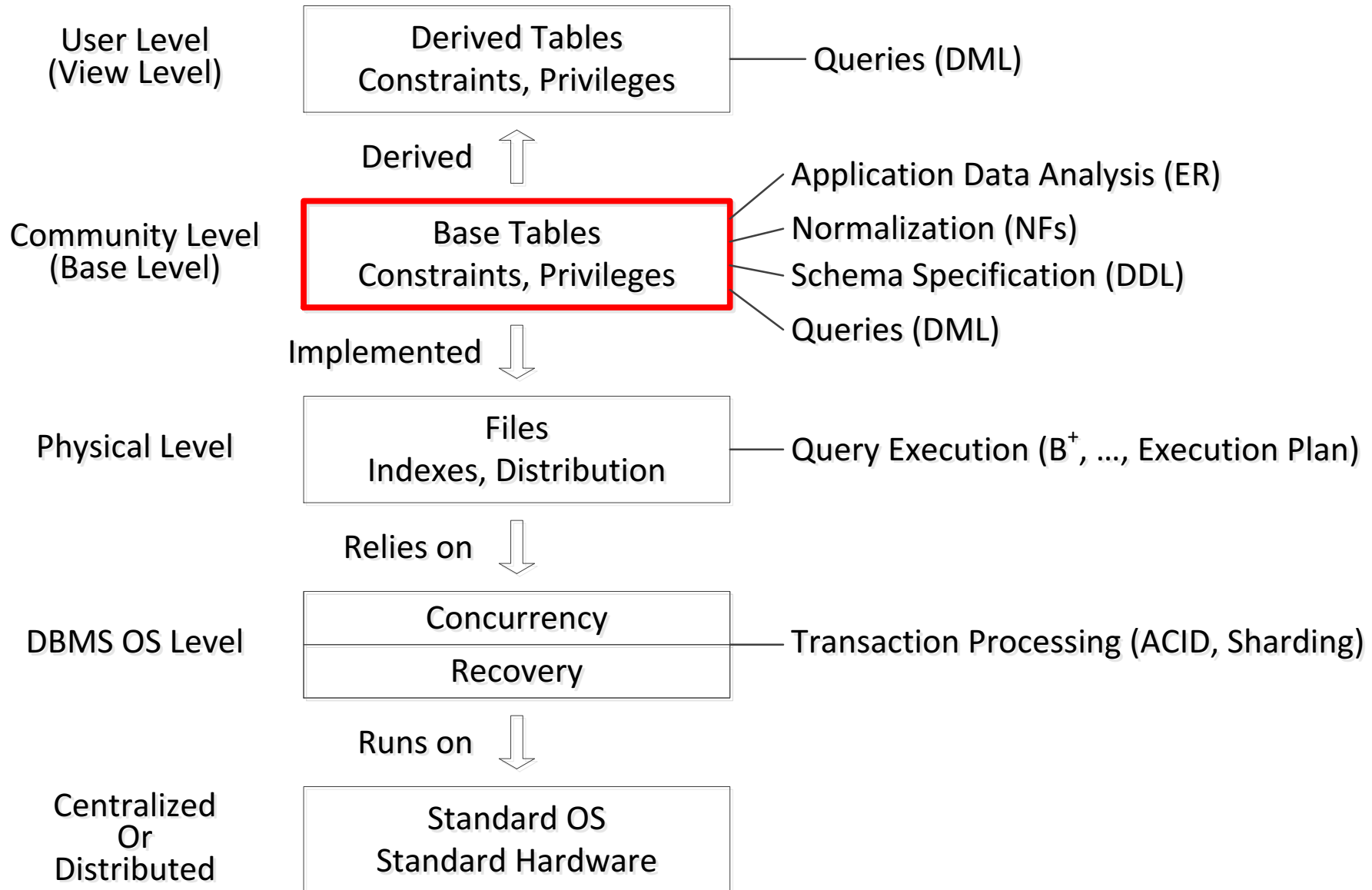


Unit 3
The Relational Model
And From ER Diagram To Relational Database

Relational Implementation in Context



Sets, Relations, and Tables

- ◆ In this unit, we learn the semantics of specifying a relational database, later we will learn the syntax of SQL for doing this
- ◆ The basic “data type”, 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***
- ◆ In SQL, such a variable is called a ***table***
- ◆ We may use the term table for a relation in this unit too
- ◆ Good review of basic concepts and terminology is available at http://en.wikipedia.org/wiki/Relational_databases

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 this, as we will see in a later unit

Sets

- ◆ Two sets A and B are equal iff (that is, if and only if) they have the same elements
- ◆ In other words, for every x : x is an element of A iff x is an element of B
- ◆ In still 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 none or at least once
- ◆ “More mathematically,”
 $A = B$ means $\forall x \{ x \in A \leftrightarrow x \in B \}$
- ◆ Therefore, as sets: $\{2, 5, 3, 7\} = \{2, 7, 5, 3, 5, 3, 3\}$
- ◆ 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**)
- ◆ 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 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 constraints exist!

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

- ◆ 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

- ◆ Since **the positions** in the tuple (1st, 2nd, 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

- ◆ To specify relations, it is enough to do what we have done above
- ◆ As long as we understand what are the domains for the columns, the following are formally fully specified relations (recall Unit 1)
 - 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

Relations

- ◆ But we will do more. We will specify, as suitable for the schema:
 - **Primary keys**
 - **Keys** (beyond primary)
 - **Foreign keys** and what they reference (we will see soon what this means)
 - Additional constraints
- ◆ Some of the constraints involve more than one relation
- ◆ The above are most important structurally
- ◆ Later, when we talk about SQL DDL, we will specify additional properties

Keys (and Superkeys)

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

Person	FN	LN	Grade	YOY
	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 that
- ◆ Then (FN, LN) is a **superkey** of Person, and in fact a **key**, because neither FN nor LN by themselves are sufficient (we are told that too)

Keys (and Superkeys)

◆ 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 and only any two tuples that are equal on the elements of these columns are (completely equal)
- ◆ **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**
- ◆ A relation always has at least one key (start with any superkey and remove unnecessary columns)
- ◆ There may be more than one key
- ◆ Exactly one key is chosen as **primary key** (there is no formal way to specify 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)

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 are 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 the columns does not do that**
- ◆ **Primary Key**: a chosen key

Keys (and Superkeys)

- ◆ We will underline the attributes of the chosen primary key
- ◆ Returning to Unit 02 and 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)
 - This implies that Country, State, Name form the primary key
 - We also have a candidate key: Longitude, Latitude

Relational Databases

- ◆ A ***relational database*** is “basically” a set of relations
- ◆ It is an instance of a relational schema

- ◆ This is what is formally correct, **but it is misleading in practice**

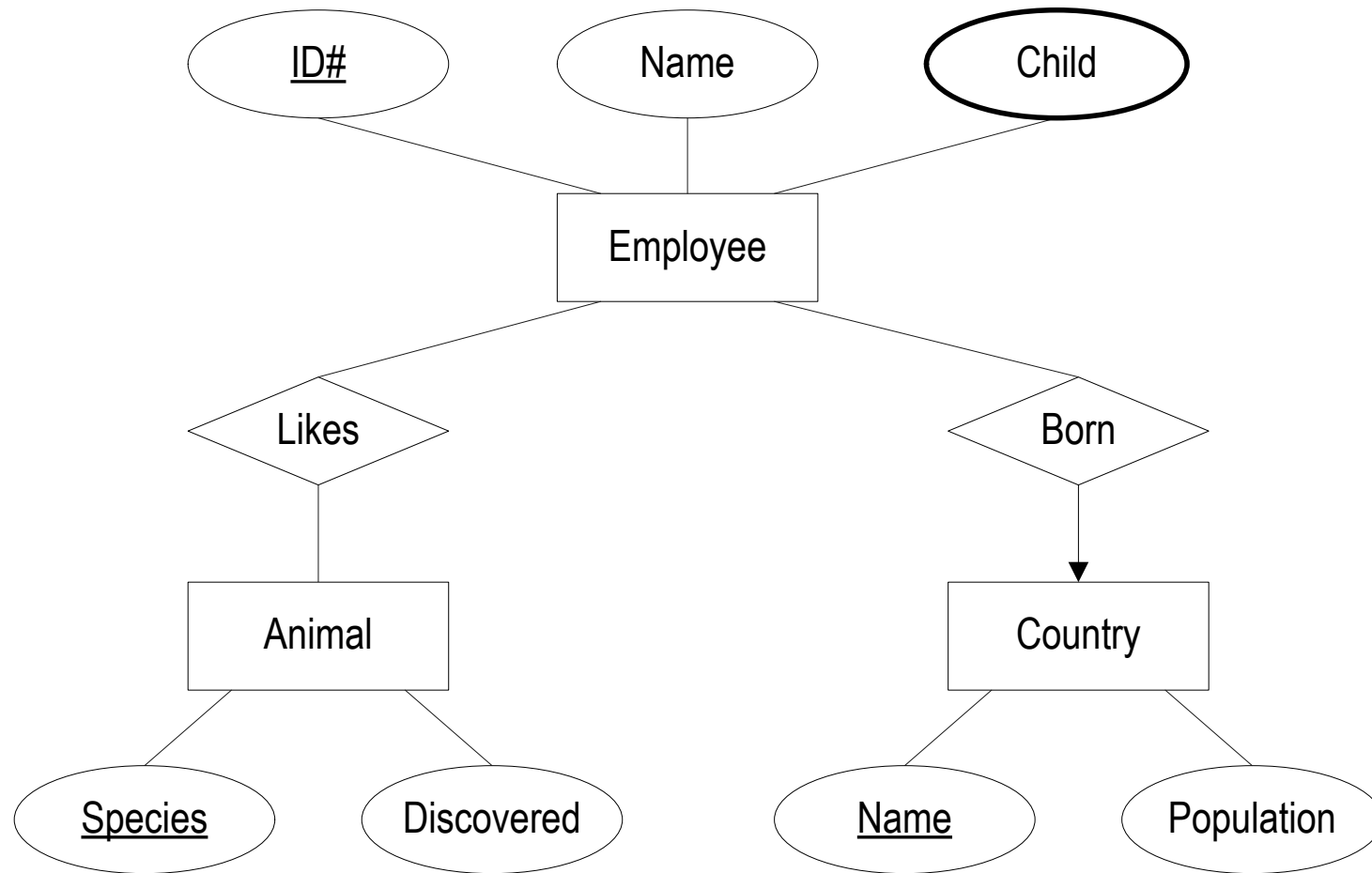
- ◆ **As we will see later, a relational database is**
 - **A set of relations**
 - **A set of binary, many-to-one mappings between them (partial functions)**

- ◆ We will know later what this means exactly, but I did not want to leave you with not quite a useful definition

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***
- ◆ We will first go through a simple example
- ◆ Then, we will go through our large example, studied previously
- ◆ Then, we look at some additional points of interest
- ◆ Finally, we summarize the process, so we are sure we understand it

Small ER Diagram



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 the primary key must not be missing** (this is a general rule, not only for this example)
 - 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

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 1150
- CN has 1330
- RU

◆ 4 Animals

- Horse in Asia
- Wolf in Asia
- Cat in Africa
- Yak in Asia
- Zebra in Africa

Country

- ◆ There are four countries, listing for them: Cname, Population (the latter only when known):
 - US
 - IN, 1150
 - CN, 1330
 - 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:

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

- ◆ Note that some “slots” are NULL, indicated by emptiness

Animal

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

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

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”
- ◆ 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 we have correct structures

Employee And Child

◆ Replace (incorrect)

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

By (correct)

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

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

Employee And Child With Better Column Names

◆ Replace (incorrect)

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

By (correct)

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

Child	<u>Parent</u>	<u>Child</u>
	1	Erica
	1	Frank
	2	Bob
	2	Frank
	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 Child’s Parent

- ◆ It is clear from the example how to handle any number of simple multivalued attributes an entity has
 - 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 a primary key for the main table and that multivalued attribute
Its primary key is the primary key of the entity combined with the 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
 - 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	<u>ID#</u>	Name
	1	Alice
	2	Bob
	4	Carol
	5	David
	6	Bob

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

Foreign Key \equiv A Binary Many-To-One Relationship Between Tables (Partial Function)

Employee	<u>ID#</u>	Name	Child	<u>ID#</u>	<u>Child</u>
	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 into it

In other words, no constraint

◆ Here the function is **total**: defined everywhere

Foreign Key \equiv A Binary Many-To-One Relationship Between Tables

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

- ◆ Another option
- ◆ Note 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 foreign key referencing Employee

Born

- ◆ Born needs to specify which employees were born in which countries (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** (actually was born in exactly one country but we may not know what it is)
- ◆ We have a **binary many-to-one relationship** between Employee and Country

Employee	<u>ID#</u>	Name	Country	<u>Cname</u>	Population
	1	Alice	→	US	
	2	Bob	→	IN	1150
	4	Carol	→	CN	1330
	5	David	→	RU	
	6	Bob	→		

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	1150
	CN	1330
	RU	

we have

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

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

Implementation For Born

- ◆ Augment Employee so instead of

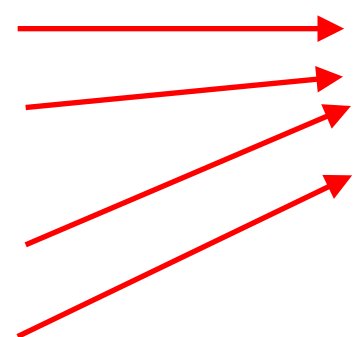
Employee	<u>ID#</u>	Name
	1	Alice
	2	Bob
	4	Carol
	5	David
	6	Bob

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

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

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

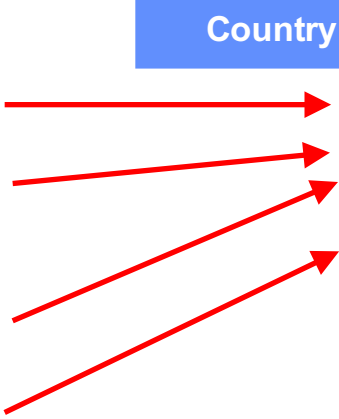
Country	<u>Cname</u>	Population
	US	
	IN	1150
	CN	1330
	RU	



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

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



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

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 about such, possibly convenient, renaming

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

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

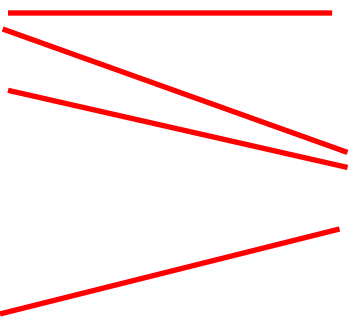
Likes

- ◆ 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	<u>ID#</u>	Name	Animal	<u>Species</u>	Discovered
	1	Alice	—	Horse	Asia
	2	Bob	—	Wolf	Asia
	4	Carol	—	Cat	Africa
	5	David	—	Yak	Asia
	6	Bob	—	Zebra	Africa



Likes (impossible implementation)

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

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

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

Animal	<u>Species</u>	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	<u>ID#</u>	<u>Species</u>
	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	<u>ID#</u>	Name	Likes	<u>ID#</u>	<u>Species</u>	Animal	<u>Species</u>	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 relationships between three tables: Employee, Likes, and Animal. The Employee table has columns Employee, ID#, and Name. The Likes table has columns Likes, ID#, and Species. The Animal table has columns Animal, Species, and Discovered. Red arrows indicate foreign key constraints: from Likes.ID# to Employee.ID# and from Likes.Species to Animal.Species. Additionally, red arrows show many-to-one mappings: from Likes.ID# to Employee.ID# (1 to 1), from Likes.Species to Animal.Species (1 to 1), and from Likes.ID# to Animal.Species (1 to 1).

