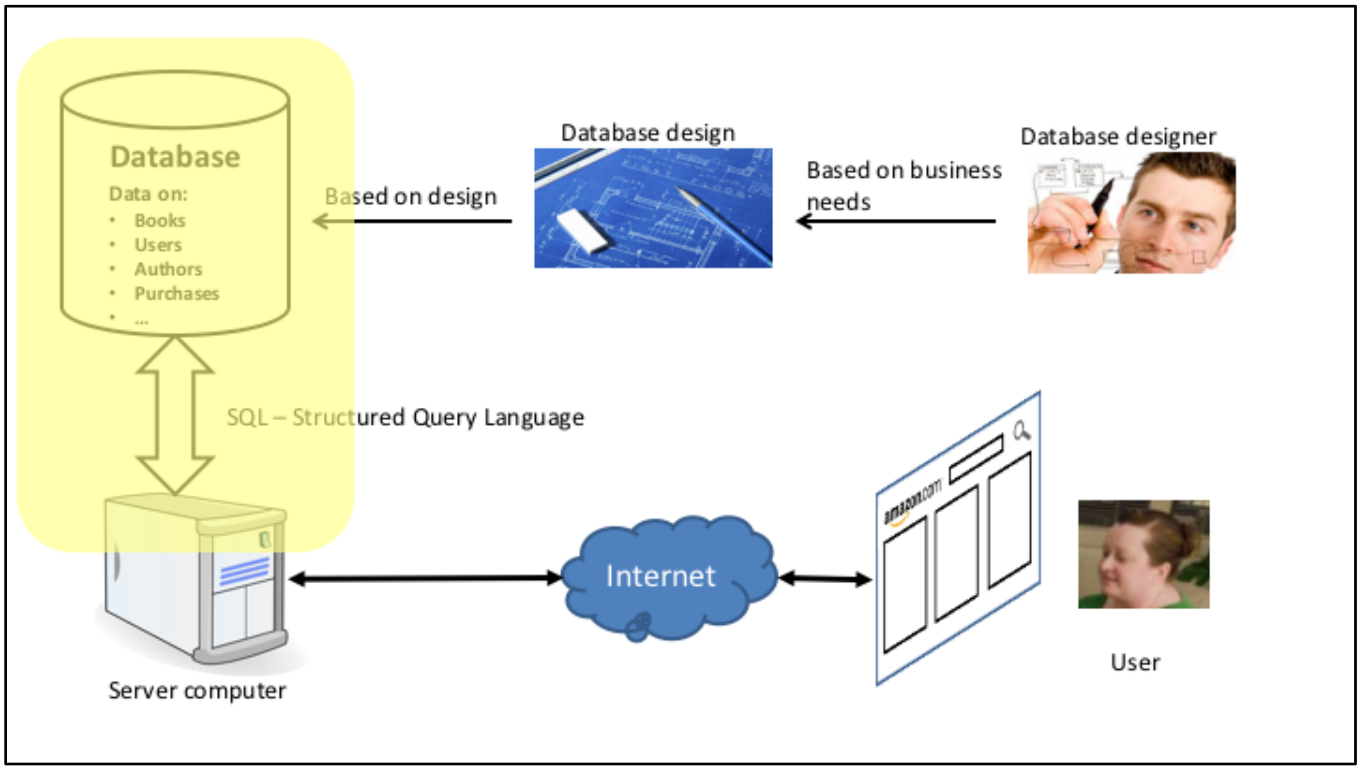
Database Design: Part 1



We have seen this figure before. Thus far we have looked at how enterprise data stored in a relational database using SQL.

By now you must be convinced that we can get out pretty much any information we need by using SQL.

As I have said a few times before, I find this all the more fascinating because at the time we design the database, we do not have to ever consider what retrieval requirements we might have .. And yet, we can easily get whatever we want from the database.

Little wonder then that relational databases have served as the backbone of IT systems for over three decades now. Without this technology, IT applications might not have developed to the extent they have.

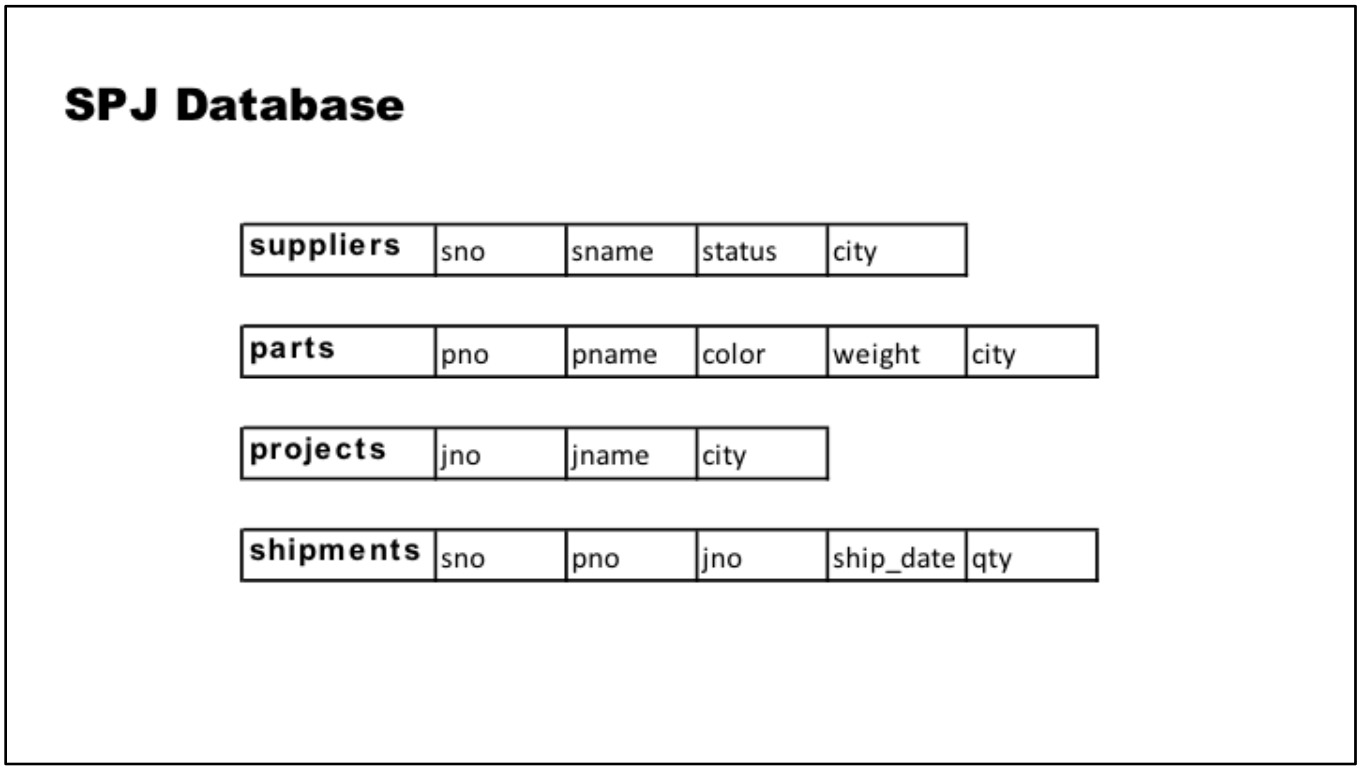
With increased volumes of data – like those that companies like Google and Facebook handle, we need different technologies – but that has to wait for another course …

