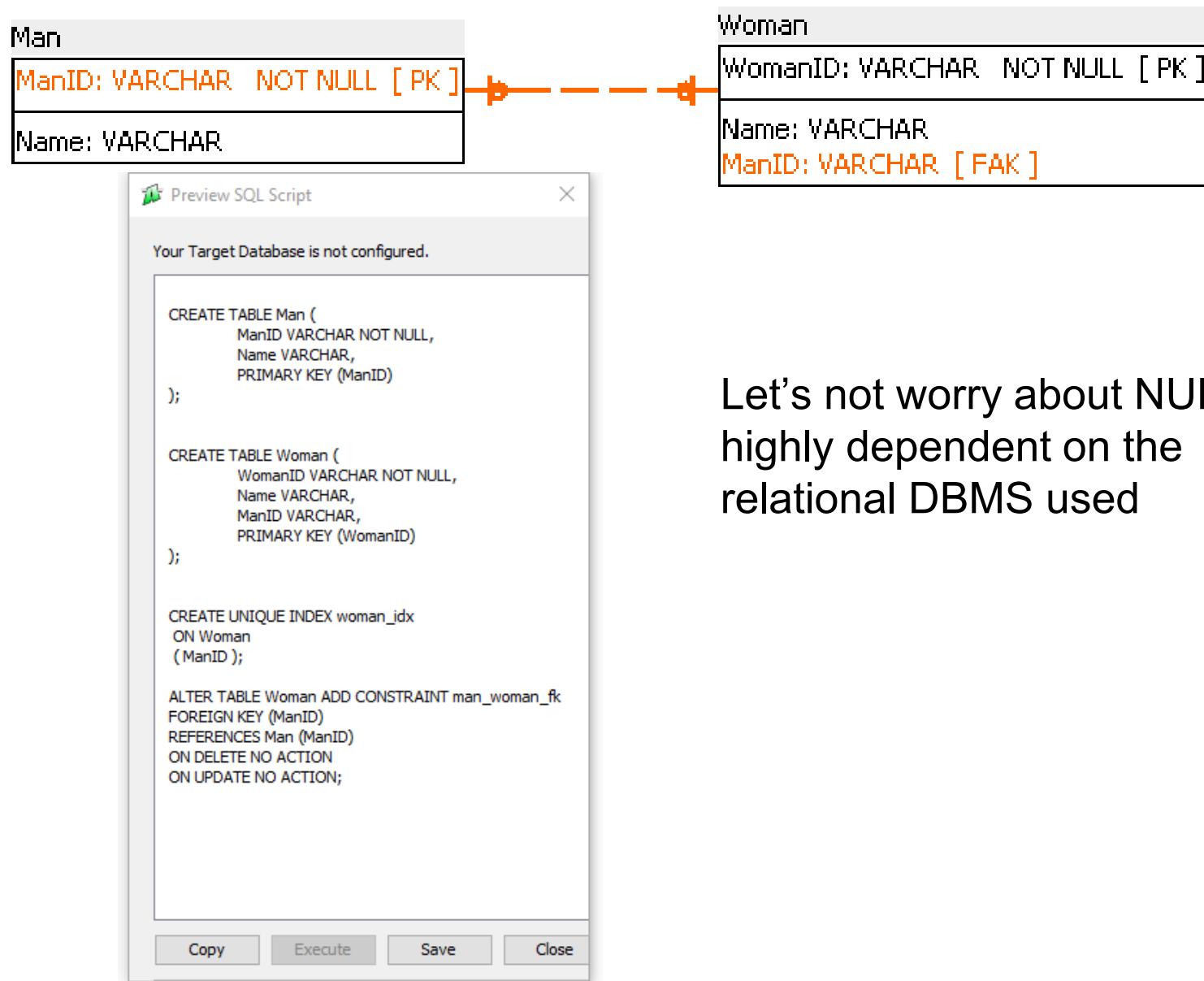
Man	<u>ManID</u>	WomanID	Woman	<u>WomanID</u>
	g	K		K
	h	L		L
	i			M N

- Man.WomanID REFERENCES Woman.WomanID
- Man.WomanID is UNIQUE

Man	<u>ManID</u>	Woman	<u>WomanID</u>	<u>ManID</u>
	g		K	g
	h		L	h
	i		M	
			N	

- Woman.ManID REFERENCES Man.ManID
- Woman.ManID is UNIQUE

# SQL Power Architect Implementation For the Second Alternative



Let's not worry about NULLs;  
highly dependent on the  
relational DBMS used

# SQL Power Architect Implementation

```
CREATE TABLE Man (
    ManID VARCHAR NOT NULL,
    Name VARCHAR,
    PRIMARY KEY (ManID)
);
```

```
CREATE TABLE Woman (
    WomanID VARCHAR NOT NULL,
    Name VARCHAR,
    ManID VARCHAR,
    PRIMARY KEY (WomanID)
);
```

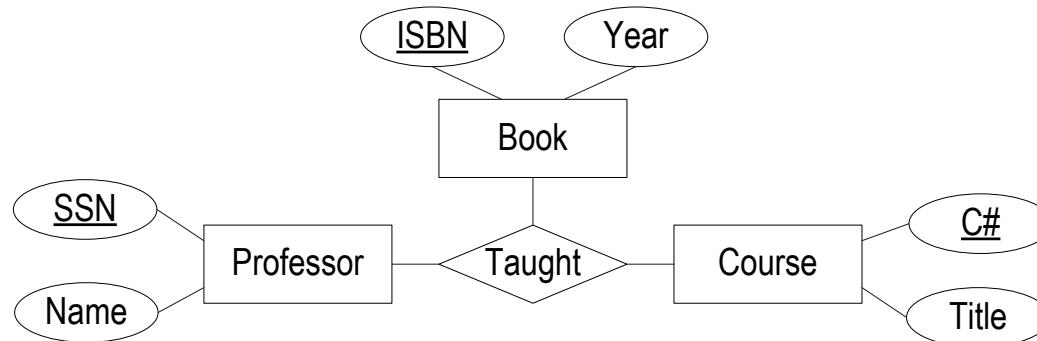
```
CREATE UNIQUE INDEX woman_idx
ON Woman
( ManID );
```

```
ALTER TABLE Woman ADD CONSTRAINT man_woman_fk
FOREIGN KEY (ManID)
REFERENCES Man (ManID)
ON DELETE NO ACTION
ON UPDATE NO ACTION;
```

## Non-Binary Relationships That Are Many-To-One

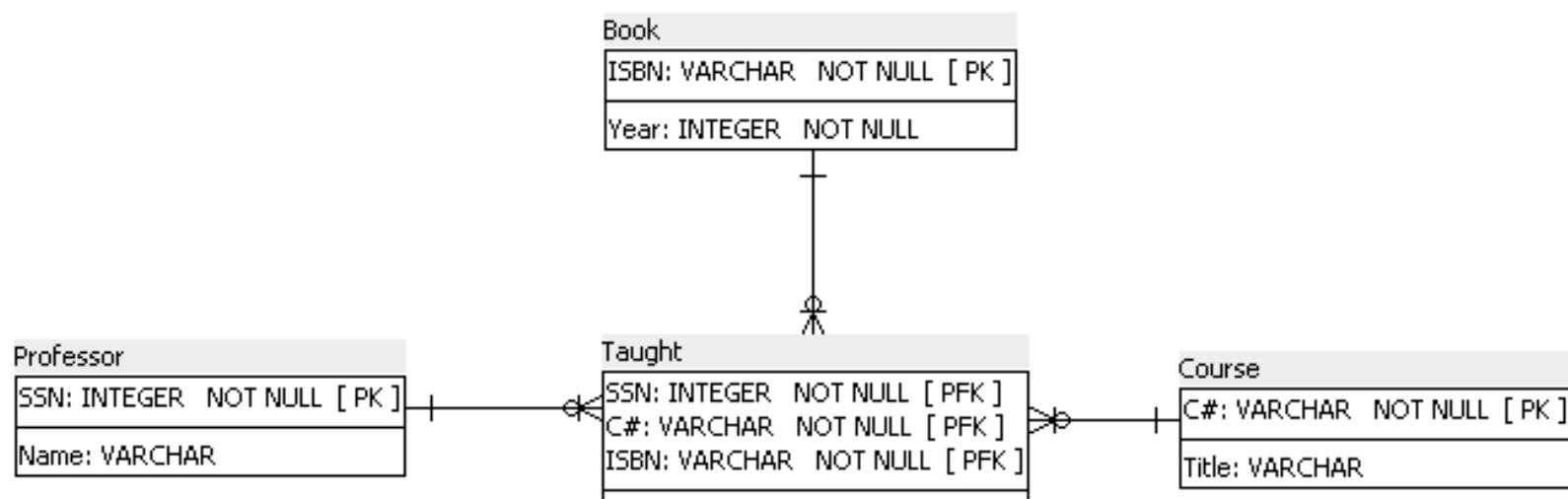
- We could not discuss this in the context of the University database, which was already quite complex
- We did discuss it earlier, but we will elaborate a little and learn something new
- We will discuss this now on several small examples
- So, issue will be how to select the primary key
  - » Again, we did it before
- Our example will be a tiny database, essentially “extracted” from the University database
- The example will deal with professors using books in courses