Using Visio

- ◆ We will use Visio for 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=55TpWp4TmMw&NR=1>
 - <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
 - They 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

- ◆ We will use Visio to specify our relational implementation
- ◆ And in fact, we could even use software to generate database specifications from the diagram to SQL DDL
- ◆ We will just focus for now on the first task

Specifying A Relational Implementation Using Visio 2010

- ◆ A drawing in Visio is not an Entity Relationship Diagram tool despite such terminology in Visio
- ◆ This is good, as **it produces a relational schema**, which is what we actually need, but this is a lower-level construct
- ◆ **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

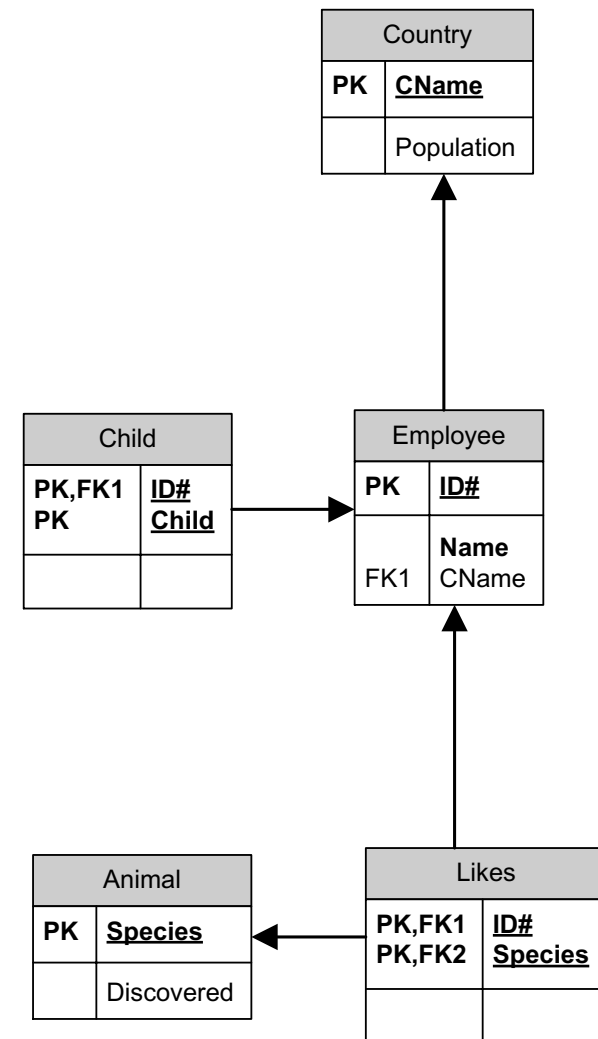
Country	<u>CName</u>	Population
	US	
	IN	1150
	CN	1330
	RU	

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

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

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

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



Cardinality Constraints

- ◆ The statement that a relationship is many-to-one as opposed to being a “standard” many-to-many relationship is really a cardinality constraint
- ◆ We will look at relationship 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 From Unit 2)



Every Person likes 0 or more Countries



Every Person likes 1 or more Countries

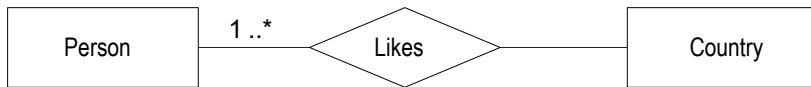
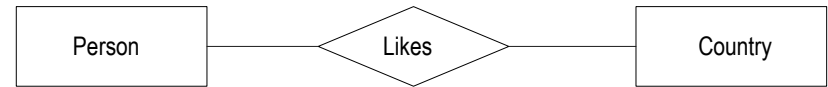
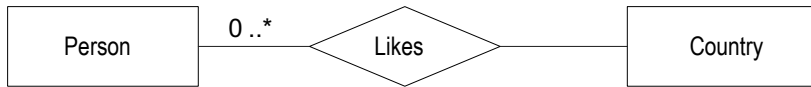


Every Person likes 0 or 1 Countries



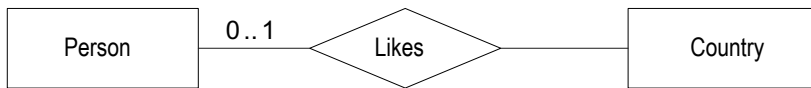
Every Person likes 1 Country

Arrow Notation Cannot Distinguish Some Cases



Crow's Feet: Improved Arrow Notation

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



Crow's Feet

- ◆ 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 can use the Crow's Feet notation

Relational Implementation For The Example

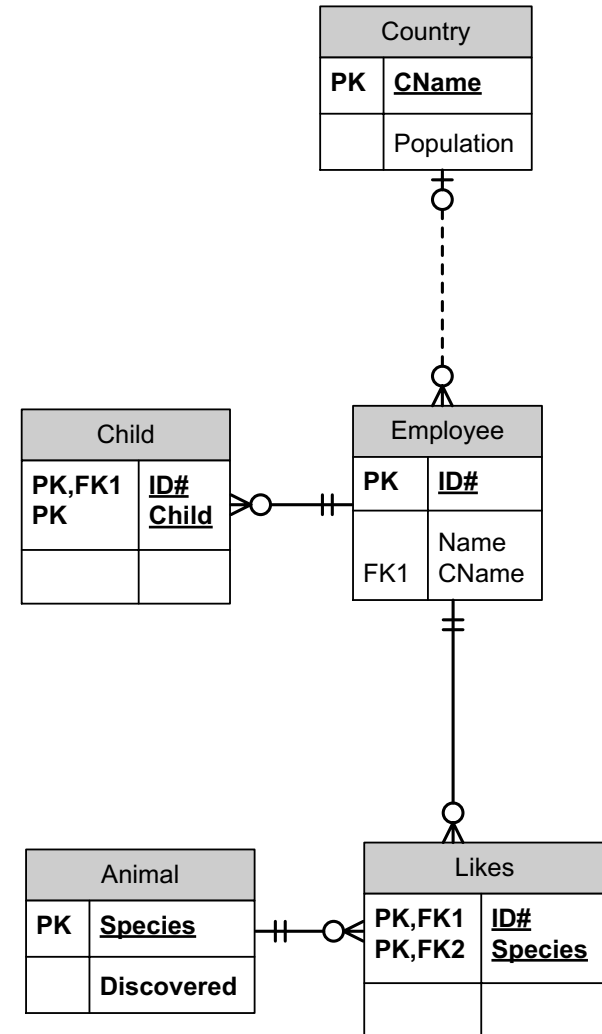
Country	<u>CName</u>	Population
	US	
	IN	1150
	CN	1330
	RU	

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

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

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

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

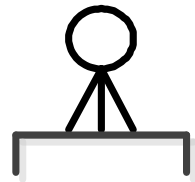


End Of Lines In Crow's Feet Notation

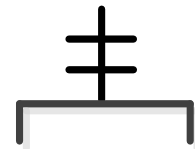
◆ 0..1



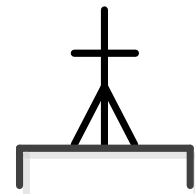
◆ 0..*



◆ 1..1

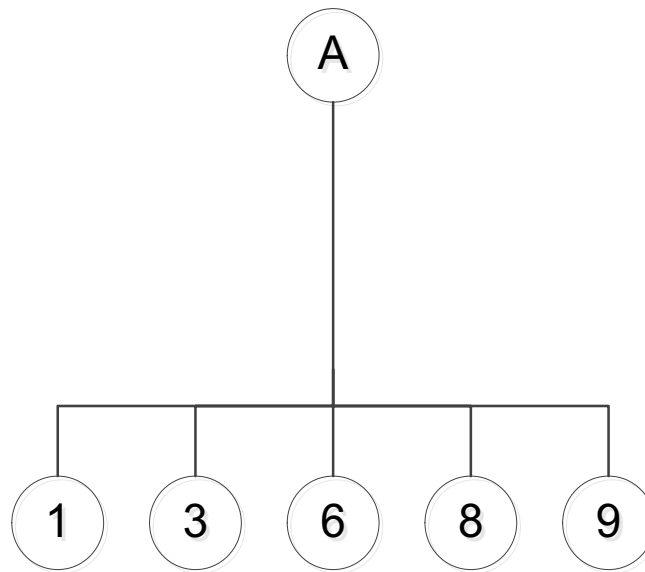


◆ 1..*



Intuition For The Notation: One-Level Tree

One Side



Many Side

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 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 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 Employee “can exist” without Country

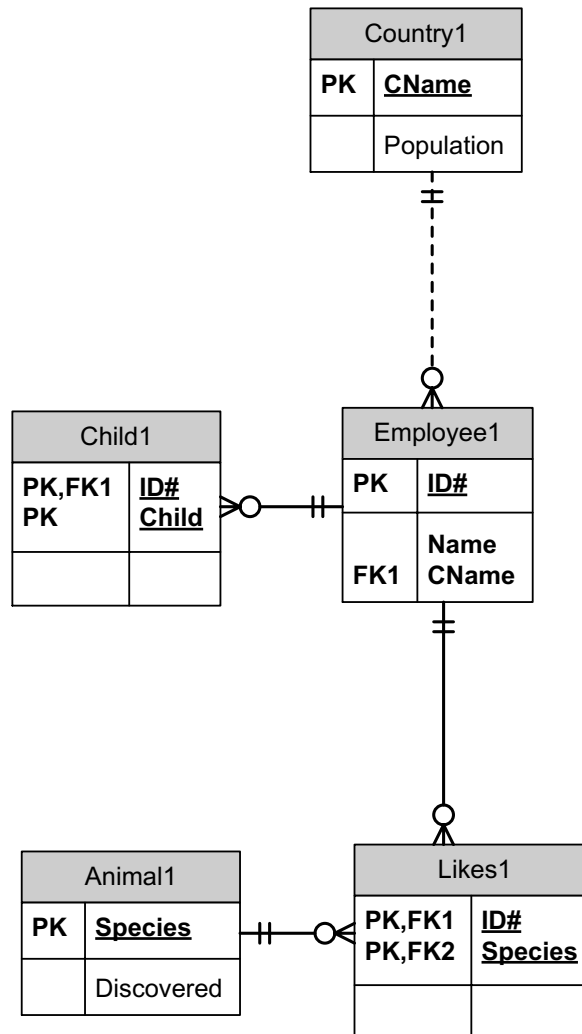
Pattern Of Lines

- ◆ *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

- ◆ 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

Employee	<u>ID#</u>	Name	Born	<u>ID#</u>	Cname	Country	<u>CName</u>	Population
	1	Alice		1	US		US	
	2	Bob		2	IN		IN	1150
	4	Carol		5	IN		CN	1330
	5	David		6	CN		RU	
	6	Bob						

Alternative Implementation For The Example

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

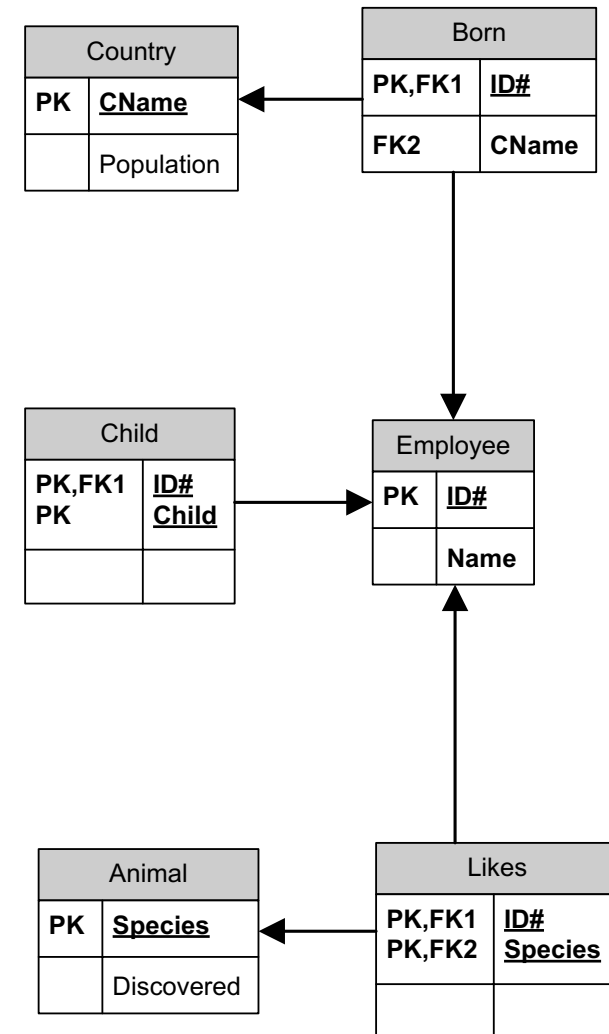
Born	<u>ID#</u>	CName
	1	US
	2	IN
	5	IN
	6	CN

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

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

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

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



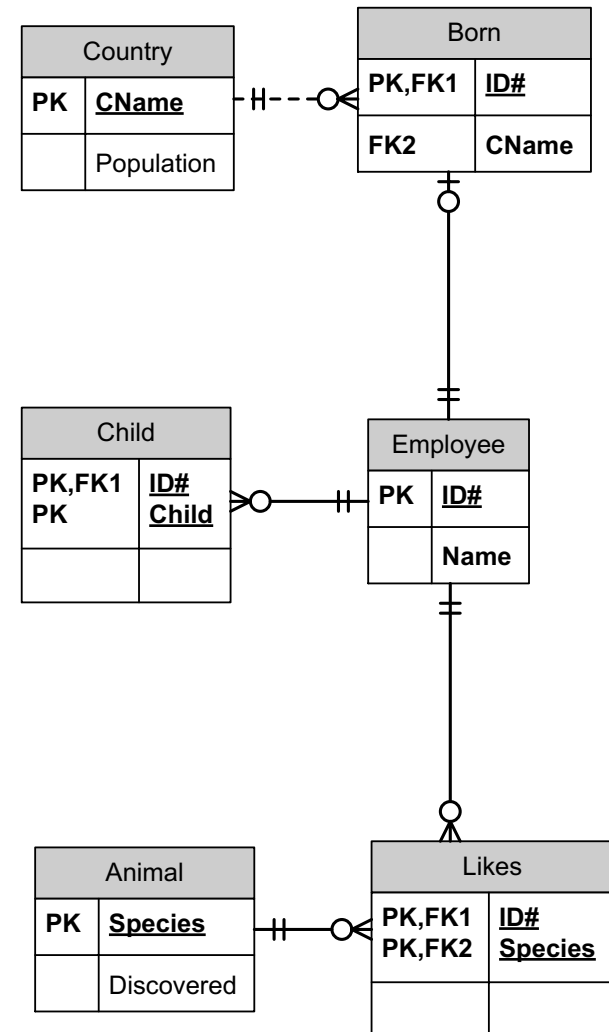
Alternative Implementation For The Example

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

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

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

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



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

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

Options For Relationships

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?