We now trun out attention to database design. We have seen several (small, to be sure) relational databases – Suppliers-parts, SBA basketball, College, Sakila and classicmodels. Each had a set of tables and we enjoyed waving our SQL wand at these to get information.

How did whoever desinged these databases decide exactly what tables to have and what fields to keep in each table. After all they could have done things in so many different ways.

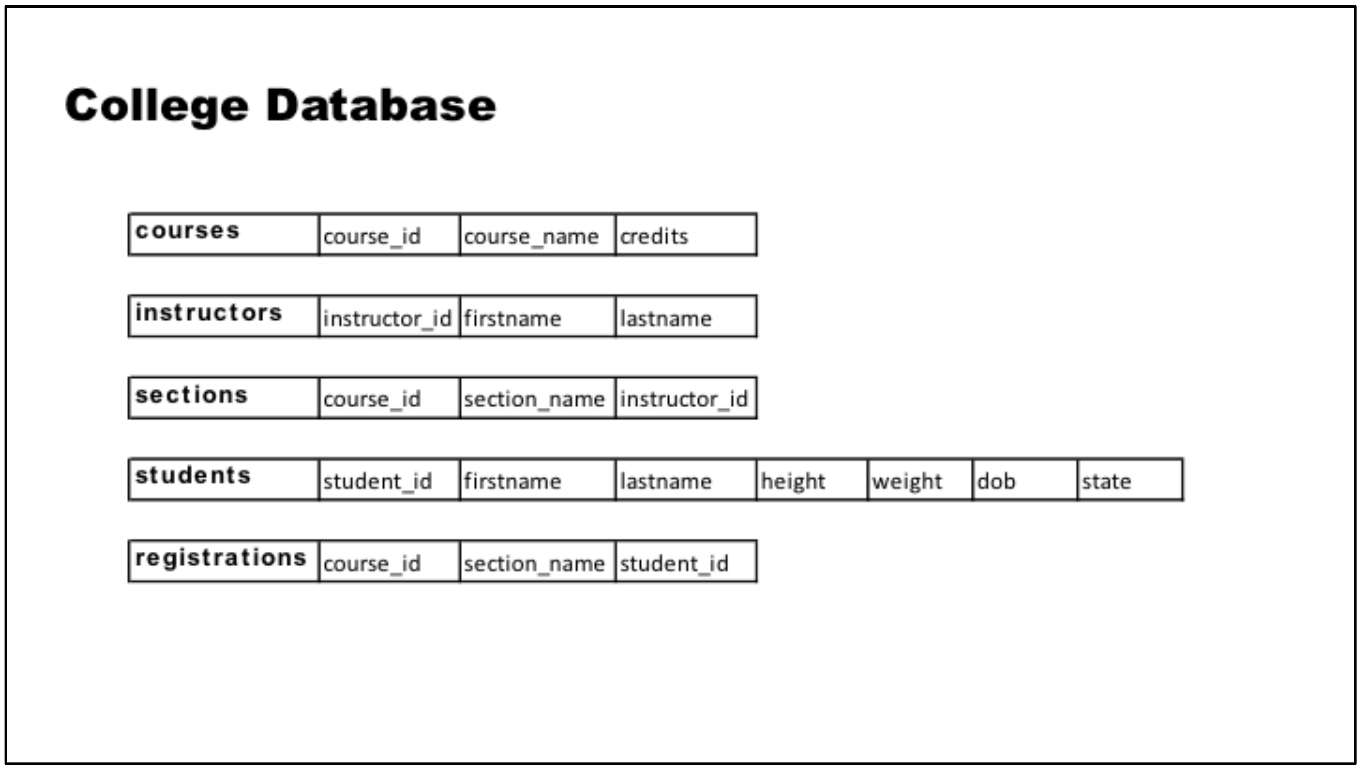
Why did they choose to do it exactly like they did?

Enter “Database Design” using “Entity Relationship Diargrams” or ERD.