# Non-Binary Relationships That Are Many-To-One

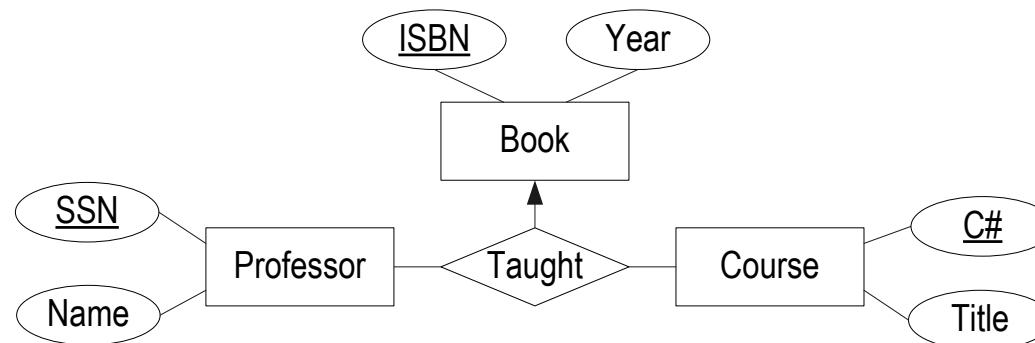


- No additional specifications

# Non-Binary Relationships That Are Many-To-One

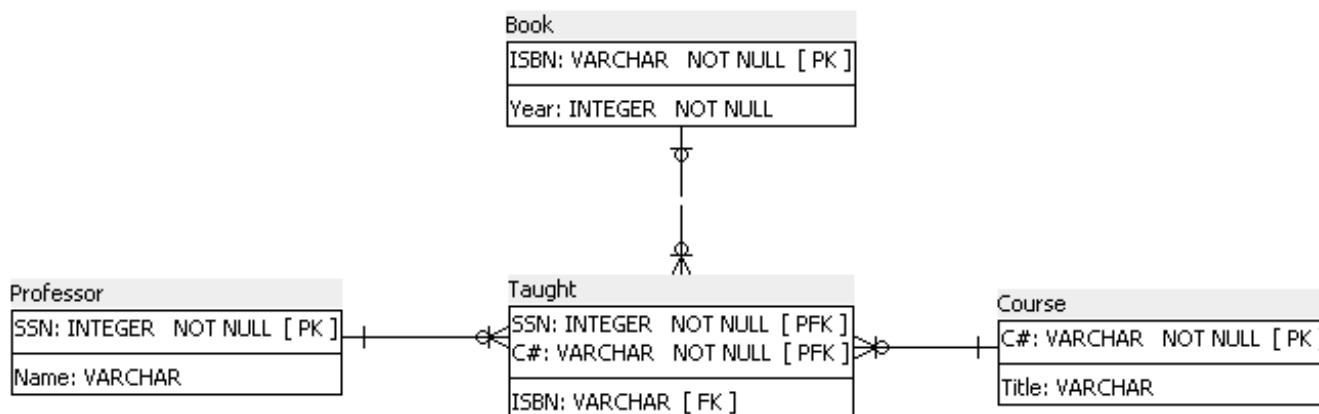


# Non-Binary Relationships That Are Many-To-One



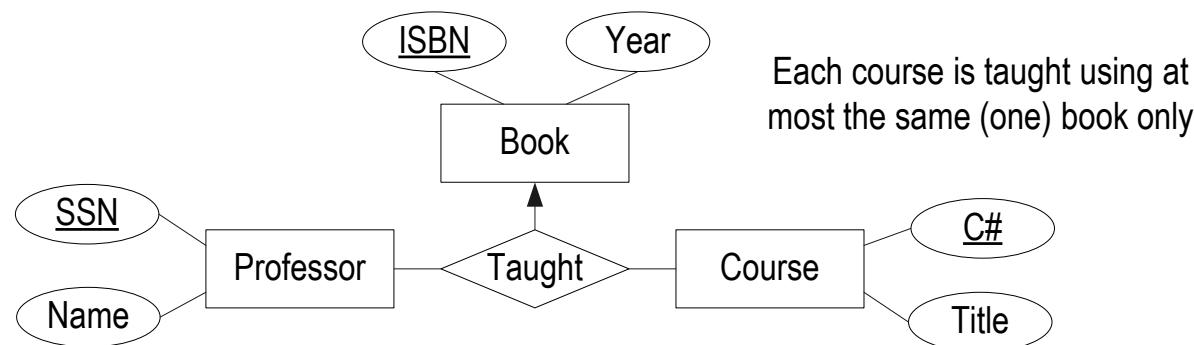
- No additional specification
- We understand this to mean that a professor taught a course using at most one book
- We understand this to mean: Book is a function of (Professor, Course)
  - » And this is actually a clearer phrasing
- But in a specific course each professor can use a different book
- But a specific professor can use a different book in each course or maybe no book at all

# Non-Binary Relationships That Are Many-To-One



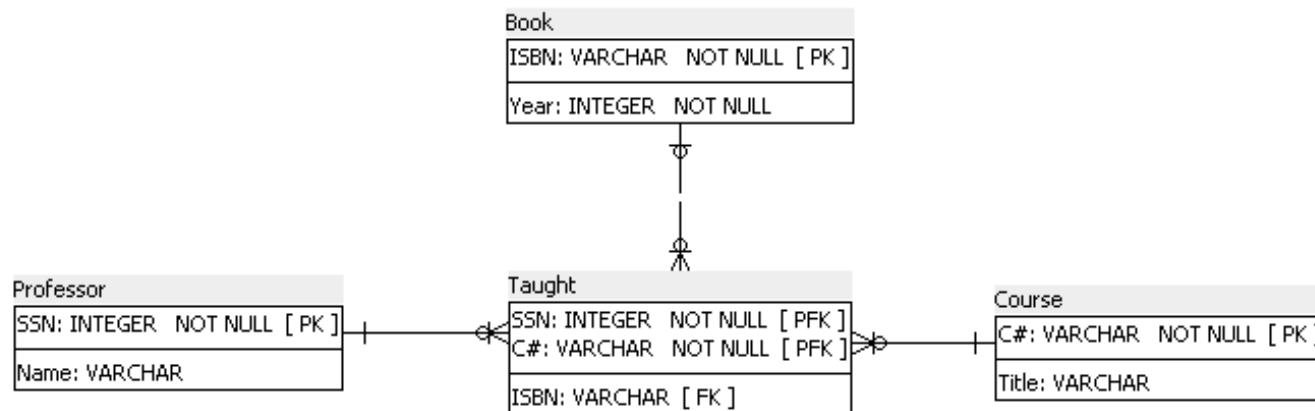
- Note that the primary key of Taught has only two attributes (“inherited” from Professor and Course)

# Non-Binary Relationships That Are Many-To-One



- We are told by an annotation that each course is taught using at most the same (one) book only, no matter which professor taught it
- We understand this to mean: Book is a function of Course
  - » And this actually a clearer phrasing but we cannot specify this using our ER diagrams conventions and therefore an annotation is needed (in the absence of more careful thinking)
- So, in such cases you need to
  - » Specify in a drawing using an arrow into the “target” entity set
  - » Add an annotation on the “source,” or “sources” entity set(s)