- ◆ 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 implementation, to have better exercises***

- ◆ ***So do this for all the homeworks and tests, when relevant***

To Remember!

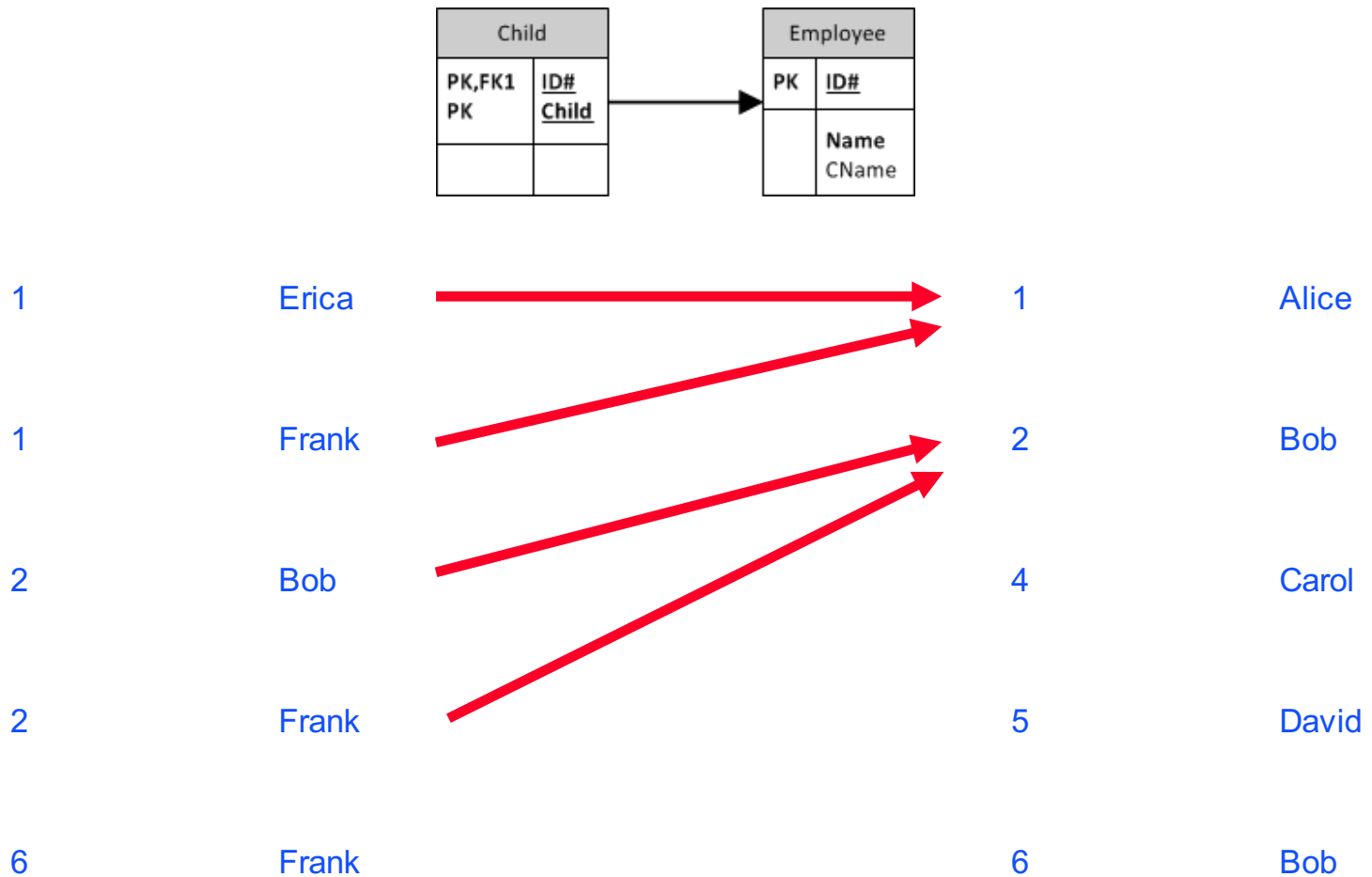
- ◆ Structurally, a relational database consists of
 1. A **set of tables with identifiers (primary keys)**
 2. A **set of many-to-one binary relationships** between them, induced by foreign key constraints

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

- ◆ Partial mapping from a set of rows into a set of rows



Very Bad Visio Diagram

- ◆ Tables are listed with attributes, specifying only which are in the primary key
- ◆ 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 Visio Diagram

- ◆ Even primary keys are not specified

Country	
	CName Population

Child	
	ID# Child

Employee	
	ID# Name CName

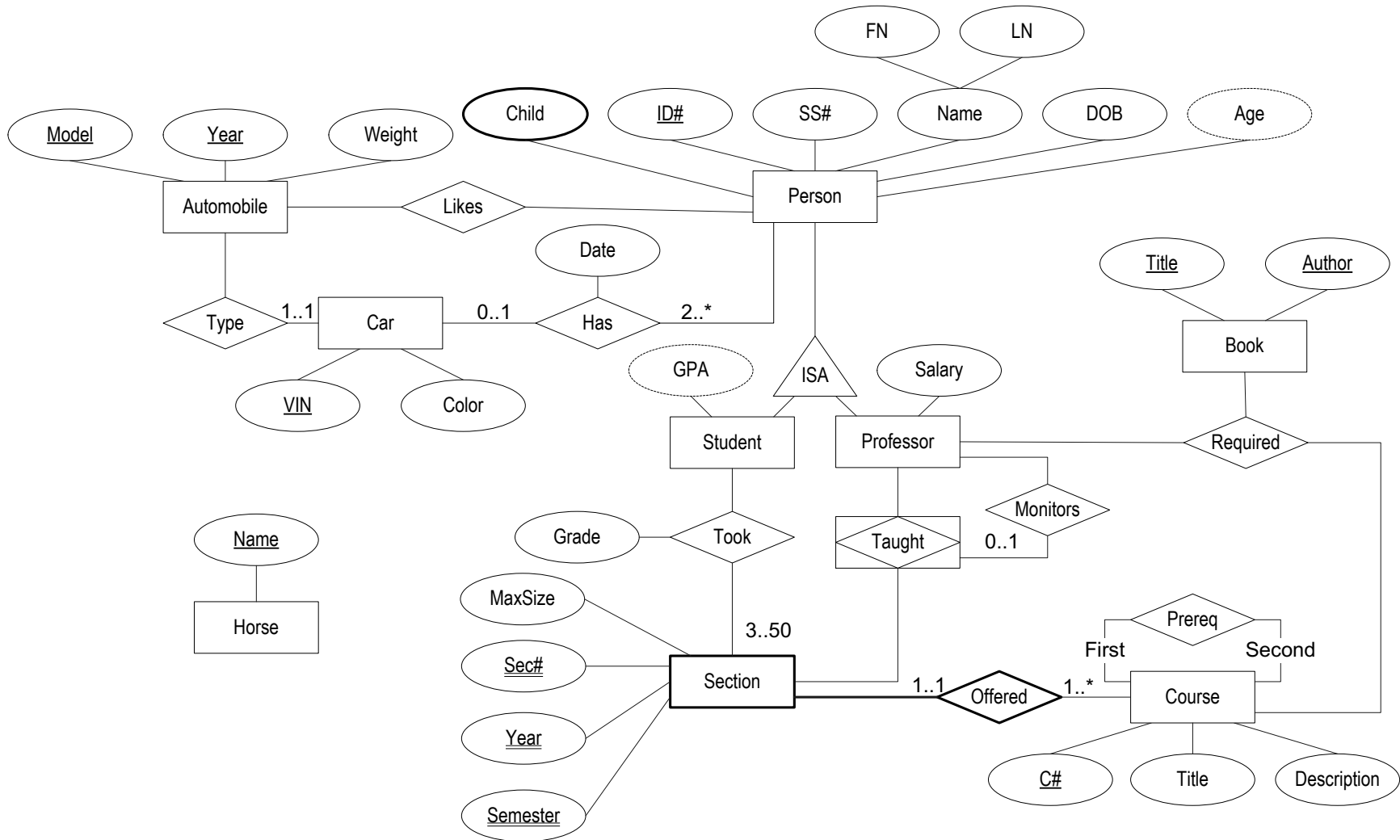
Animal	
	Species Discovered

Likes	
	ID# Species

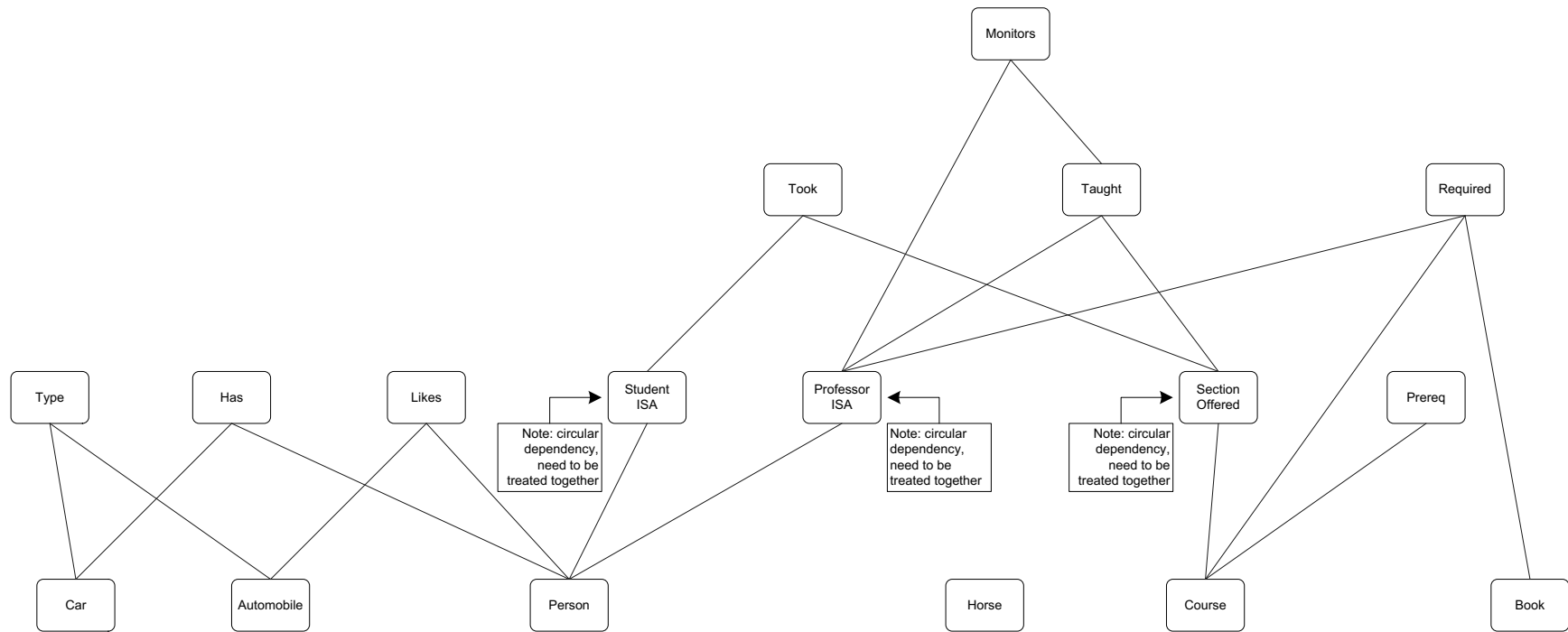
From ER Diagram To Relational Database

- ◆ 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”
- ◆ We both give a narrative description, similar to actual SQL DDL (so we are learning about actual relational databases) and Visio 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***