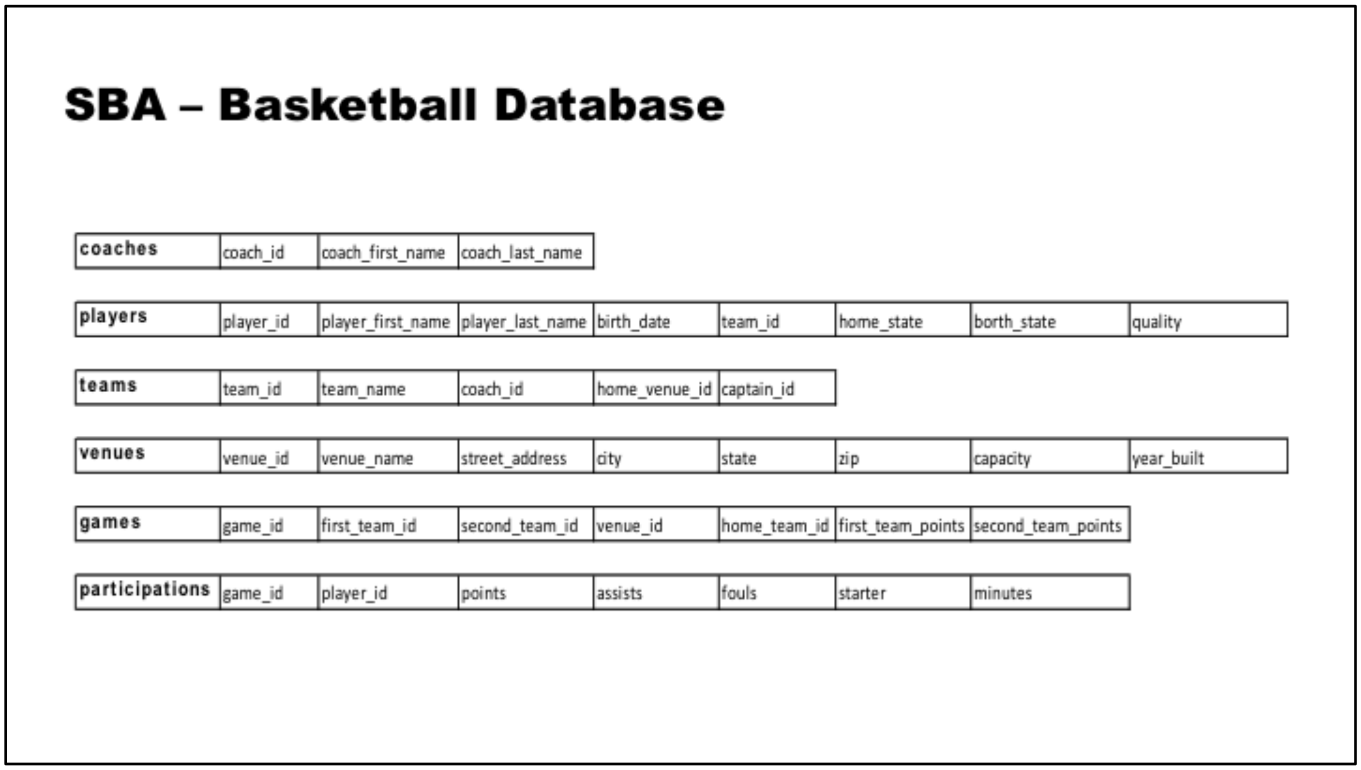
SPJ Database

suppliers	sno	sname	status	city	
parts	pno	pname	color	weight	city
projects	jno	jname	city		
shipments	sno	pno	jno	ship_date	qty



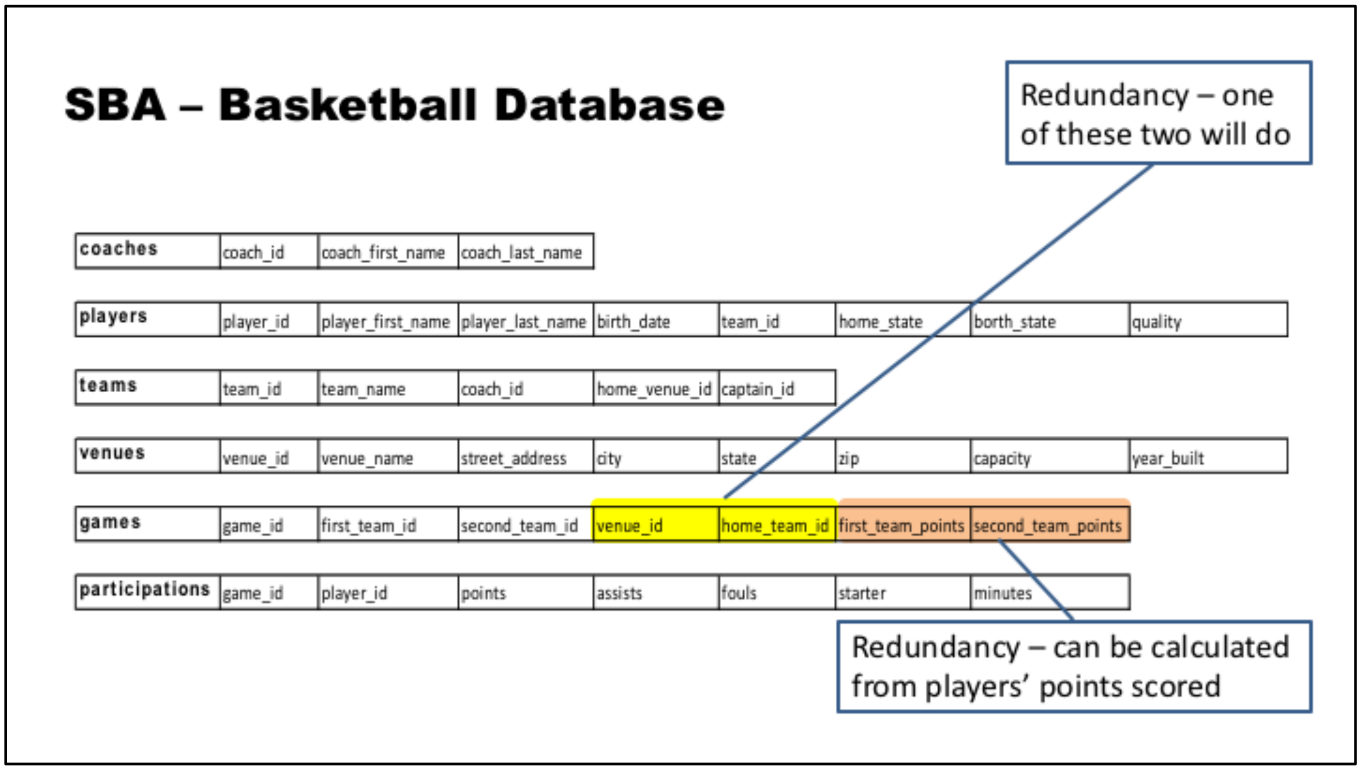
College Database

courses	course_id	course_name	credits				
instructors	instructor_id	firstname	lastname				
sections	course_id	section_name	instructor_id				
students	student_id	firstname	lastname	height	weight	dob	state
registrations	course_id	section_name	student_id				



SBA – Basketball Database

coaches	coach_id	coach_first_name	coach_last_name					
players	player_id	player_first_name	player_last_name	birth_date	team_id	home_state	borth_state	quality
teams	team_id	team_name	coach_id	home_venue_id	captain_id			
venues	venue_id	venue_name	street_address	city	state	zip	capacity	year_built
games	game_id	first_team_id	second_team_id	venue_id	home_team_id	first_team_points	second_team_points	
participations	game_id	player_id	points	assists	fouls	starter	minutes	