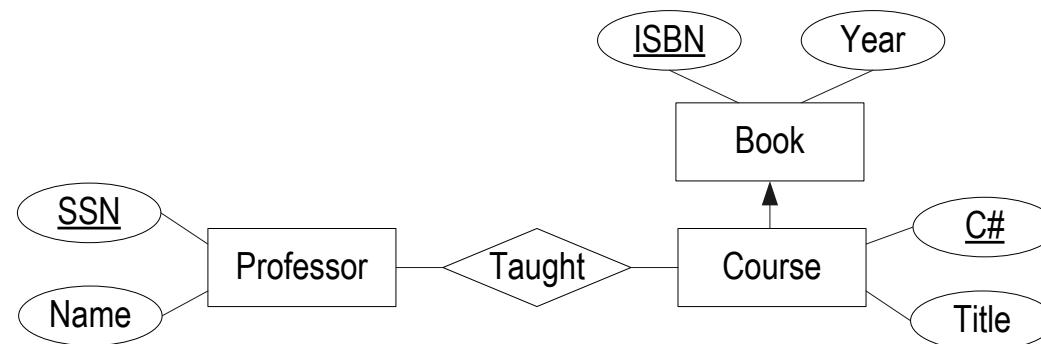
# Non-Binary Relationships That Are Many-To-One



- **We annotate that ISBN in Taught is only a function of C#**
- Incidentally, this is a bad design, which was derived from the ER diagram without further considerations
- In the Normalization unit we will formally understand why the design was bad and will learn algorithms for automatically fixing such bad designs

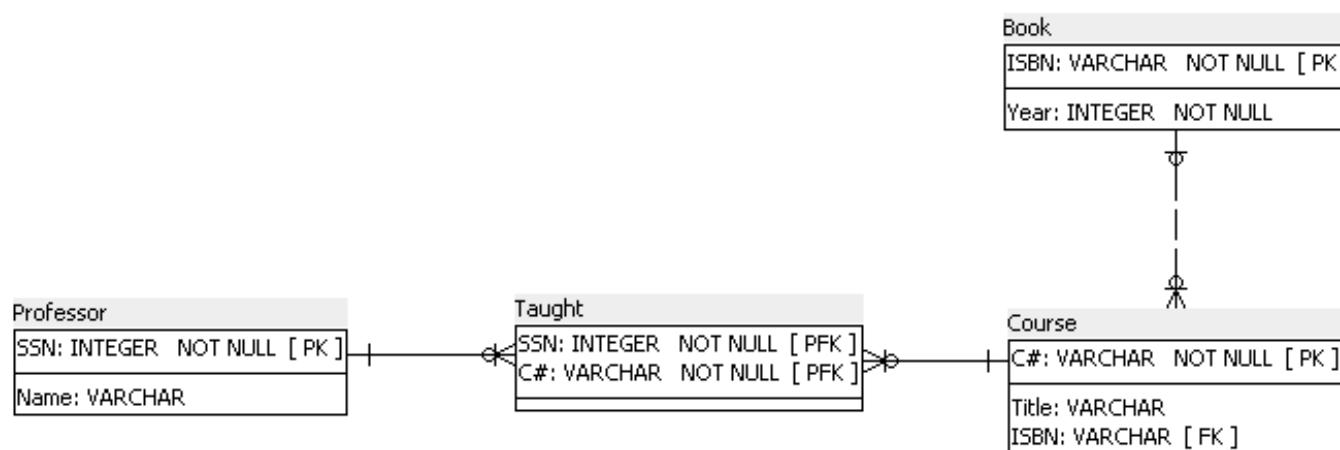
# Non-Binary Relationships That Are Many-To-One

- A much better ER diagram that expresses the constraint more naturally and without annotations
- But do not do such improvements (in this unit)



# Non-Binary Relationships That Are Many-To-One

- And the better ER diagram naturally transforms into a better relational implementation



## Referential Integrity: Example

Professor	ID#	Salary
	5	1
	7	2
	9	2

Taught	ID#	C#	Year	Semester	Sec#	Monitor
	5	G22.2433	2009	Spring	001	7

- Assume that we have some professors in table Professor, with rows: 5,1 and 7,2
- There is a row in Taught 5,G22.2433,2009,Spring,001,7
- This means that 5 teaches a specific section and 7 monitors this assignment

# Referential Integrity: Example

Professor	ID#	Salary
5	1	
7	2	
9	2	

Taught	ID#	C#	Year	Semester	Sec#	Monitors
5	G22.2433		2009	Spring	001	7

- A user accesses the database and **attempts to delete a row** (or all rows like this, recall that duplicates are permitted) 5,1 **from Professor**
- What should happen, as there is a row in Taught referencing this row in Professor?
- A user accesses the database and attempts to delete row 9,2 from Professor?
- What should happen, as there is no row in Taught referencing this row in Professor?

# Referential Integrity: Example

- Part of specification of foreign key in in Taught
  - » Covered again in the SQL DDL unit
- An action on Professor can be denied, or can trigger an action on Taught
- For example
  - » ON DELETE NO ACTION

This means that the “needed” row in Professor cannot be deleted  
Of course, it is possible to delete the row from Taught and then from the Professor (if no other row in in any table in the database “needs” that row in Professor)
  - » ON DELETE CASCADE

This means that if the a row is deleted from Professor, all the rows in Taught referring to it are deleted too
  - » ON DELETE SET NULL

This means, that the value referring to no-longer-existing professor is replaced by NULL  
In our example, this is not possible for ID# as it is a part of the primary key of Taught, but is possible for Monitor

## Referential Integrity: Another Example

- Part of specification of foreign key in Professor
- An action on Person can be denied, or can trigger an action on Professor
- For example
  - » ON UPDATE CASCADE

This means that if the value of ID# in Person is changed, this value of ID# also propagates to Professor
- Could (and probably should) add to Taught and Required:
  - » ON UPDATE CASCADE

In appropriate attributes, so that the change of ID# in Professor also propagates to them

In Taught in both ID# and Monitor

In Required in ID#
- Good mechanism for maintenance of consistency

- Frequently it is desirable that the primary keys not be interesting by themselves but serve as internal identifiers of tuples in a table
- It is then less likely that the real world will impose legal or practical constraints on their use
  - » For example, NYU used Social Security Numbers as identifiers, but a file of them was hacked
  - » Also, in future, maybe it will be illegal to use Social Security Numbers for any purpose other than Social Security taxes
  - » Maybe the manufacturer of Automobile will change retroactively the value of Model (name change to enhance sales)

# How To Think About Surrogate Keys

- You can think of them as just being row numbers
- We will meet them again when we talk about AUTOINCREMENT keys
- Note that surrogate keys “strip away” semantics
- We will denote the surrogate using “&” in our examples

# Designs With Natural And Surrogate Keys

## ▪ Natural (semantic) primary keys

Person	<u>FN</u>	<u>LN</u>	Height
a	P	165	
c	P	187	
a	Q		
d	R	173	

Likes	<u>FN</u>	<u>LN</u>	Name
a	P	SA	
c	P	US	
a	Q	SA	
c	P	IN	

Country	<u>Name</u>
CN	
IL	
SA	
US	
IN	

Person	<u>Person&amp;</u>	<u>FN</u>	<u>LN</u>	Height
1	a	P	165	
2	c	P	187	
3	a	Q		
4	d	R	173	