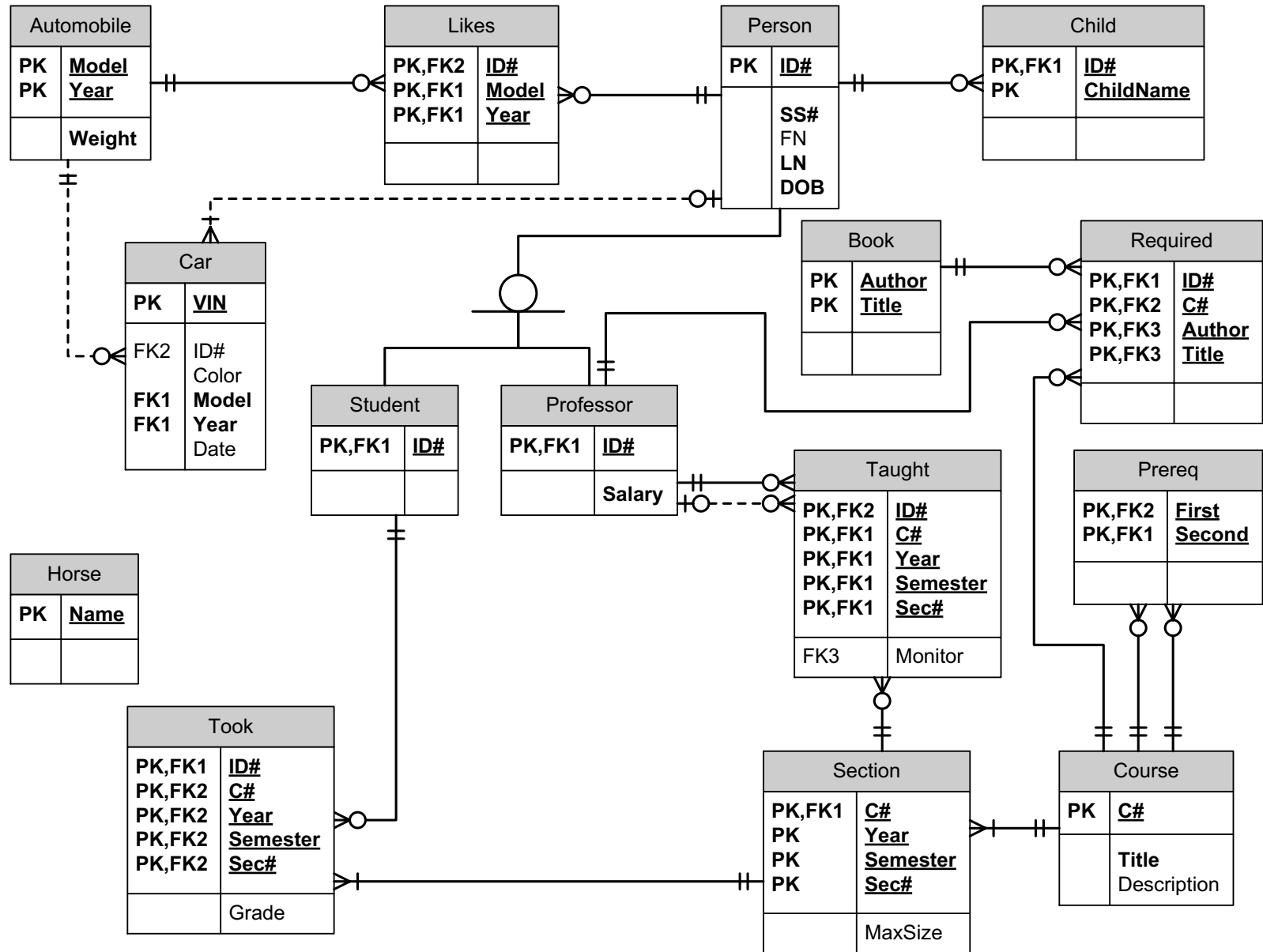
Our ER Diagram



Hierarchy For Our ER Diagram



We Will Produce



Horse

Horse	
PK	<u>Name</u>

Person

- ◆ 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

- ◆ 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

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

Horse	
PK	<u>Name</u>

Child

- ◆ 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

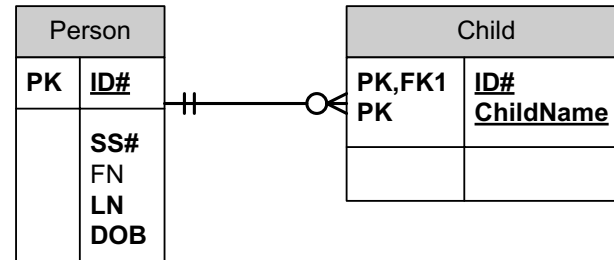
Person And Child

- ◆ 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



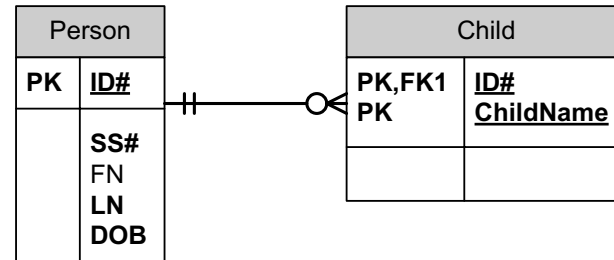
Horse	
PK	<u>Name</u>

Automobile

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

Automobile

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

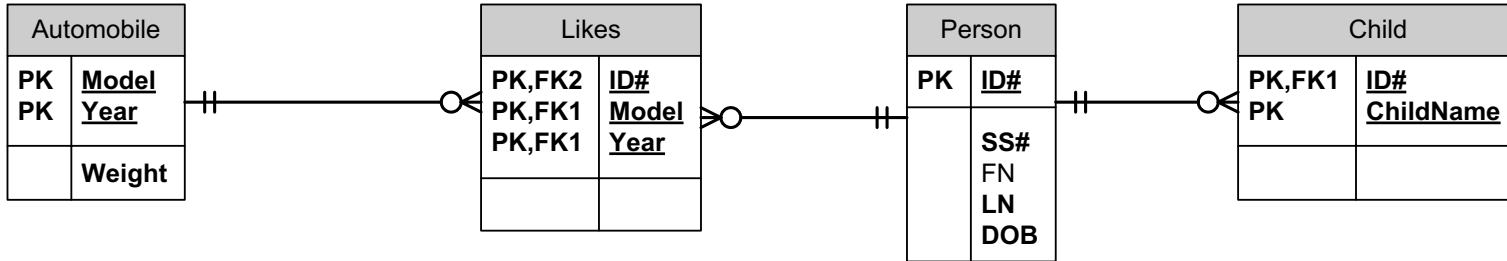


Horse	
PK	<u>Name</u>

Likes

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

Likes

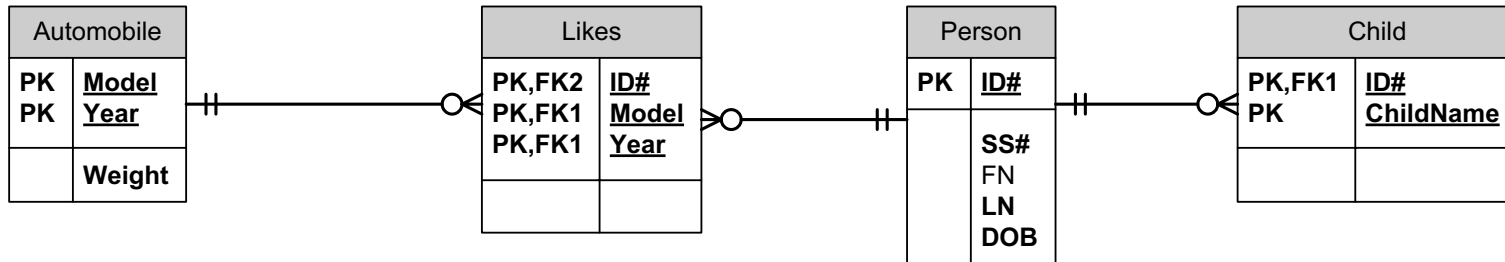


Horse	
PK	<u>Name</u>

Car

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

Car



Car	
PK	<u>VIN</u>
	Color

Horse	
PK	<u>Name</u>

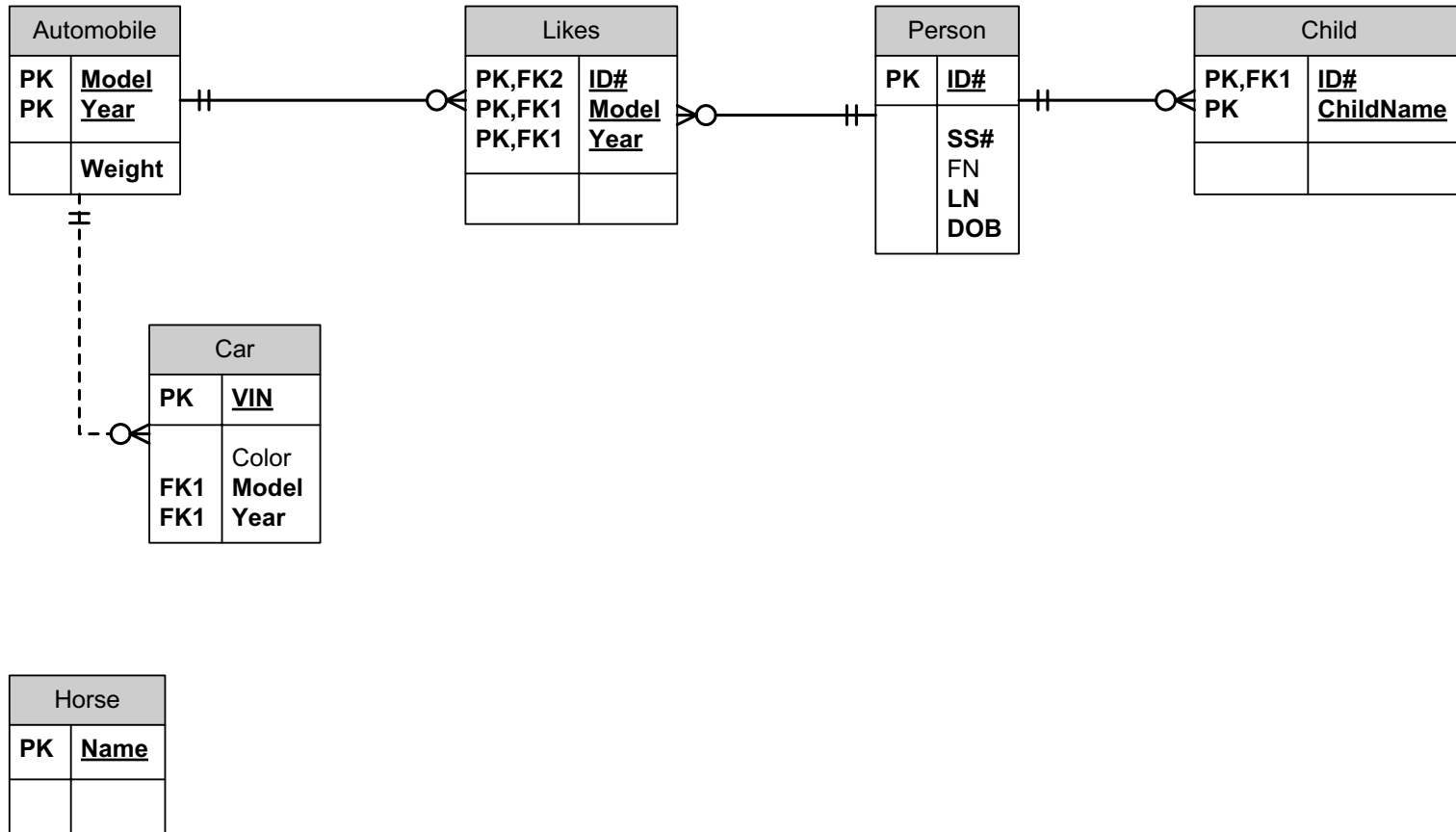
Type

- ◆ 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

Car

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

Type



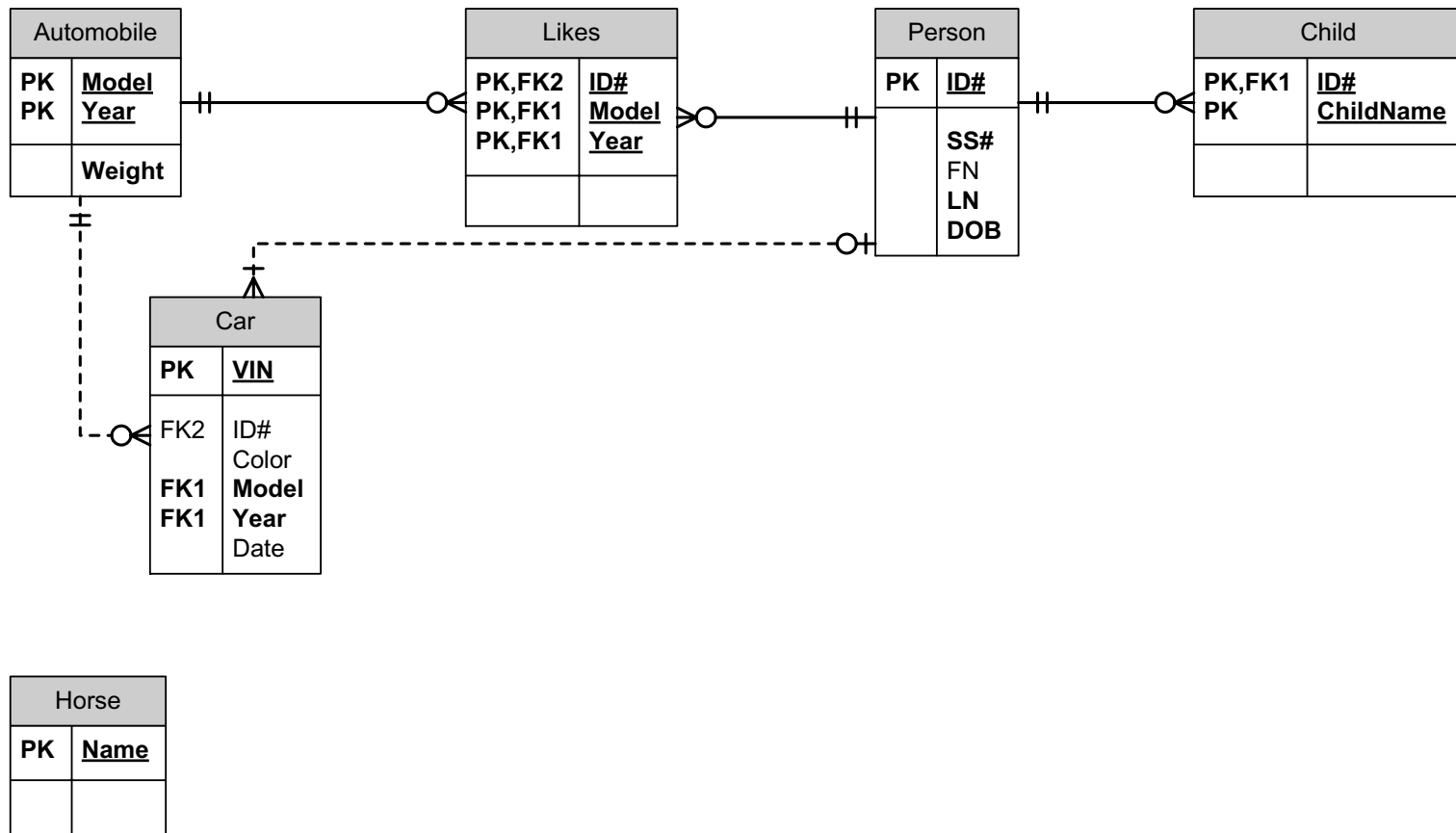
Has

- ◆ As Has is a binary many-to-one relationship, the attributed 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 also, and it is done in the examples in ExtrasForUnit03