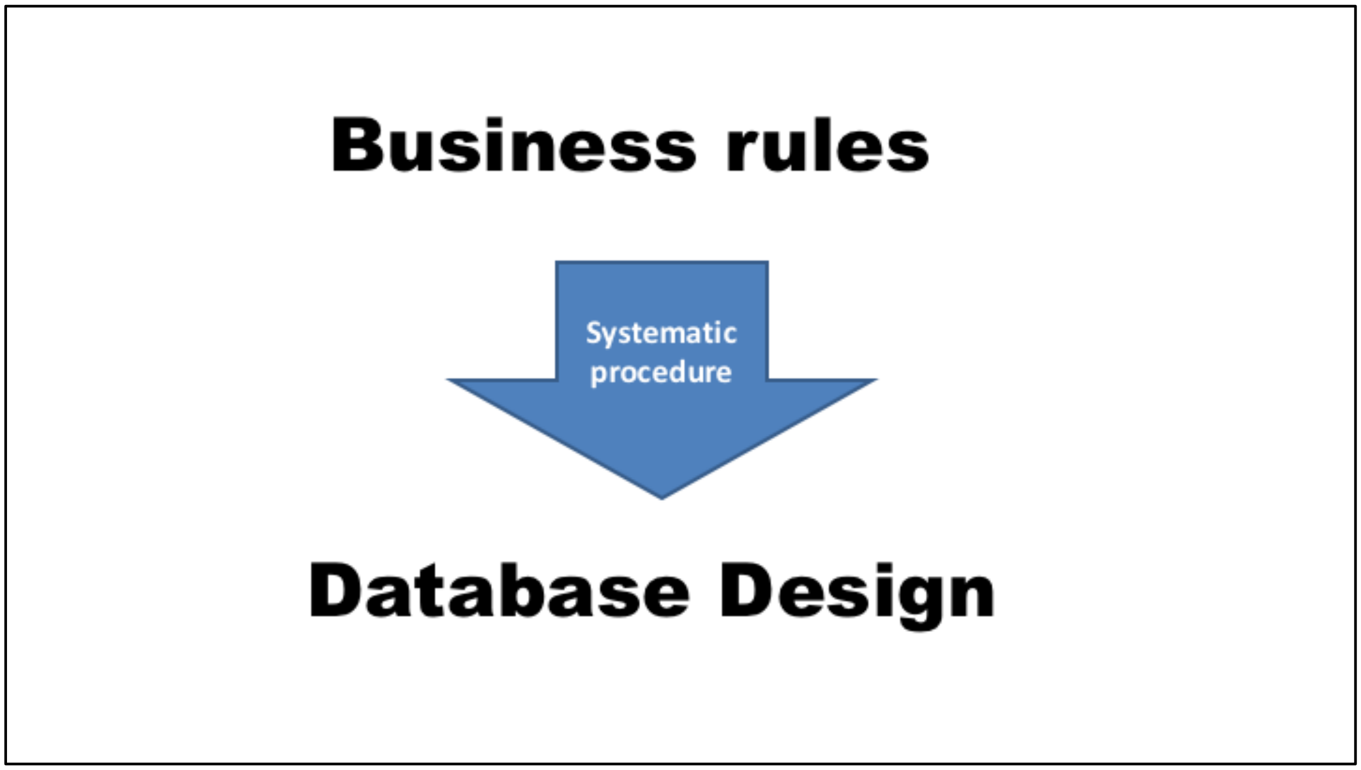


SBA – Basketball Database

Redundancy – one of these two will do

coaches	coach_id	coach_first_name	coach_last_name					
players	player_id	player_first_name	player_last_name	birth_date	team_id	home_state	borth_state	quality
teams	team_id	team_name	coach_id	home_venue_id	captain_id			
venues	venue_id	venue_name	street_address	city	state	zip	capacity	year_built
games	game_id	first_team_id	second_team_id	venue_id	home_team_id	first_team_points	second_team_points	
participations	game_id	player_id	points	assists	fouls	starter	minutes	

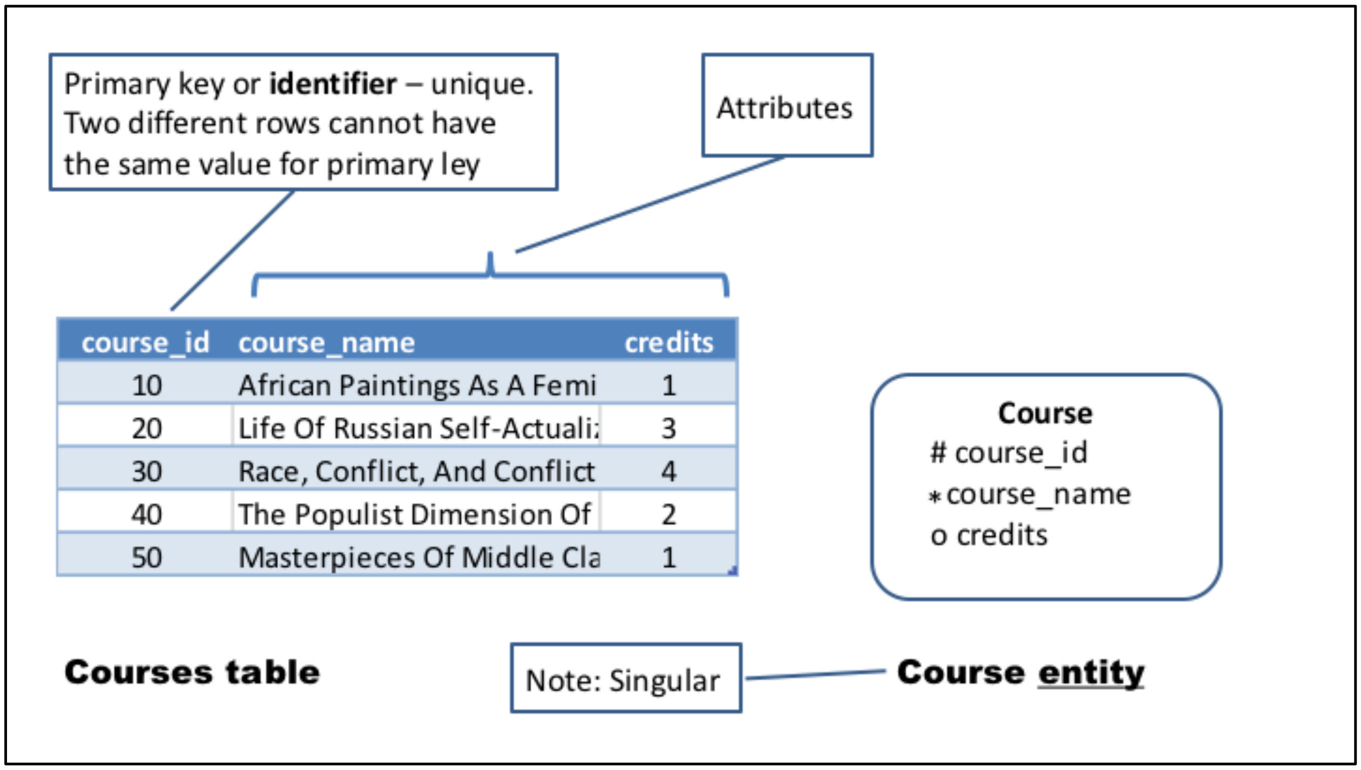
Redundancy – can be calculated from players' points scored