m

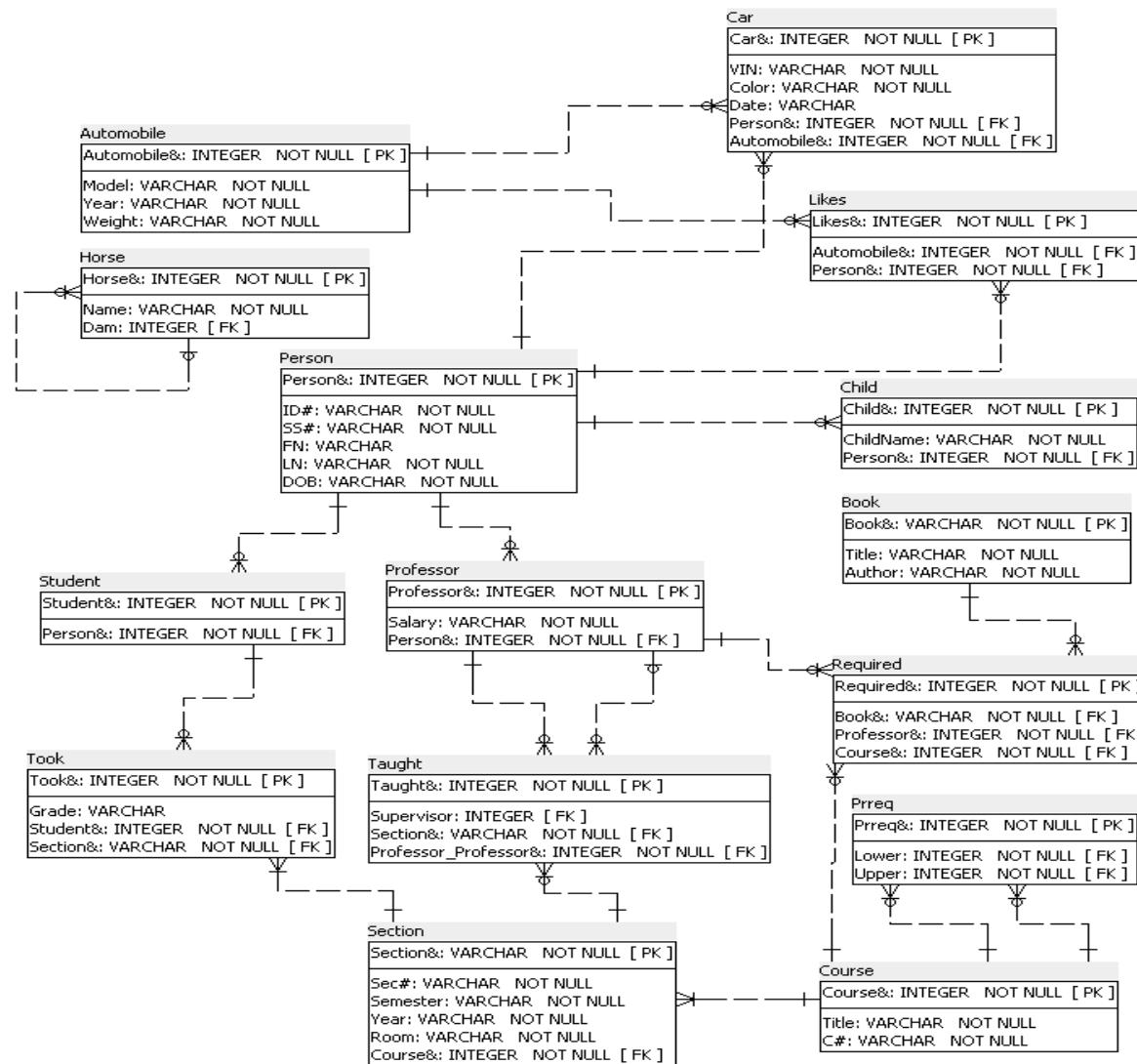
Likes	<u>Likes&amp;</u>	<u>Person&amp;</u>	<u>Country&amp;</u>
1	1	3	
2	2	4	
3	3	3	
4	2	5	

Country	<u>Country&amp;</u>	Name
1		CN
2		IL
3		SA
4		US
5		IN

## Surrogate Primary Keys Replacing Former “Semantic/Natural” Primary Keys

- We can redesign our application so that the primary keys are all surrogate and of one attribute only
- A very obvious way is to assign serial numbers to entities in an entity set (and in other places too)
  - » We discuss this in the unit dealing with SQL DDL
  - » And, very loosely speaking in that Unit, as an entity set becomes a table in a relational implementation with each entity in its own row, make the (surrogate) key of the 1<sup>st</sup> row to be 1, of the 2<sup>nd</sup> row to be 2, etc.
- There are advantages to indexing in the presence of surrogate keys
  - » We will discuss that in the Unit dealing with physical design
- Surrogate attributes are denoted in the following example by ending with “&”

# Our Application With Surrogate Primary Keys



## Surrogate Primary Keys And Storage Of Some Data

- Recall that in our previous example ID# was the primary key of Person
- It propagated into many tables, e.g., Taught
- What if it becomes illegal to store ID#?
  - » We are in big trouble: need to reorganize the database
- If we use surrogate primary keys, Person& is the attribute that propagates
- We can delete ID# without much pain

## Our Old Annotations

- Our “old” annotations, such as the bounds on the size of a Section are needed and important, but we will skip them here

# Important: Former Primary Keys Must Be Made UNIQUE and NOT NULL

- Note that some of the former primary keys are now surrogates
  - For example, former primary key of Professor (ID#) is now replaced by a surrogate (Person&)
- Note that some of the former primary key are now partially surrogates
  - For example, former primary key of Child (ID#,ChildName) is now replaced by (Person&,ChildName), which no longer is the actual primary key
- ***But we need to tell the database that former primary keys are UNIQUE, as, e.g., no two Person& could have the same ID# and the ID#'s cannot be NULL***

## Caveat

- We have gone through a pretty complex example and examined important cases of transforming an ER diagram into a relational implementation
- However, there is no complete set of “recipes” for doing this automatically
- Just do the best reasonable thing you can, and be ready to justify it (to yourself and to your customer)

## Annotate, Annotate, Annotate ...

- A relational schema specification should be annotated with all known constraints
- Annotations are needed so that whoever converts a relational schema specification + annotations into an SQL DDL + any added necessary constraints can account for all the constraints imposed on the application
- You do not need to put in annotations any constraints that are reflected in the schema specification, using a tool such as SQL Power Architect
- *In the assignments, you must not put such constraints (that is constraints implicit in what SQL Power Architect can express) in annotations*
- *Just to make sure that you realize which constraints are already reflected in an SQL Power Architect specification*

## Important: Former Primary Keys Must Be Made UNIQUE and NOT NULL

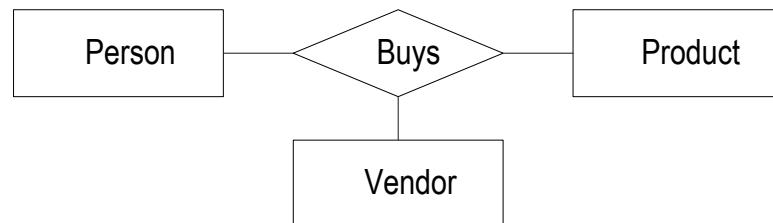
- Horse: UNIQUE(Name), Name NOT NULL
- Person: UNIQUE(ID#), ID# NOT NULL,  
UNIQUE(SS#), SS# NOT NULL
- Child: UNIQUE(Person&,ChildName), Person&  
NOT NULL, ChildName NOT NULL
- Automobile: UNIQUE(Model,Year), Model NOT  
NULL, Year NOT NULL
- Car: UNIQUE(VIN), VIN NOT NULL
- Book: UNIQUE(Author,Title), Author NOT NULL,  
Title NOT NULL
- Course: UNIQUE(C#), C# NOT NULL

## Important: Former Primary Keys Must Be Made UNIQUE and NOT NULL

- Prereq: UNIQUE(First,Second), First NOT NULL, Second NOT NULL
- Required: UNIQUE(Book&,Professor&,Course&), Book& NOT NULL, Professor& NOT NULL, Course& NOT NULL
- Section: UNIQUE(Year,Semester,Sec#,Course&), Year NOT NULL, Semester, NOT NULL, Sec# NOT NULL, Course& NOT NULL
- Took: UNIQUE(Section&,Student&), Section& NOT NULL, Student& NOT NULL
- Taught: UNIQUE(Professor&,Section&), Professor& NOT NULL, Section& NOT NULL