Car

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

Has



ISA

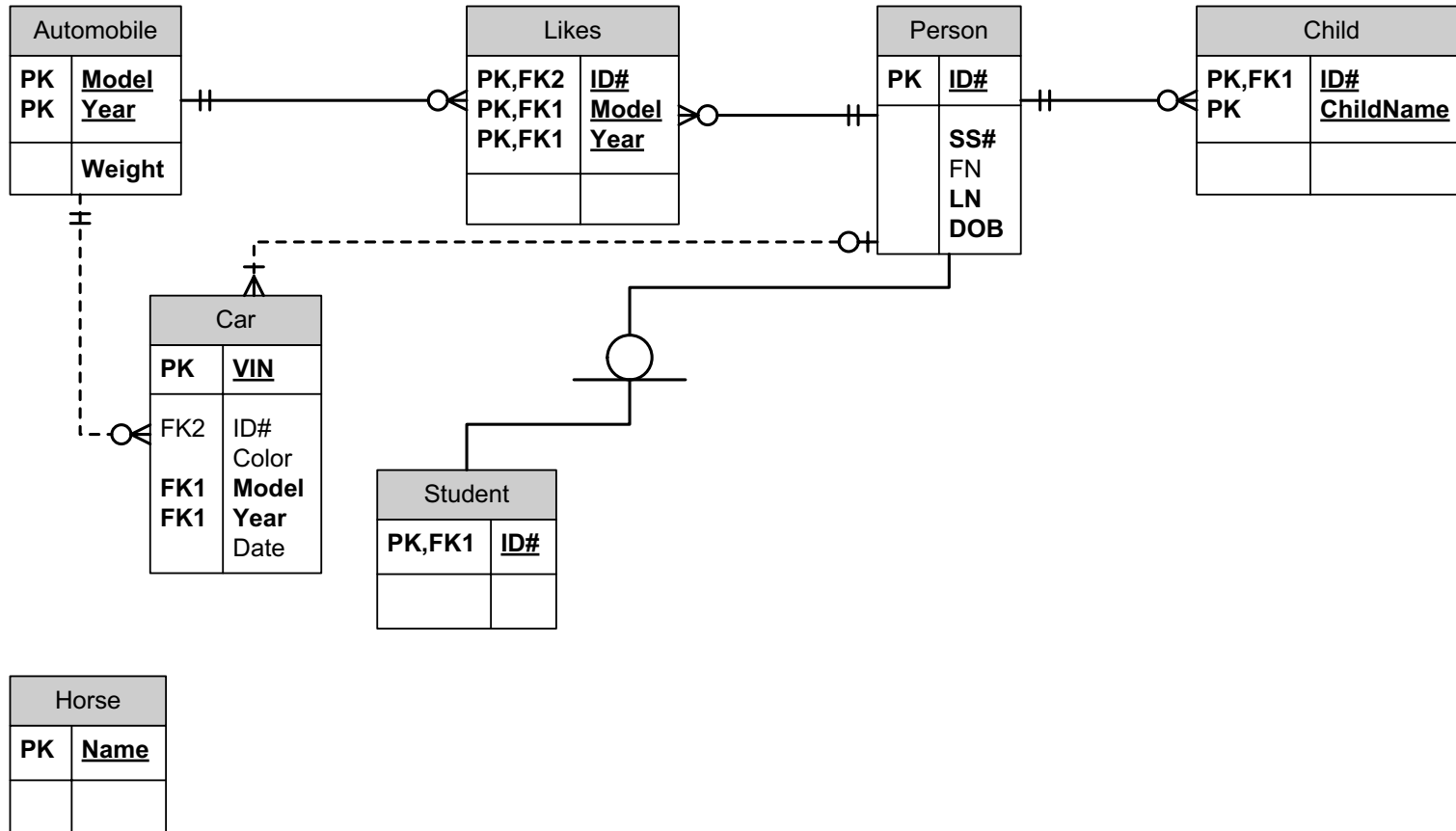
- ◆ We do not define a table for ISA
- ◆ This/these relationship/s is/are “embedded” in Student and Professor

Student

- ◆ 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

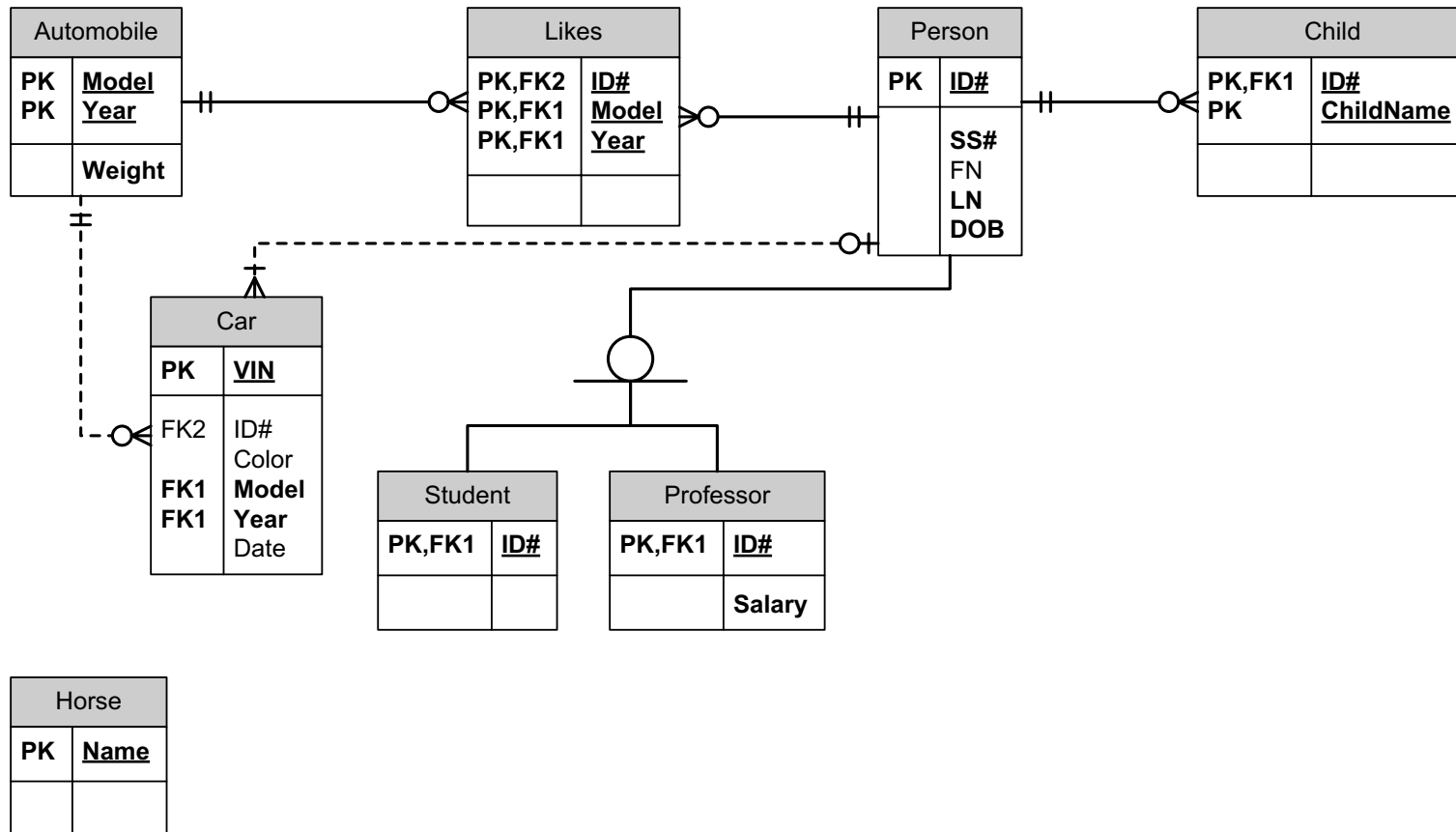
Student And ISA



Professor

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

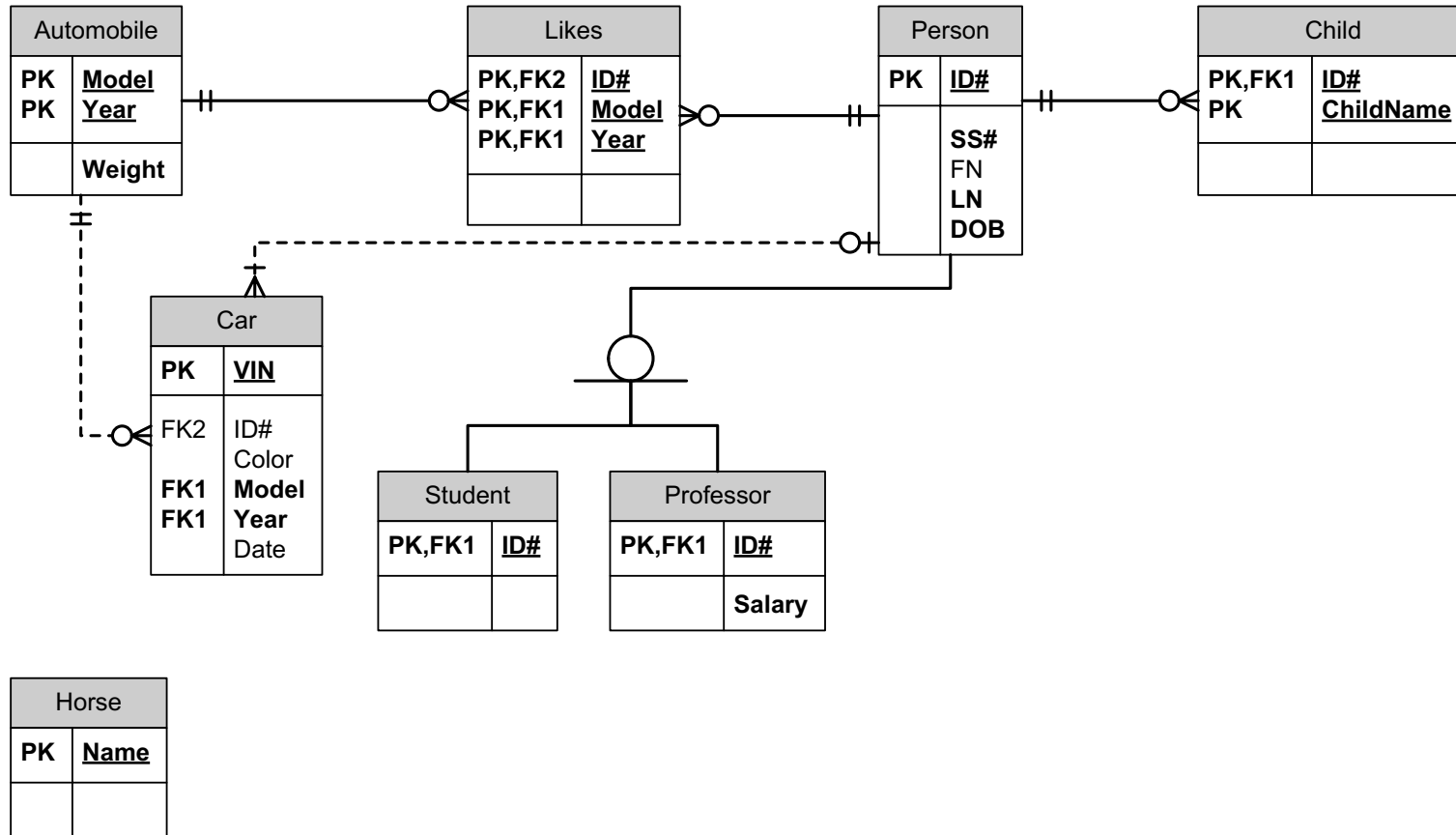
Professor And ISA



Course

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

Course



Course	
PK	<u>C#</u>
	Title Description

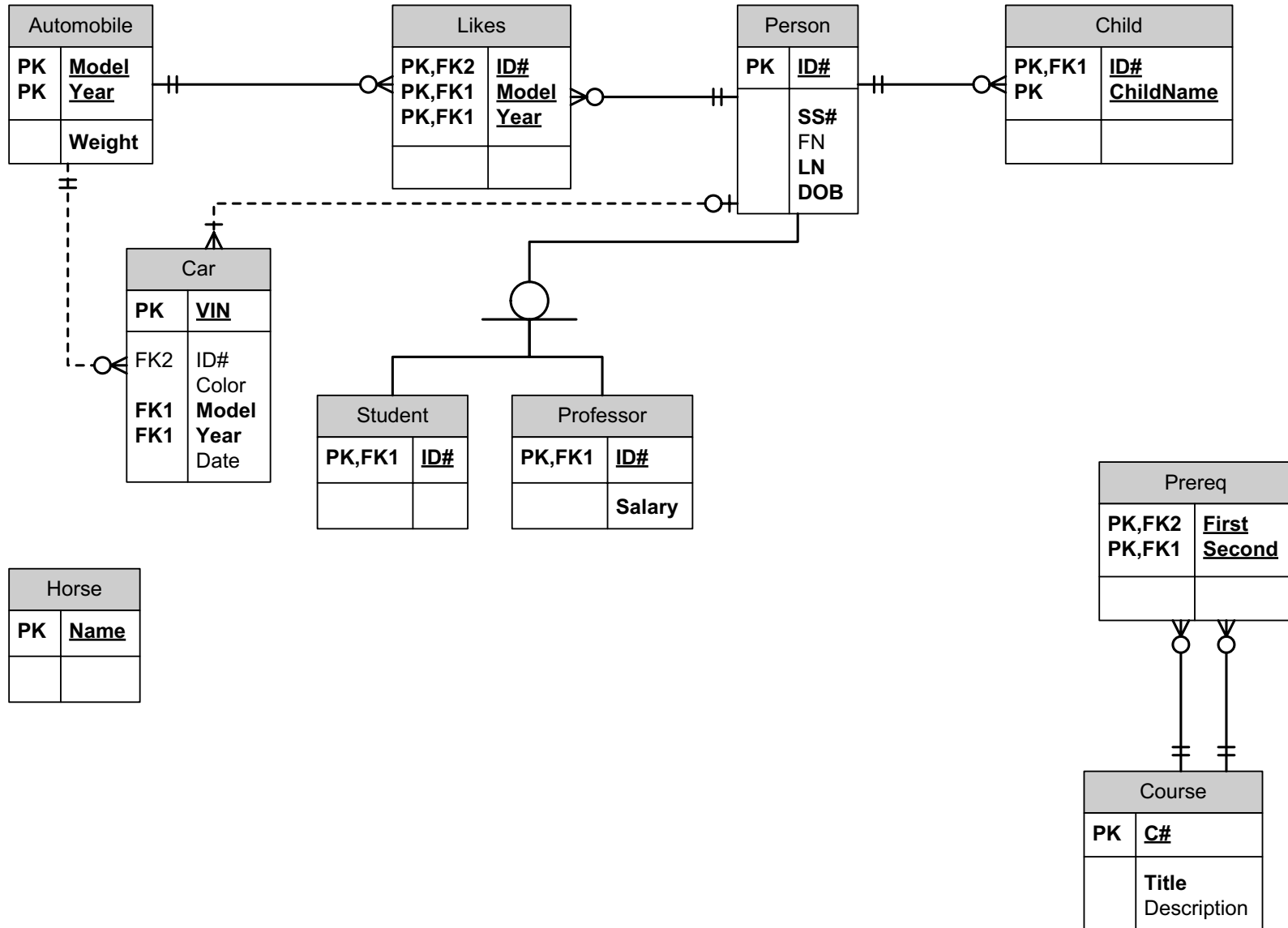
Prerequisite

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

Prereq

- ◆ 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 (but as annotations)
 - Maybe other conditions, based on numbering conventions specifying course levels