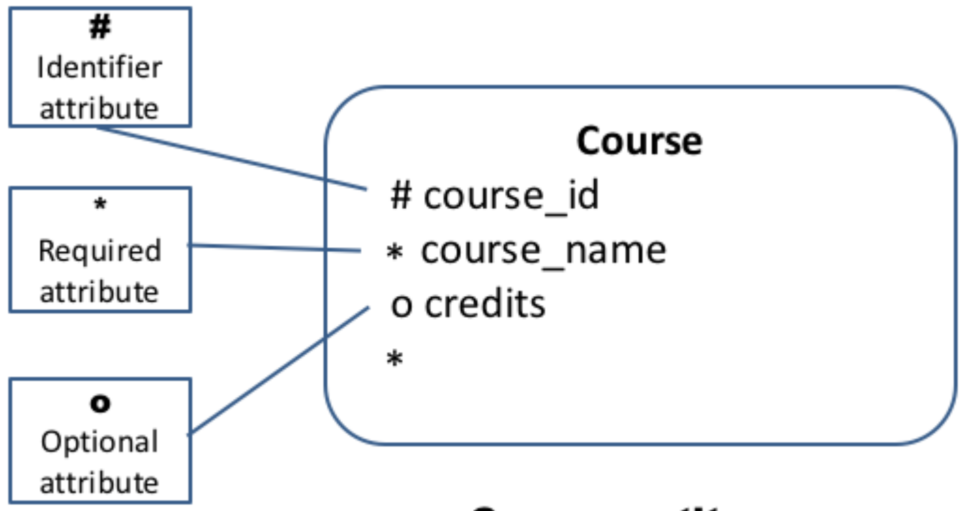
We will learn a structured process to understand a business situation and convert that understanding into a relational database design that can support almost any queries that would likely arise in managing operations.

AT this stage we do not need to explicitly consider what information might need to be retrieved. Just represent some kinds of business rules in a rigorous diagrammatic notation and the database design automatically pops out of the process. Cool stuff indeed!

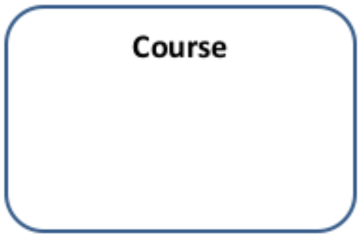
Business knowledge plays a key role in this process.


Primary key or **identifier** – unique. Two different rows cannot have the same value for primary ley
Attributes
Course
# course_id
* course_name
o credits
Courses table
Note: Singular
Course entity

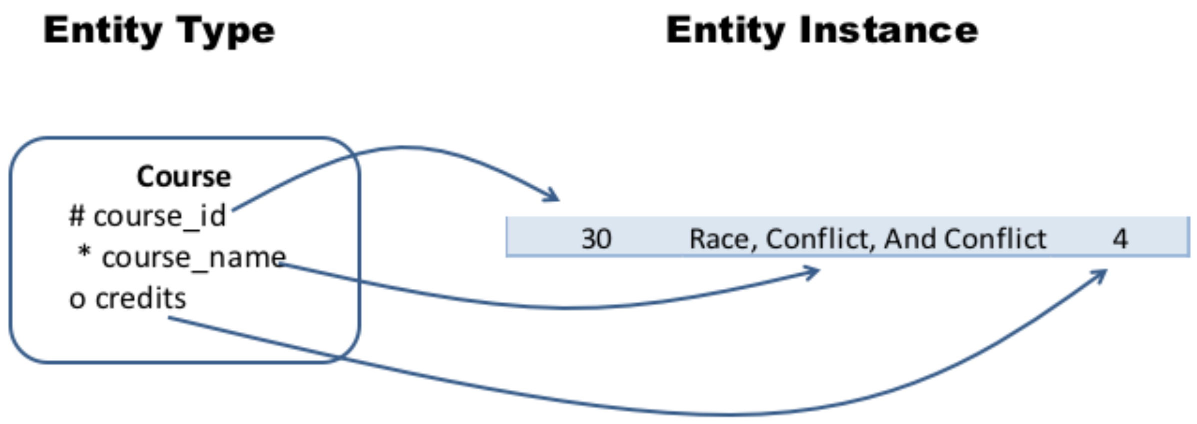
course_id	course_name	credits
10	African Paintings As A Femi	1
20	Life Of Russian Self-Actuali:	3
30	Race, Conflict, And Conflict	4
40	The Populist Dimension Of	2
50	Masterpieces Of Middle Cla	1