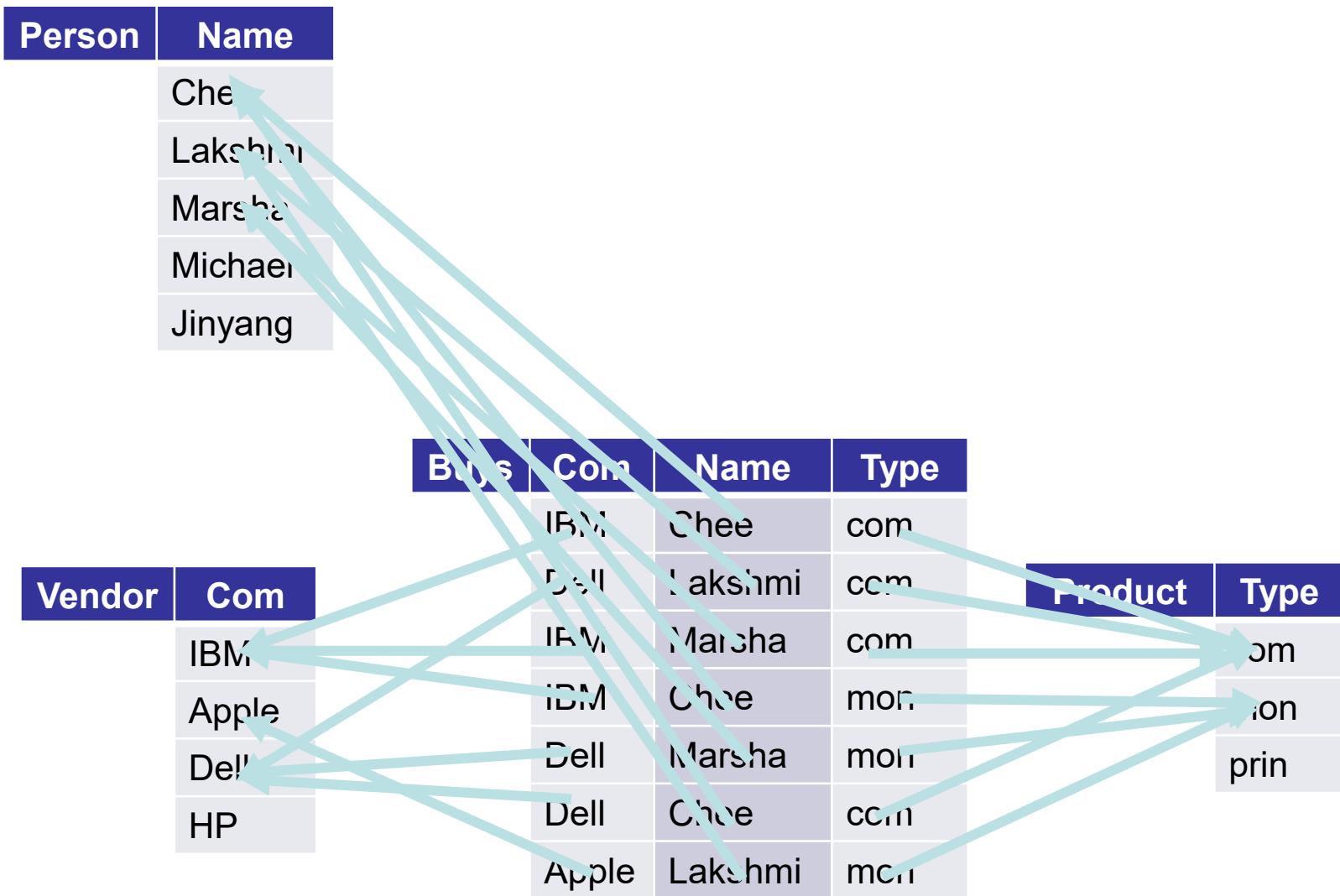
# Ternary Relationship (We Saw the ER Diagram Before)

- Let's look at Buys listing all tuples of  $(x,y,z)$  where Person  $x$  Buys Product  $y$  from Vendor  $z$
- We describe an instance 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



# Ternary Relationship Implementation

## Three Binary Many-To-One Mappings

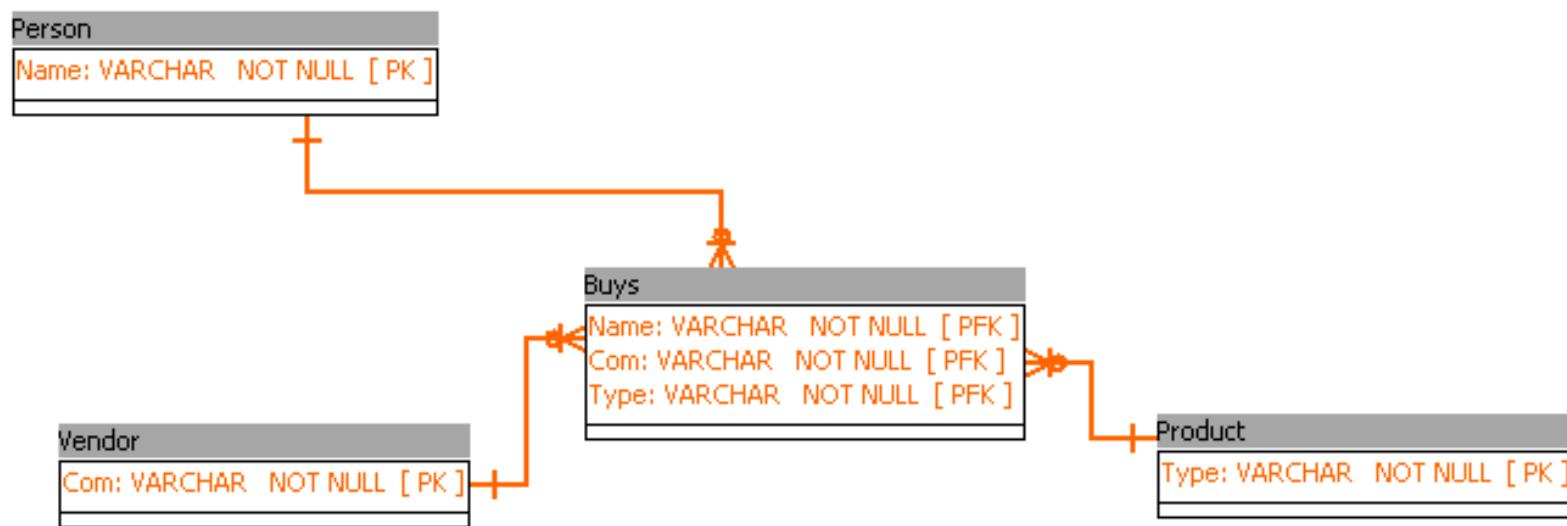


# Ternary Relationship Implementation

## Three Binary Many-To-One Mappings

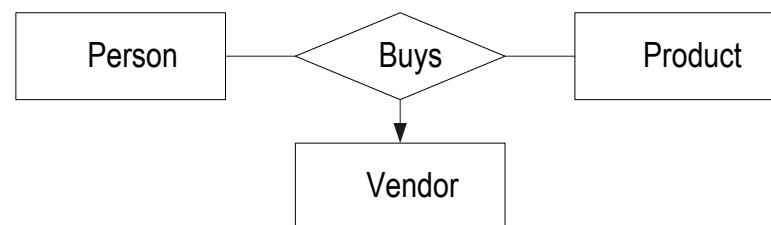
- We have 3 binary-many-to-one mappings
  - » Also referred to as partial functions, which happen to be total in the implementation
  - » Recall, “partial” means that the function does not have to be defined on all of its domain, but can be defined on all of its domain
- Our 3 mappings are
  - » From the rows of Buys into the rows of Person
  - » From the rows of Buys into the rows of Vendor
  - » From the rows of Buys into the rows of Product
- These mapping are specified using foreign keys
  - » We will see this again when we look at a relational implementation using SQL Power Architect
  - » Same thing using MySQL Workbench