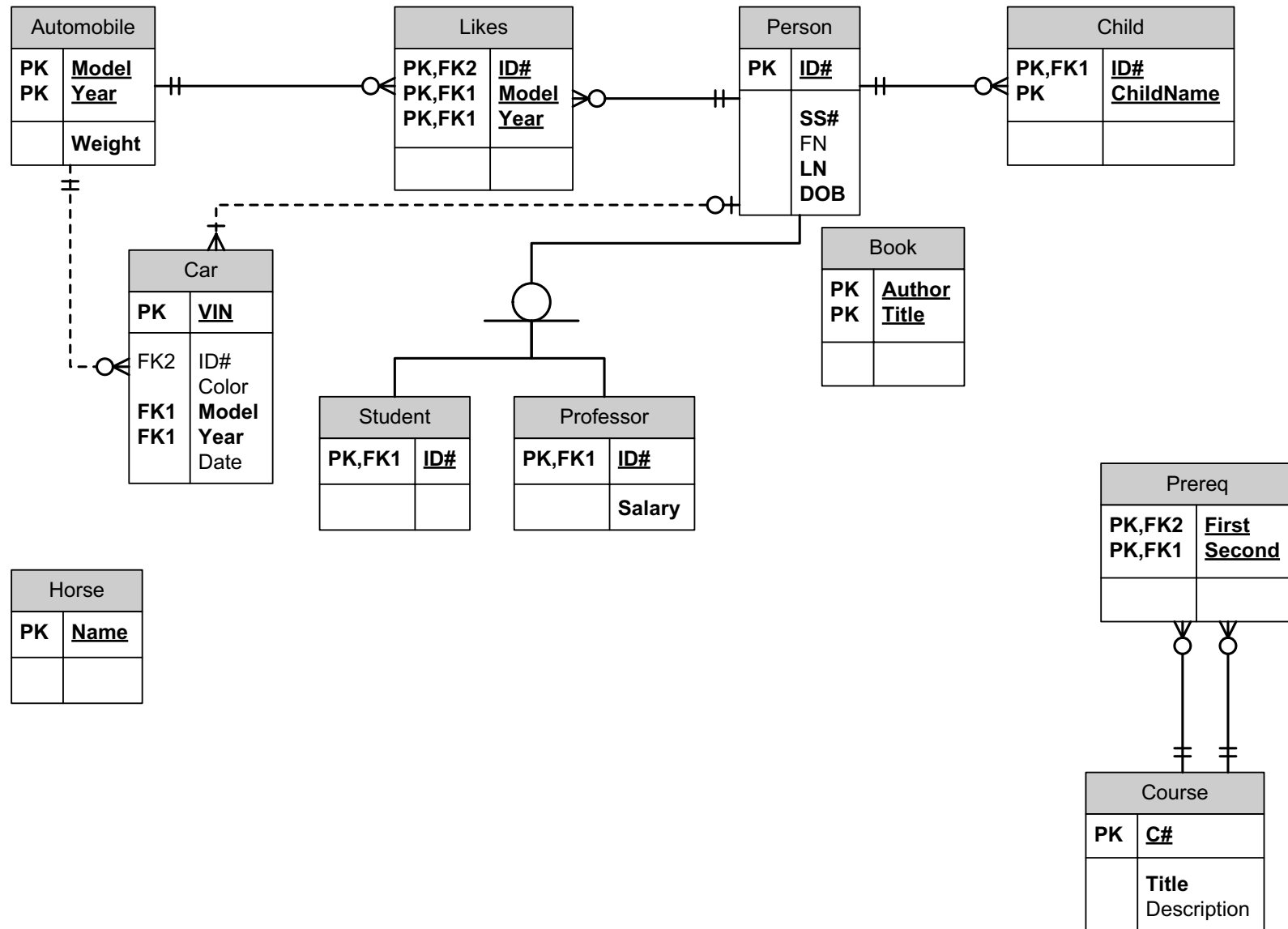
Prereq



Book

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

Book



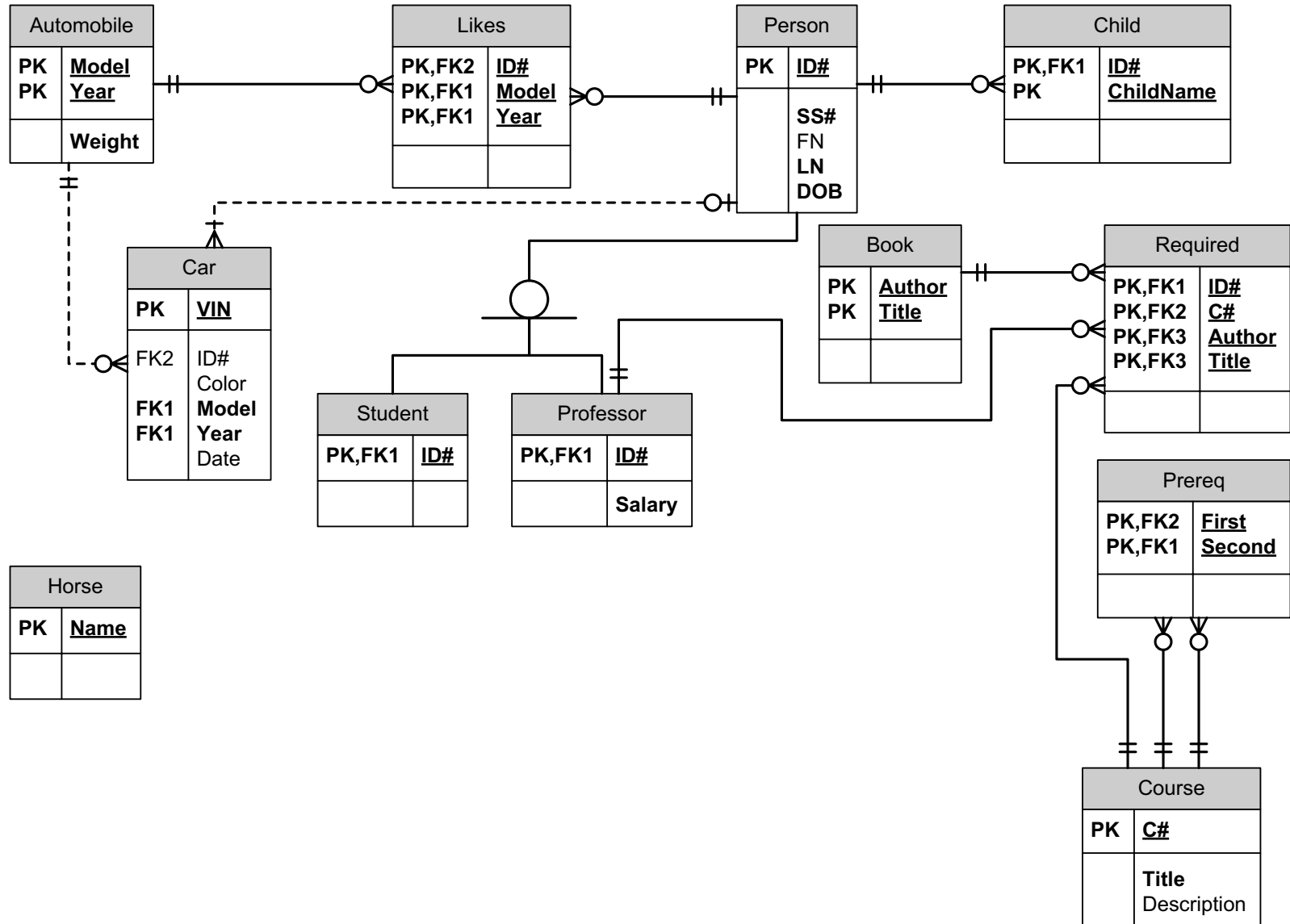
Required

- ◆ 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

Required

- ◆ 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



Section

- ◆ 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 Visio drawing how the
requirement of having at least one Section is modeled

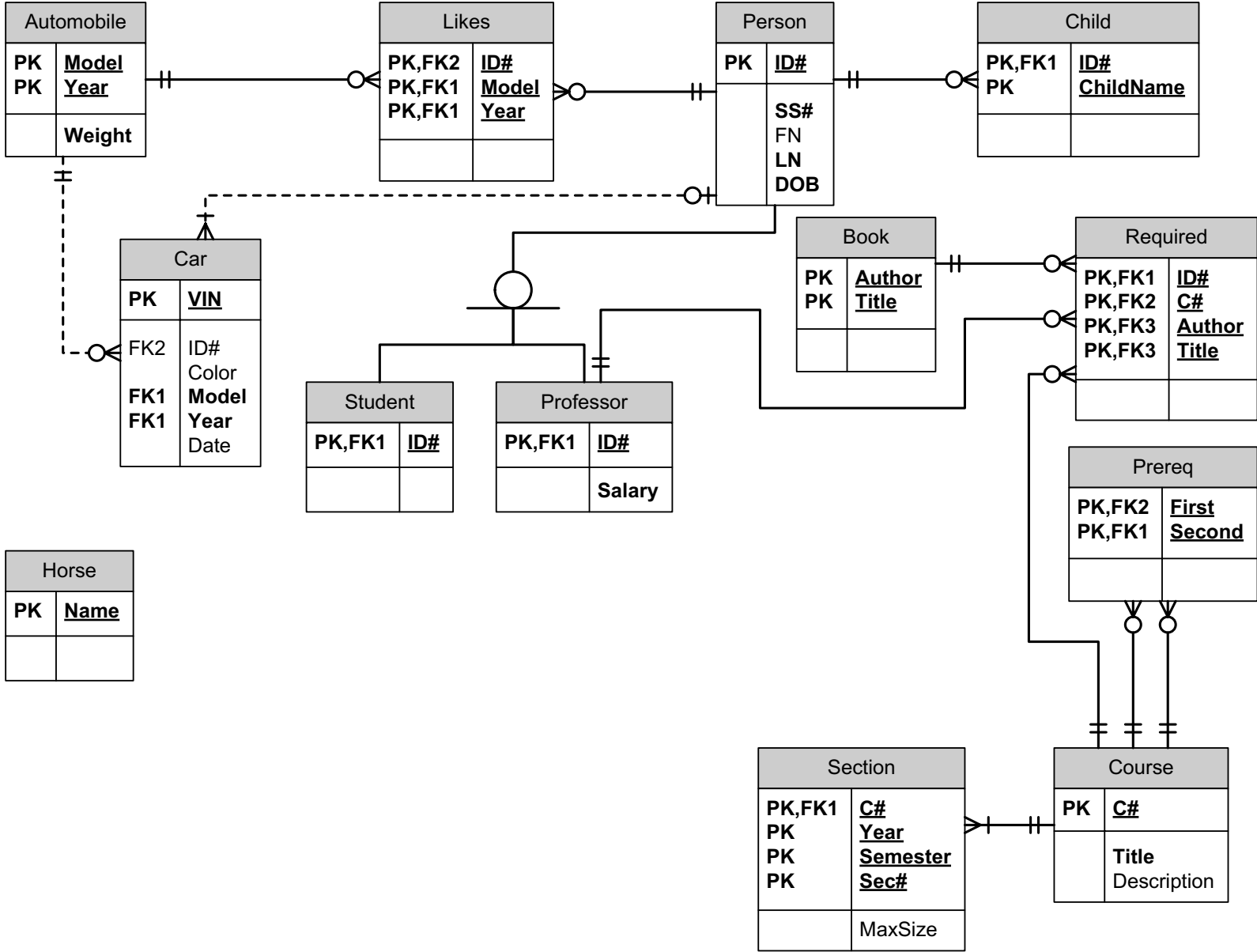
Section

- ◆ Section is our first example of a weak entity

Offered

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

Section + Offered



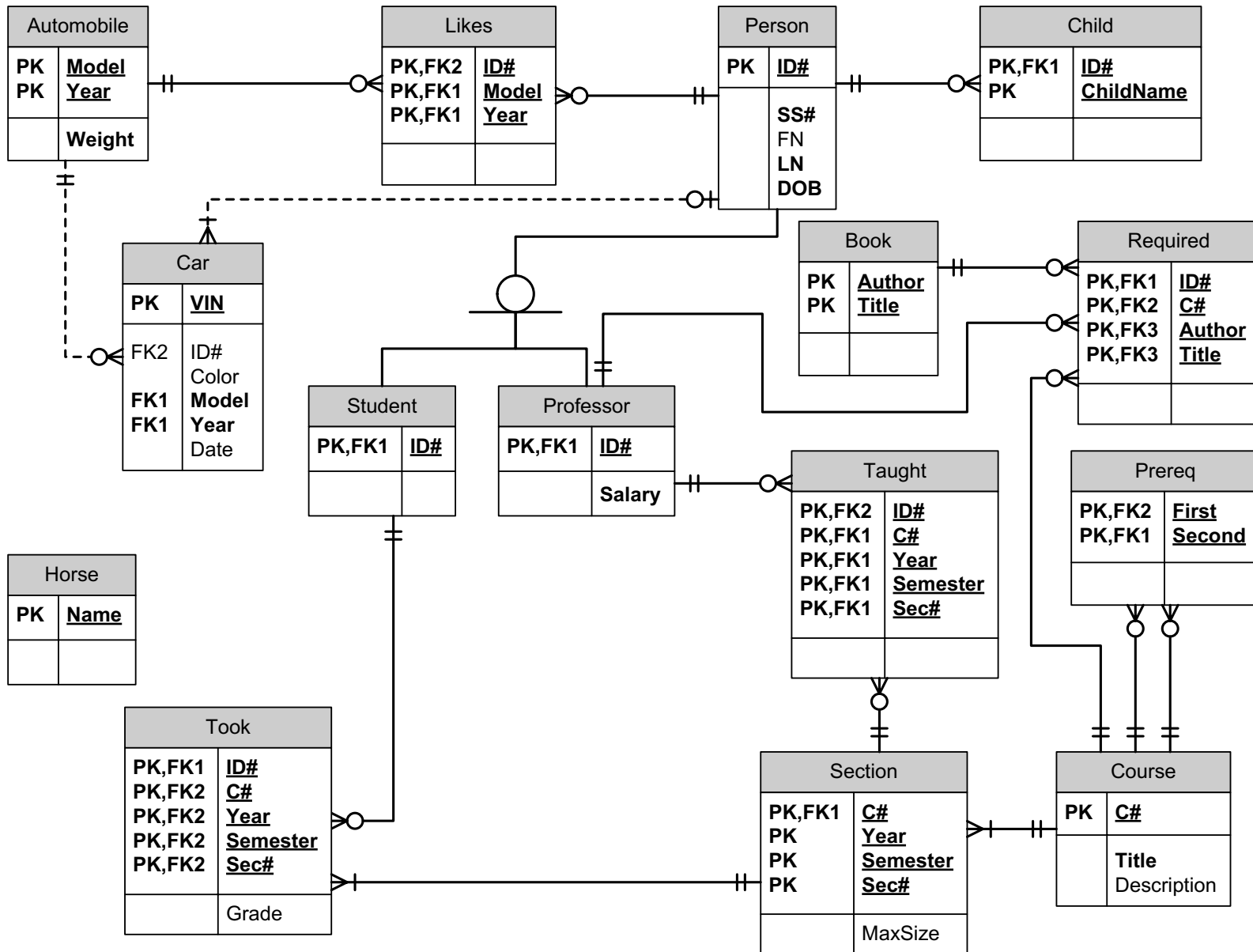
Took

- ◆ 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 Visio 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