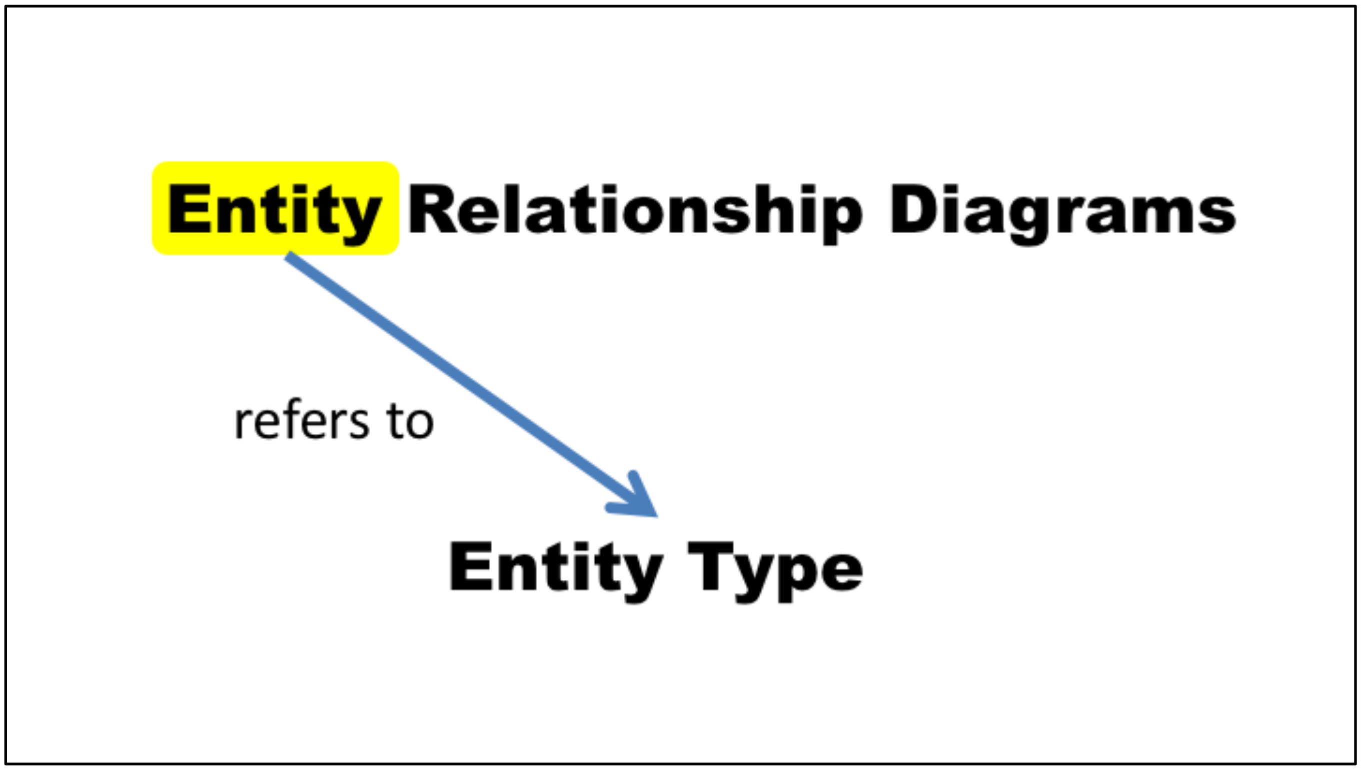


Course entity



Sometimes we do not show the attributes


Entity Type
Entity Instance
Course
# course_id
* course_name
o credits
30
Race, Conflict, And Conflict
4



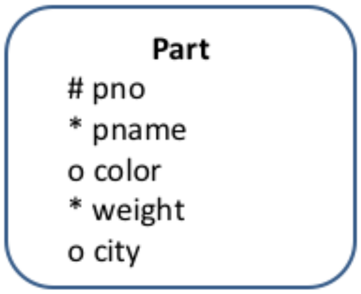
Entity Relationship Diagrams

$$\begin{tikzcd}\text{Entity} \arrow[d, "\text{refers to}"'] \\ \text{Entity Type}\end{tikzcd}$$

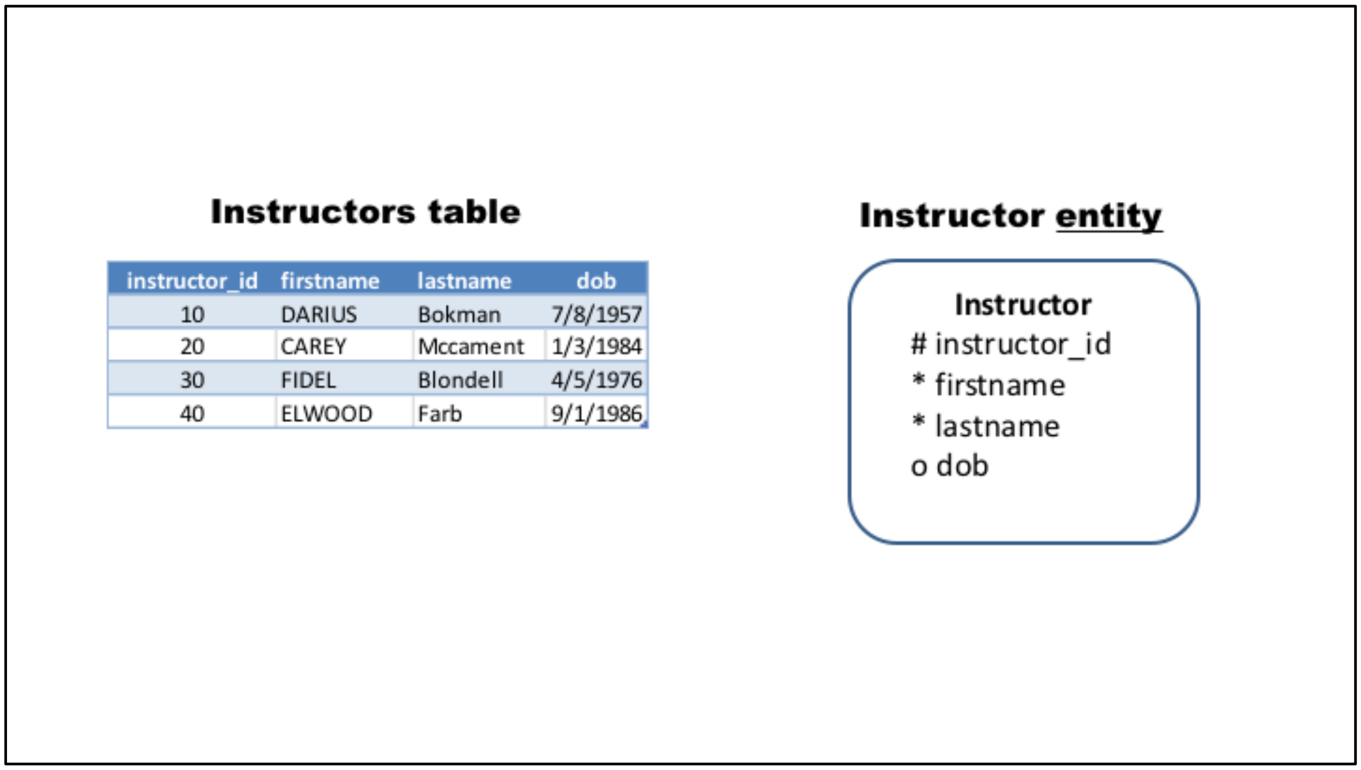
Your turn: Draw the entity corresponding to the parts table shown below. Pause the video, do the work and only then proceed. (Make your own assumptions regarding required and optional attributes.)