# Ternary Relationship Implementation



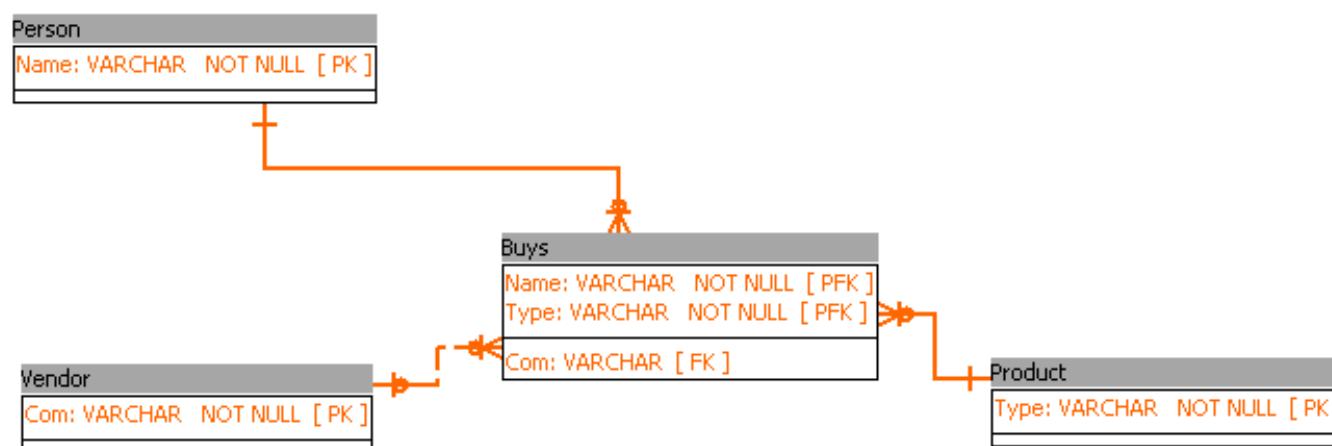
# Ternary Relationship Which Is a Function of Two Variables

- Let's look at Buys listing all tuples of  $(x,y,z)$  where person  $x$  Buys Product  $y$  from Vendor  $z$
- But now a Person Buys a specific Product only from one Vendor
- This simply means that Vendor is a partial function from the cartesian product Person  $\times$  Product
- For every pair (person, product) there is at most one vendor
  - » It is partial and not total, if for some pair (person, product) the vendor is not known
  - » Otherwise, it is guaranteed to be total



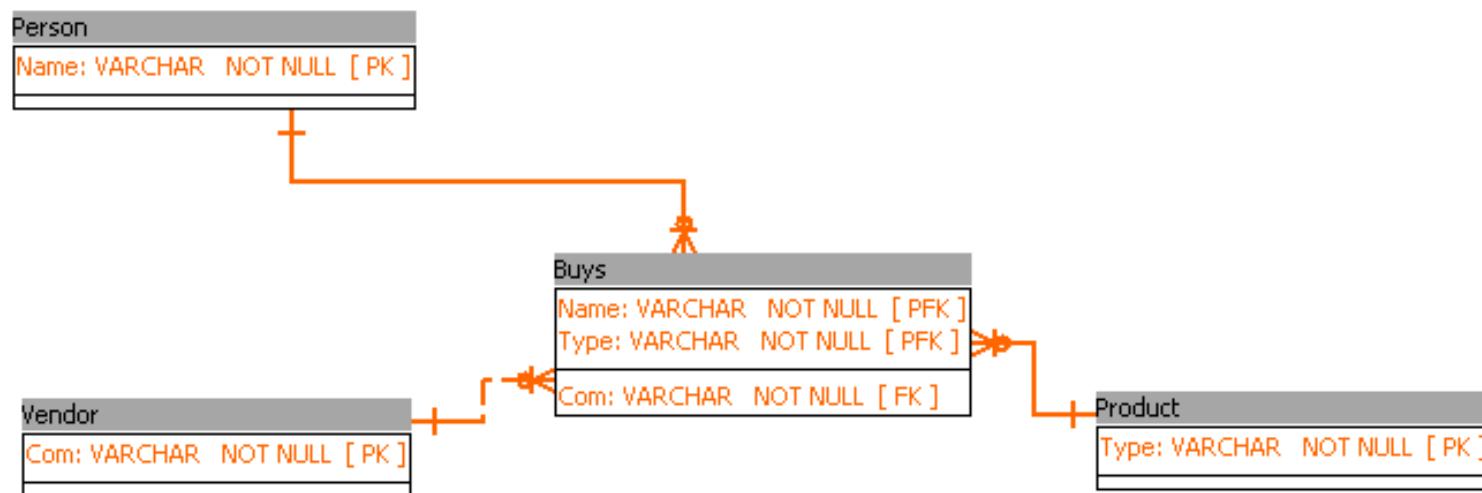
# Ternary Relationship Com (pany) May Not Be Always Known

- Note that Com (Vendor) is no longer a part of the primary key of Buys

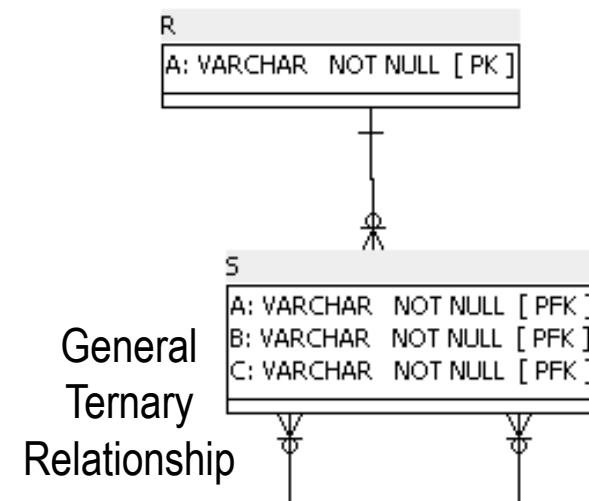


# Ternary Relationship Com (pany) Always Known

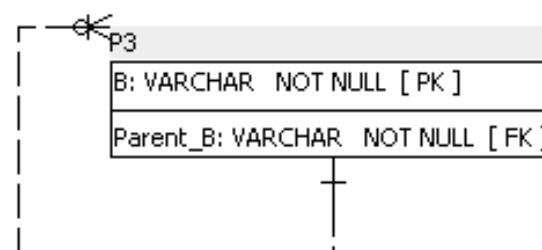
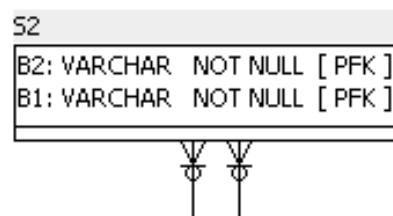
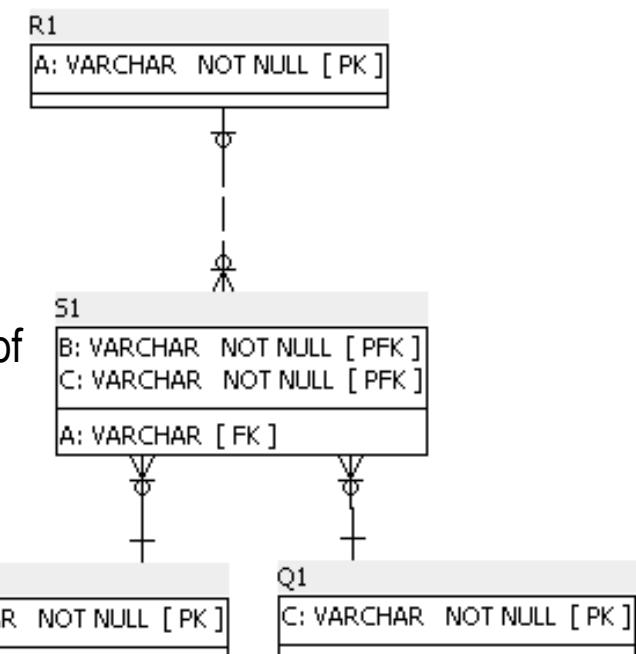
- Note that Com (Vendor) is no longer a part of the primary key of Buys