Took

- ◆ 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

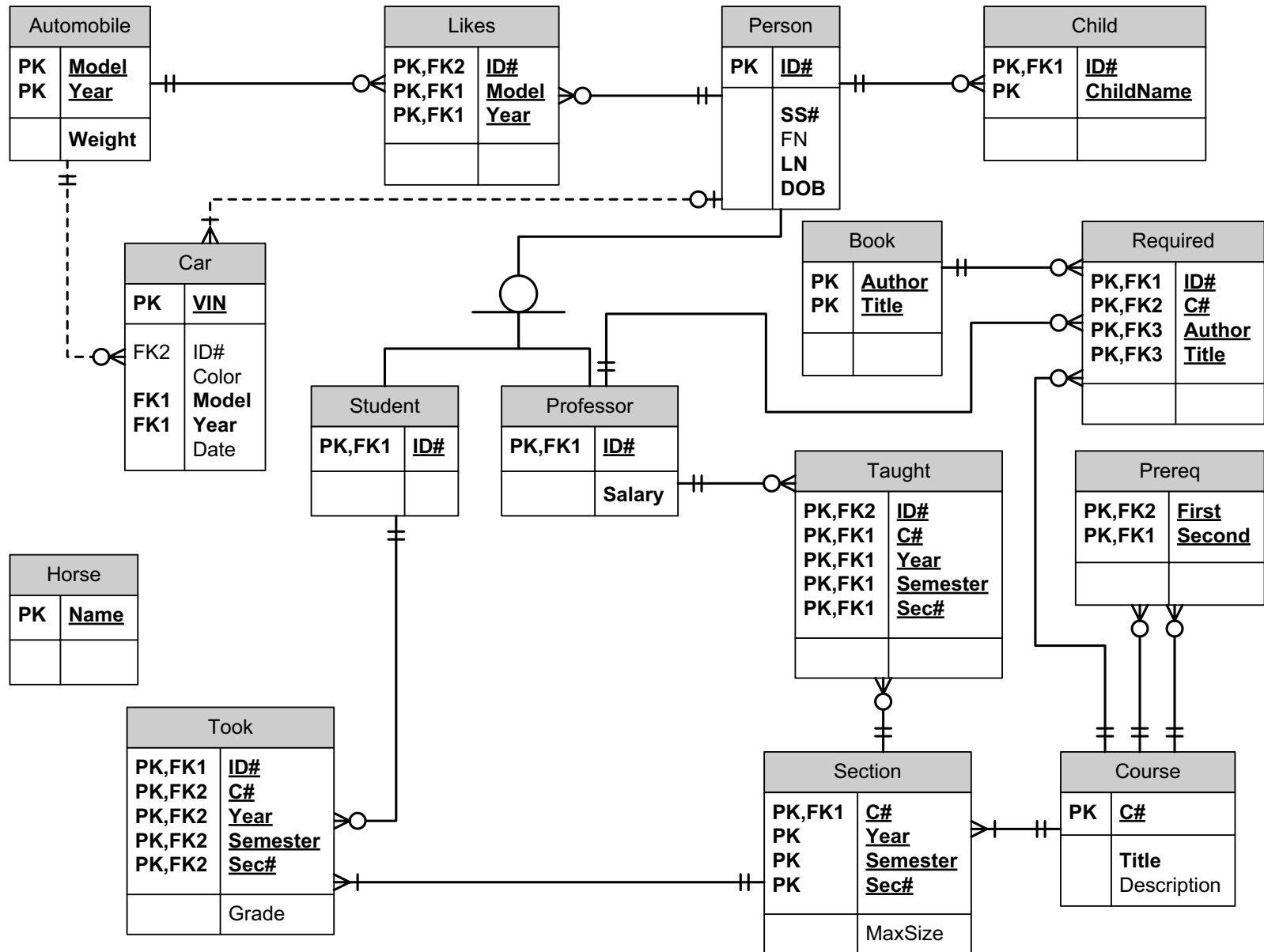
Took



Taught

- ◆ 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



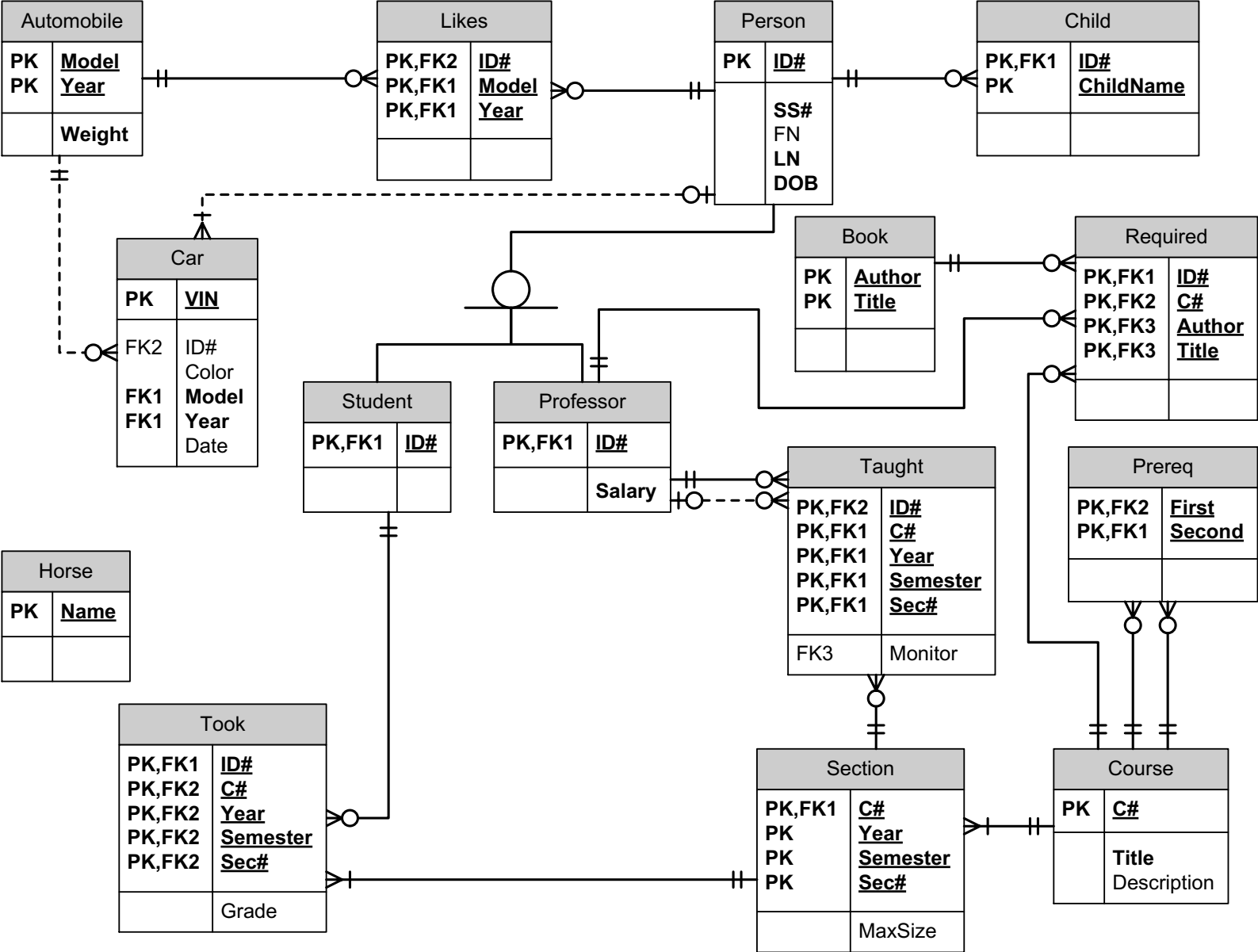
Monitors

- ◆ 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

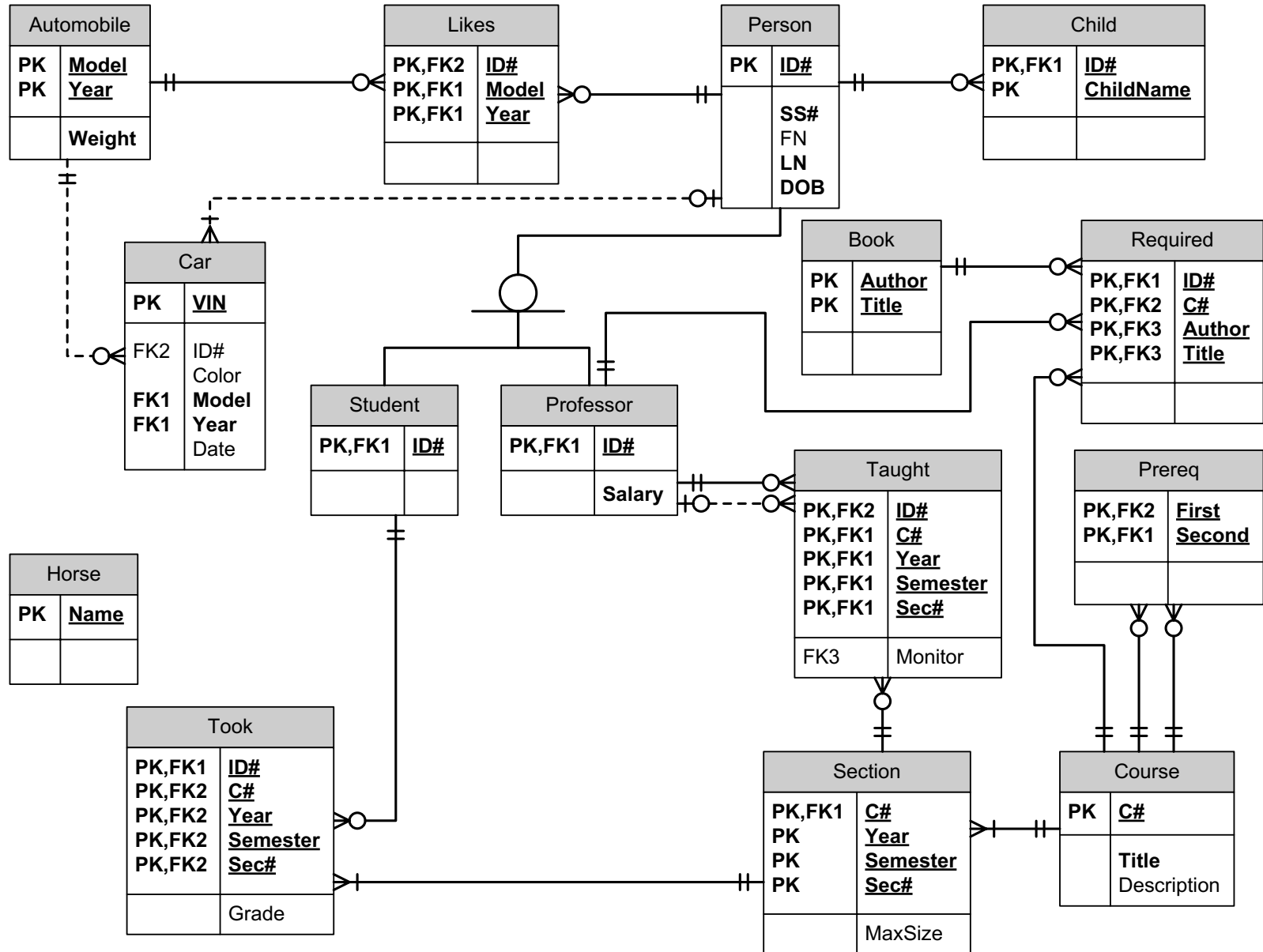
Monitors

- ◆ 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);
- ◆ **Note:** this definition of Taught replaces the original definition as, of course, we do not have two copies of the table Taught