pno	pname	color	weight	city
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Parts table



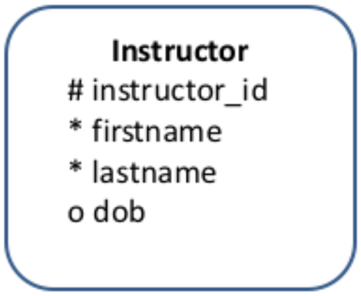
Part entity

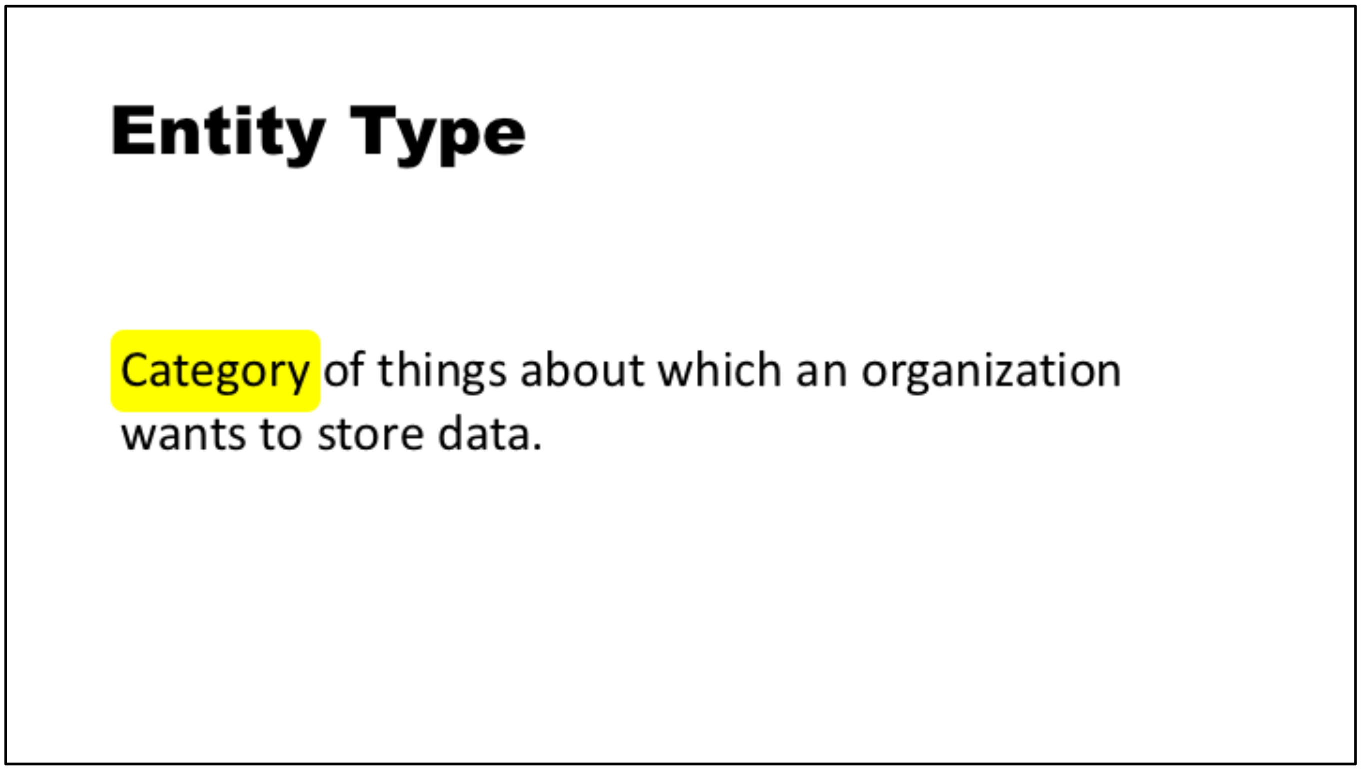


Instructors table

instructor_id	firstname	lastname	dob
10	DARIUS	Bokman	7/8/1957
20	CAREY	Mccament	1/3/1984
30	FIDEL	Blondell	4/5/1976
40	ELWOOD	Farb	9/1/1986