# Synopsis of Four Representative Cases



Partial Function of Two Variables



General Binary Relationship Between a Table and Itself

Partial Function From a Table to Itself

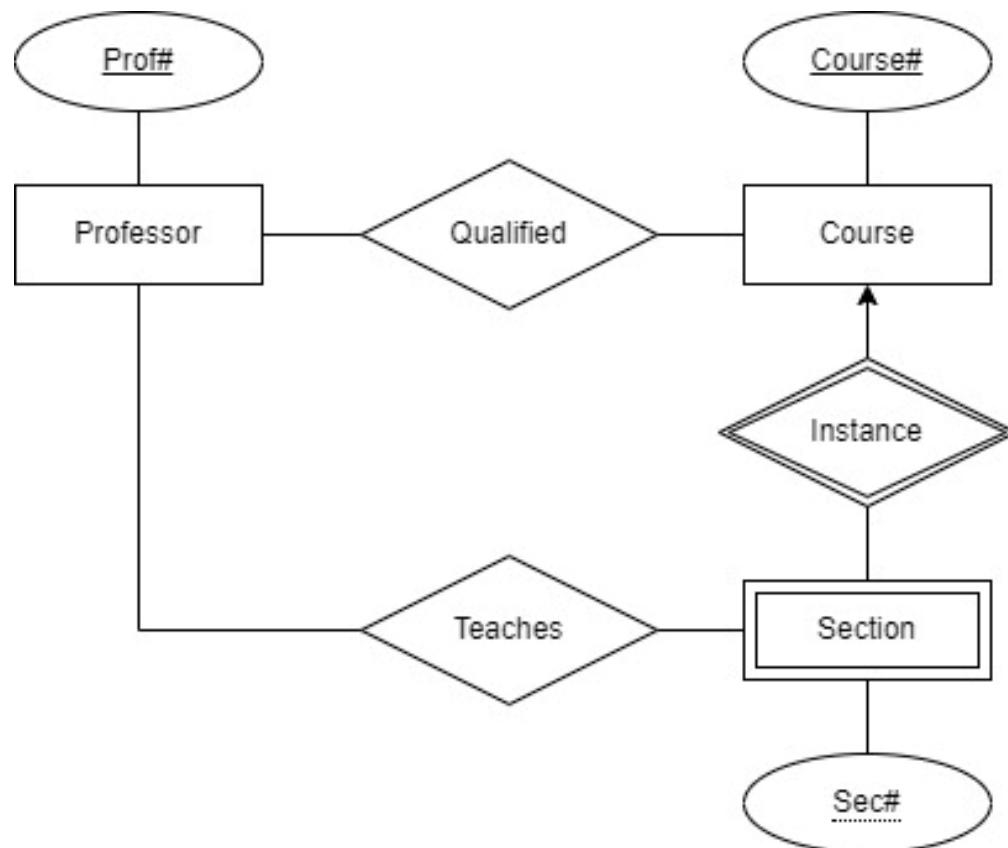
## Interesting Example of Semantic Constraint

- We will discuss an interesting example of a semantic constraint and its reflection in a relational database design
- It will also exemplify the power of the specification of foreign keys

## The Application

- We maintain information about which Professors are Qualified to Teach which Course
- We Select Professors to Teach Sections
- We want to Select a Professor to Teach a Section if it is Qualified to Teach the “related” Course

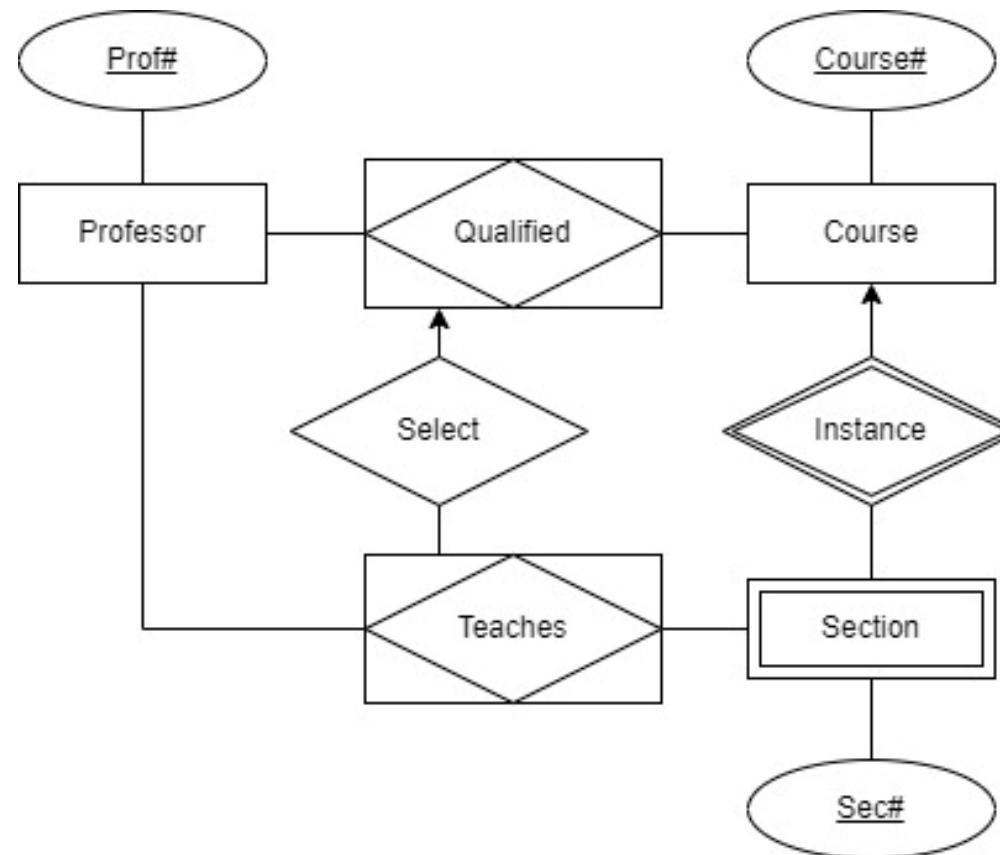
# ER Diagram



## Annotations

- If a Professor Teaches a Section, then that Professor is Qualified to Teach the Course (of which this is a Section)

# ER Diagram: A Little Better



## Annotations

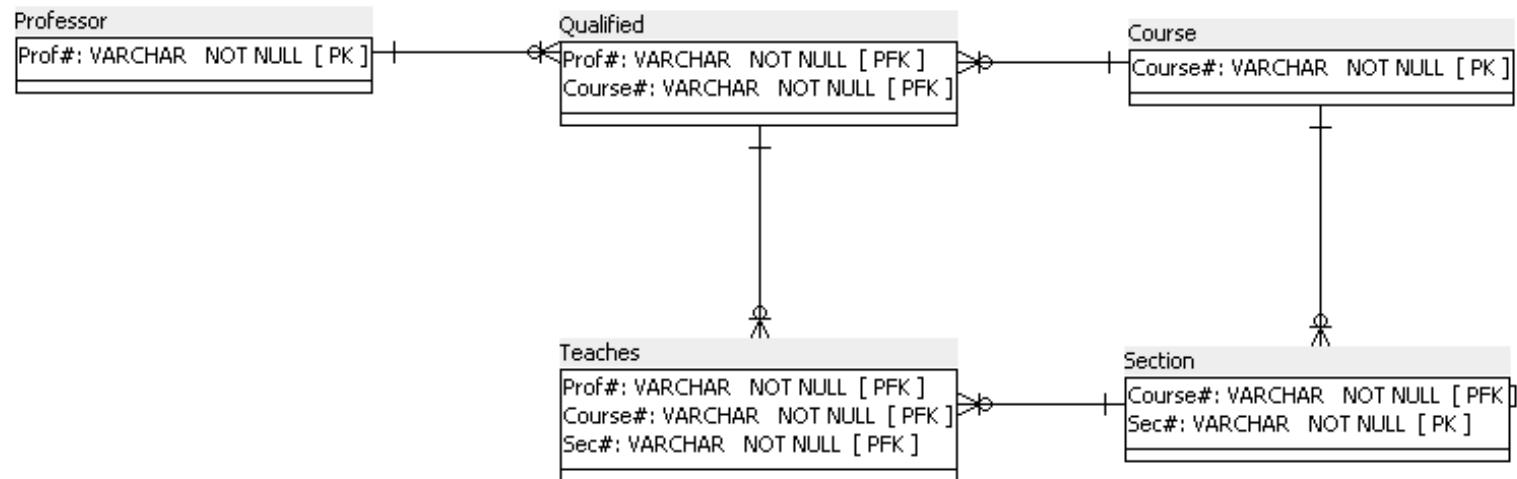
- If a Professor is Selected to Teach a Section, then that Professor is Qualified to Teach the Course (of which this is a Section)