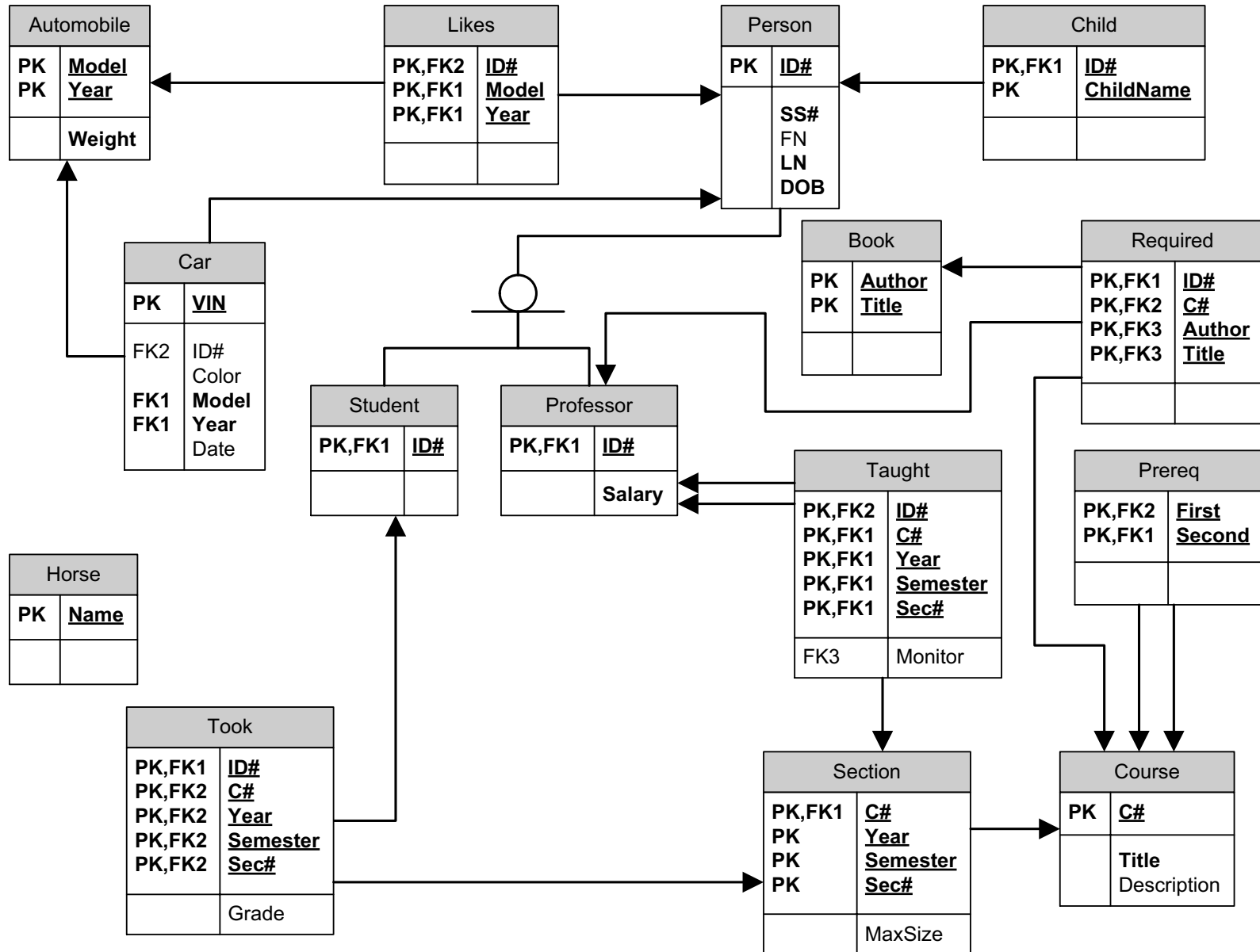
Monitors



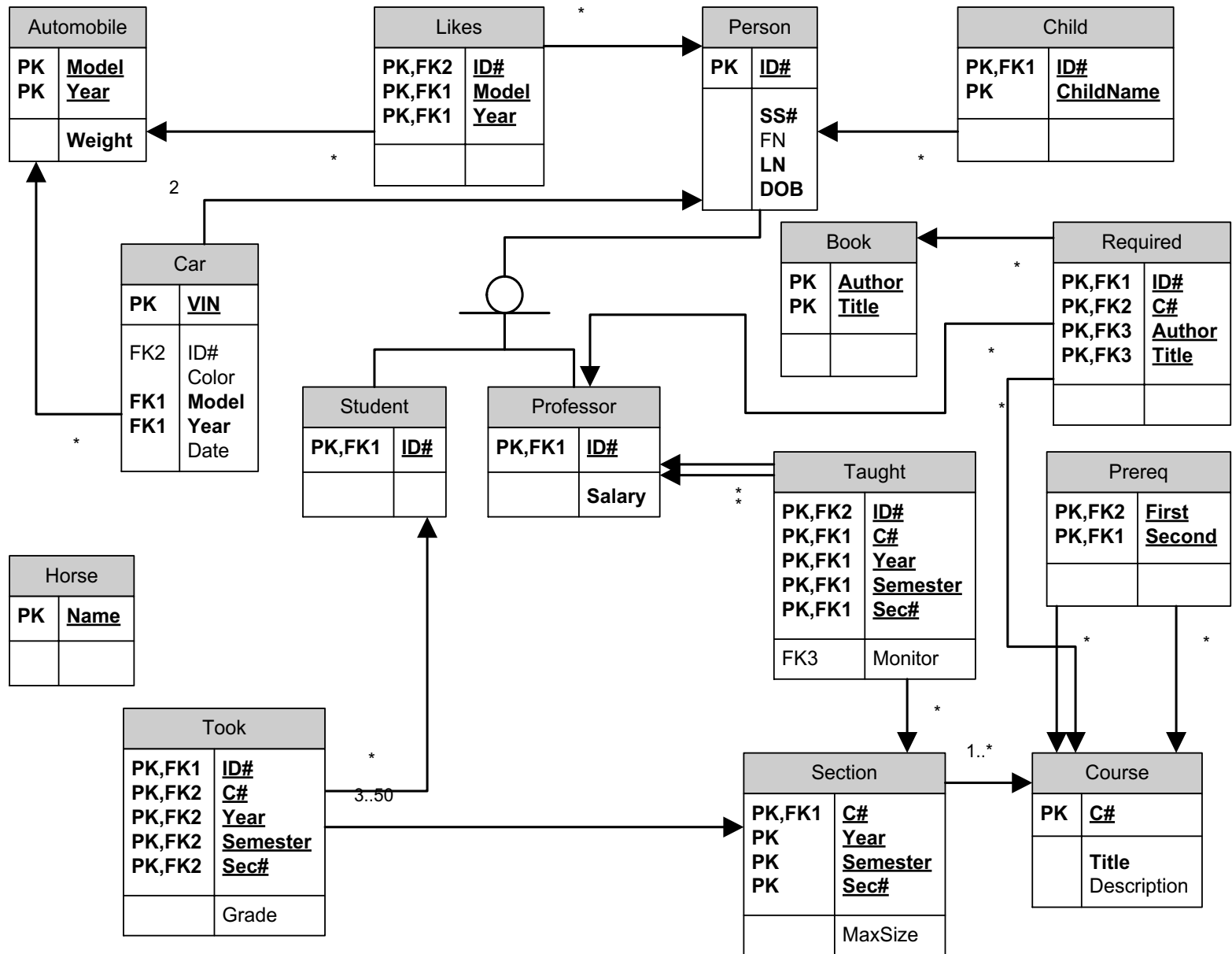
We Are Done



Arrow Notation



Arrows And Cardinality Notation



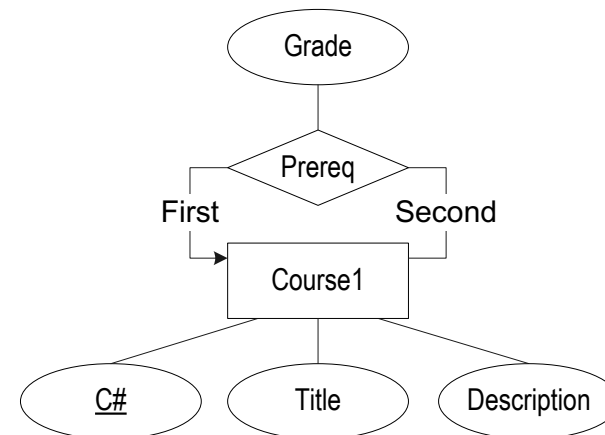
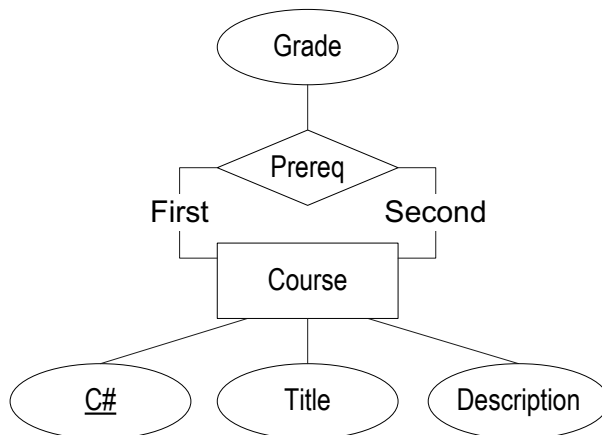
Additional Points

- ◆ We did not write out on the diagram various constraints that must be known, such as
 - At least preliminary domains, e.g., number, string, etc.
 - What is the maximum permitted section size
- ◆ ***This must be done for a proper documentation of the application's requirements***

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

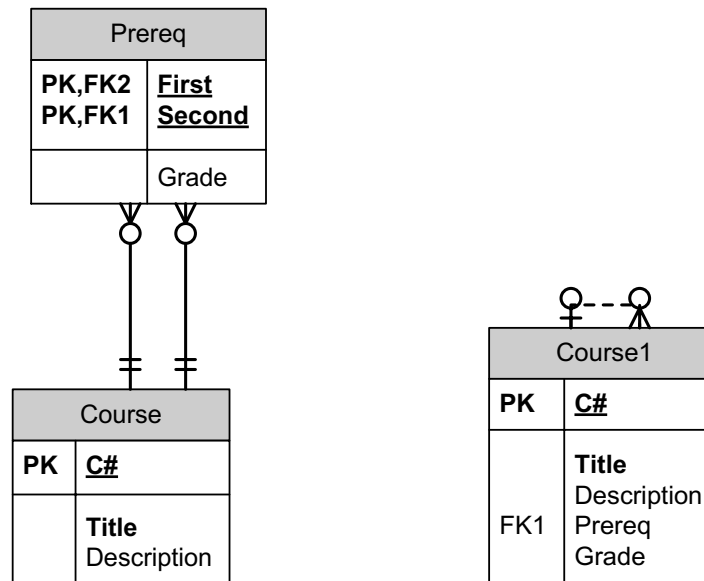
Recursive Relationships: Example

- ◆ 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

- ◆ 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

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

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

- ◆ Part of specification of foreign key in 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 in any table in the database “needs” the 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 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

Temporal Databases

- ◆ 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

- ◆ 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

- ◆ 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 produce 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 produce 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

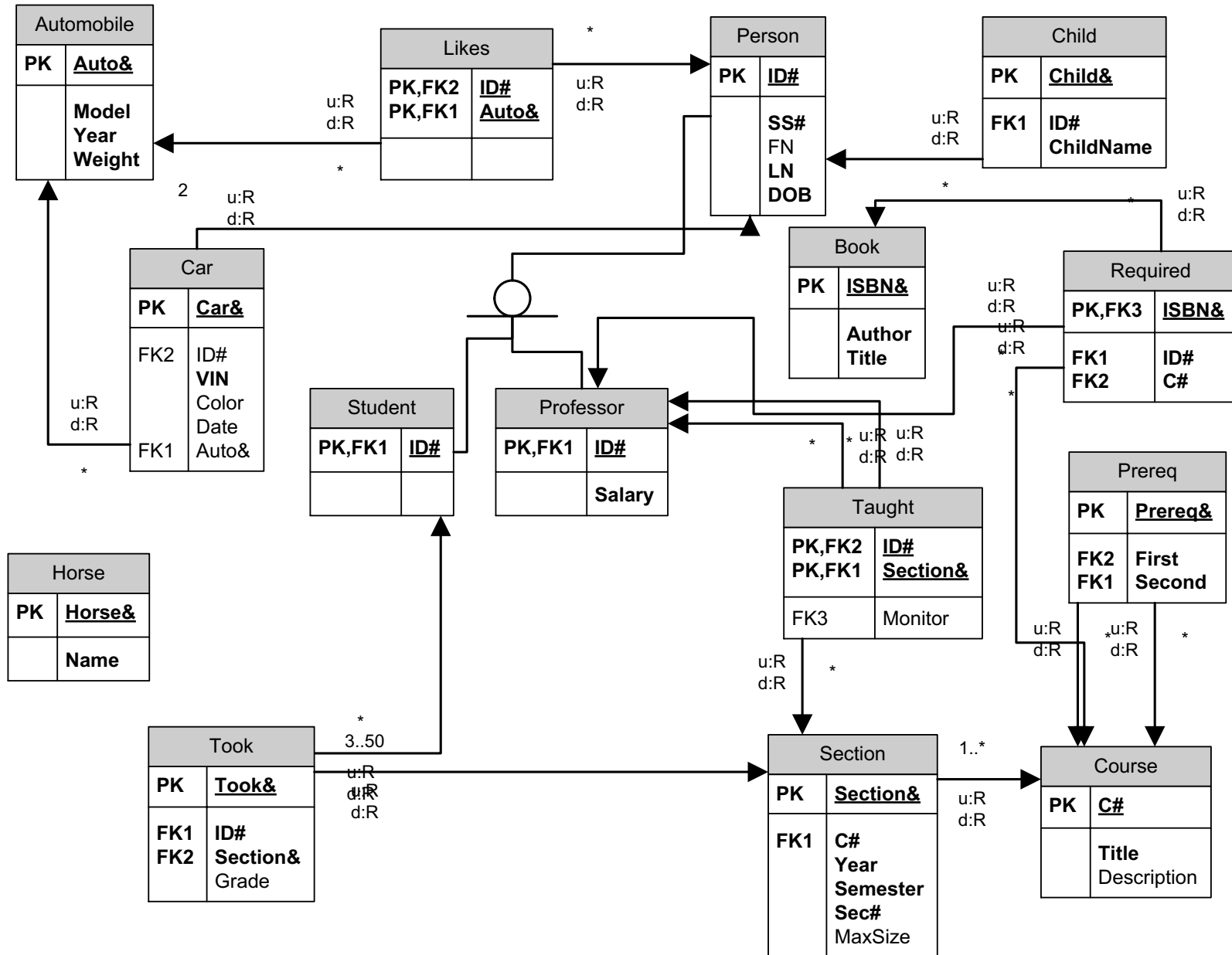
Caveat

- ◆ 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 it automatically
- ◆ Just do a reasonable thing and be ready to justify it (not to me, to yourself)

“Synthetic” Primary Keys

- ◆ Frequently it is desirable that the primary keys are not 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 taxes
 - NYU switched to N-numbers
- ◆ So we can redesign our application so that the primary keys are all synthetic and of one attribute only
- ◆ This is easy and very fast using Visio
- ◆ Synthetic attributes are denoted here by ending with “&”
- ◆ Do not use synthetic keys on homework or exam

Our Application with Synthetic Primary Keys



Key Ideas

- ◆ 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
- ◆ Crow's feet notation: ends of lines
- ◆ Crow's feet notation: pattern of lines
- ◆ Synthetic keys