- The drawing is not sufficient
- It only says that a Professor Teaching a Section may be qualified to teach a Course (which may be unrelated to the action Section that the Professor Teaches
- So, we must say (directly or indirectly) that
  - » The mapping is **total**
  - » It maps an entity in Teaches into the **relevant** entity in Qualified

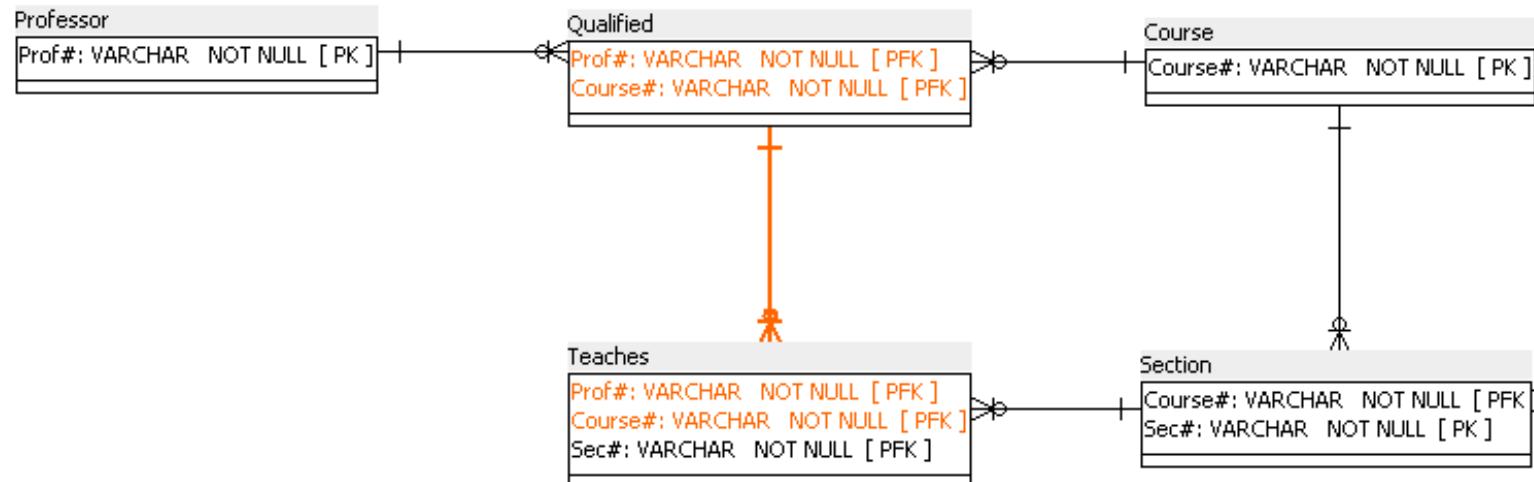
## Relational Implementation

- Let's think about a good design for the application

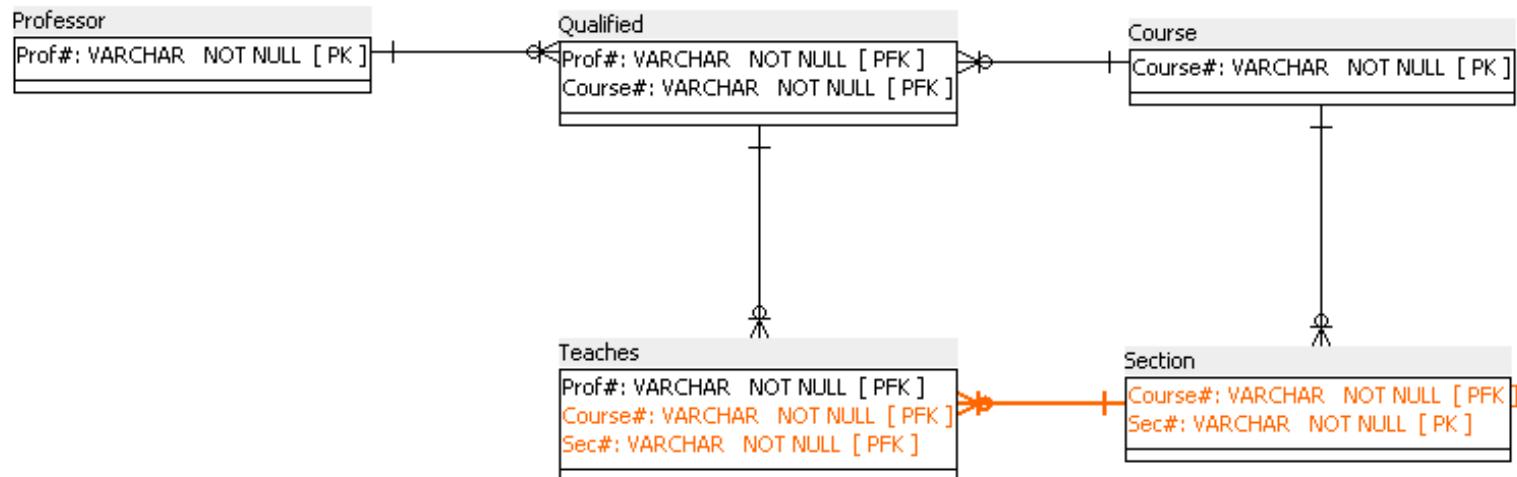
# Relational Implementation



# Relational Implementation With Highlighted Mapping



# Relational Implementation With Highlighted Mapping



## Comments

- Very clean design
- No annotations needed

# Agenda

1 Session Overview

2 ER and EER to Relational Mapping

3 Database Design Methodology and UML

4 Mapping Relational Design to ER/EER Case Study

5 Comparing Notations and Other Topics

6 Summary and Conclusion



# Key Ideas

- Overview
- Sets
- Relations and tables
- Relational schema
- Primary keys
- Implementing an ER diagram as a relational schema (relational database)
- General implementation of strong entities
- Handling attributes of different types
- General implementation of relationships
- Possible special implementation of binary many-to-one relationships
- Implementation of ISA
- Implementation of weak entities

# Key Ideas

- Foreign keys
- Primary key / foreign key constraints inducing many-to-one relationships between tables
- Concept of referential integrity
- SQL Power Architect
- Crow's feet notation: ends of lines
- Crow's feet notation: pattern of lines
- Non-binary many-to-one relationships
- Recursive relationships
- Surrogate keys

## Some Points to Remember for Assignments

- Below are a few points covered in the notes that are worth remembering while doing the homework
  - » They are phrased informally
- Do not introduce a table for a relationship if that can be avoided
  - » In our Animals example, do not introduce a table for Born
- Be careful with non-binary many-to-one relationships and pay attention which attributes should be in the primary key
  - » See our second Taught example
- Note our treatment of recursive relationships, though if you think about it, that was essentially a special case of what we have considered before
- Note how a one-to-one mapping is implemented using **UNIQUE**

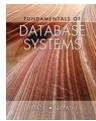
## Some Points to Remember for Assignments

- Specify in annotations all the constraints that cannot be specified by the SQL Power Architect model (which will be assigned)  
***and only such constraints***
- And refer to tables and columns and not to entities and attributes

- Basic ER model concepts of entities and their attributes
  - Different types of attributes
  - Structural constraints on relationships
- ER diagrams represent E-R schemas
- UML class diagrams relate to ER modeling concepts
- Enhanced ER or EER model
  - Extensions to ER model that improve its representational capabilities
  - Subclass and its superclass
  - Category or union type
- Notation and terminology of UML for representing specialization and generalization

# Assignments & Readings

- Readings
  - » Slides and Handouts posted on the course web site
  - » Textbook: Chapter 9
  
- Assignment #4
  - » Textbook exercises: 9.2, 9.3, 9.6, 9.7, 9.8, 9.9
  
- Project Framework Setup (ongoing)



# Next Session: Relational Algebra, Relational Calculus, and SQL



# Any Questions?