Instructor entity





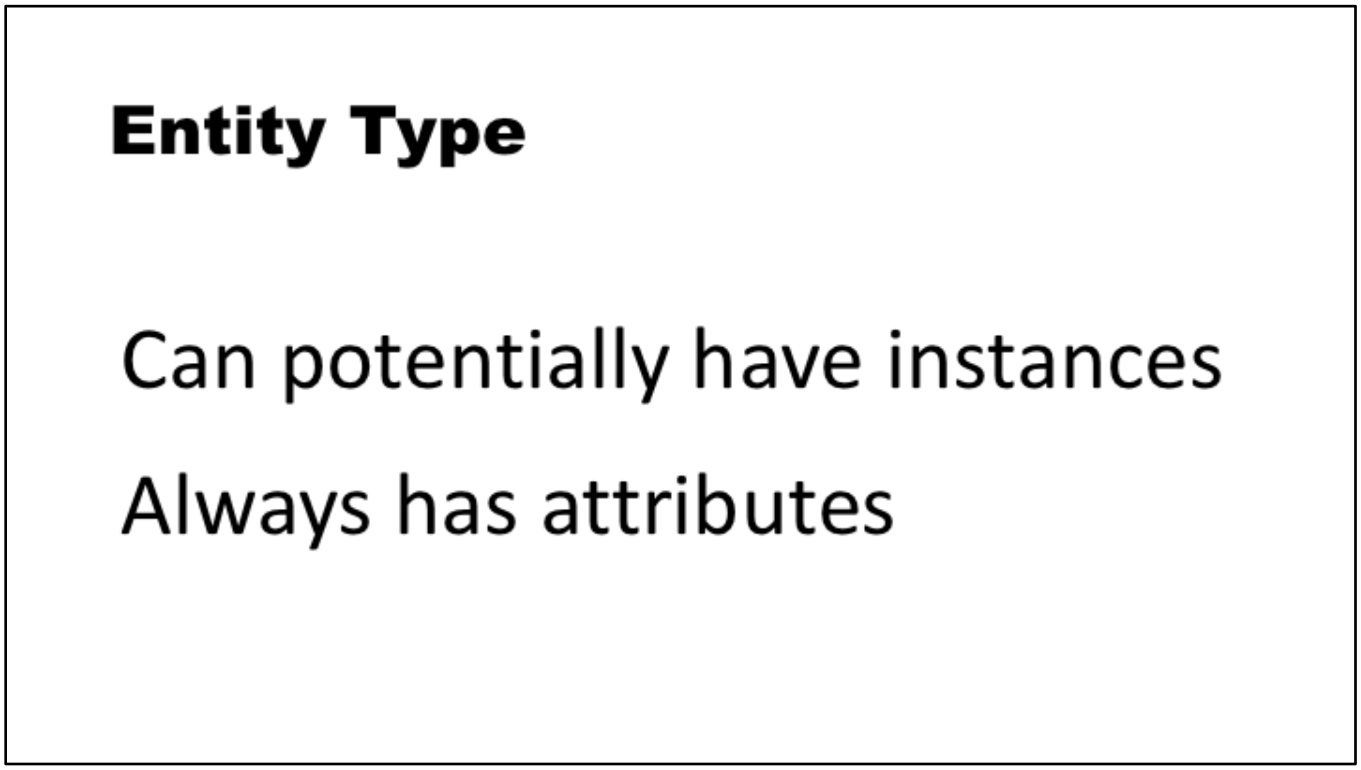
Entity Type

Category of things about which an organization wants to store data.

An entity type represents a category or concept. You should pay special attention to this fact. Entity types differ from entity instances.

The concept "Person" is an entity type. Specific persons like you and I are instances of this concept.

In this sense an entity type is abstract and instances are concrete.



Entity Type

Can potentially have instances

Always has attributes

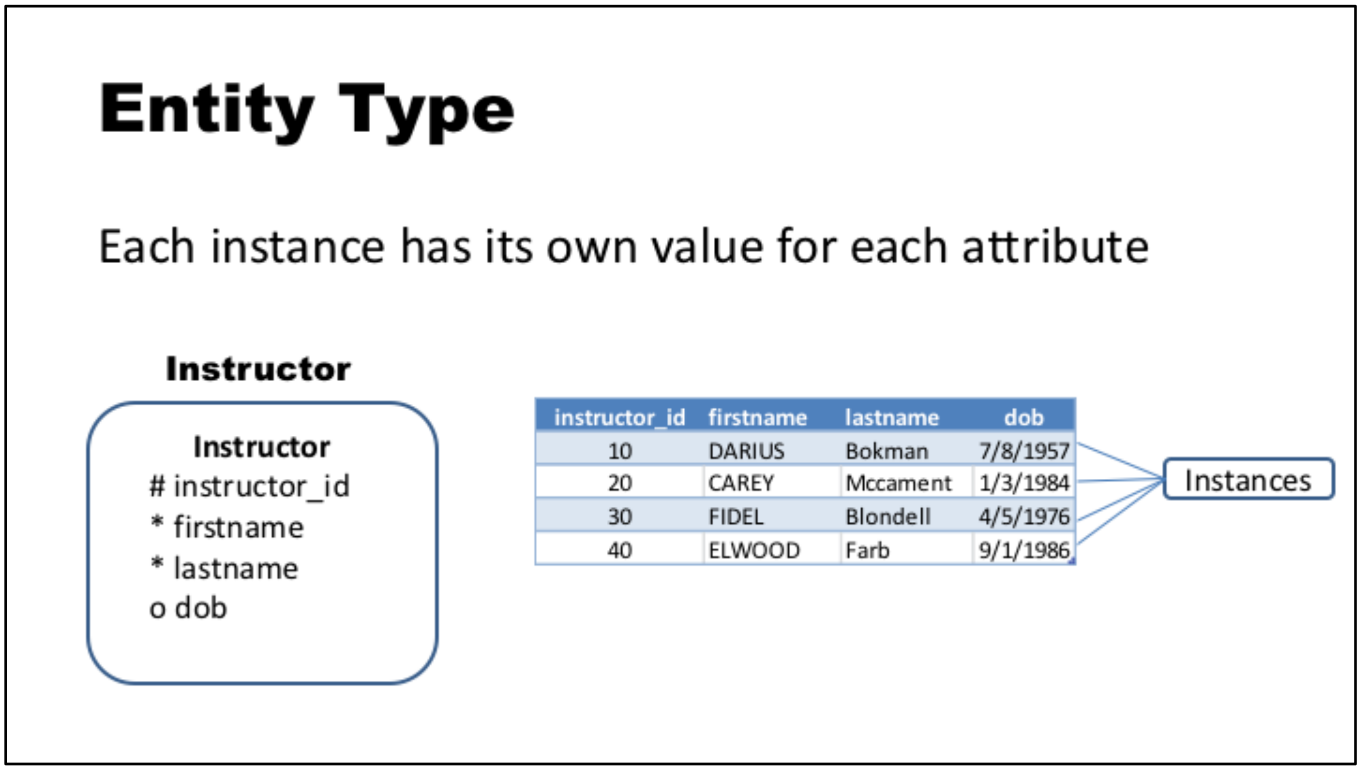
If you come across something that cannot have instances, then it cannot be an entity type – it could be an entity instance.

For example, consider a specific customer like customer A. Can we have instances of customer A? No. Customer A is customer A and does not represent a concept. Customer A is a specific, concrete customer.

On the other hand "Customer" is a concept and we could have several instances of the concept.

Entity types can have attributes. Each instance of an entity type will have its own values for each of the attributes.

In this sense you can also see an entity type as a template describing whatever is common to a family of objects or things.



Entity Type

Each instance has its own value for each attribute

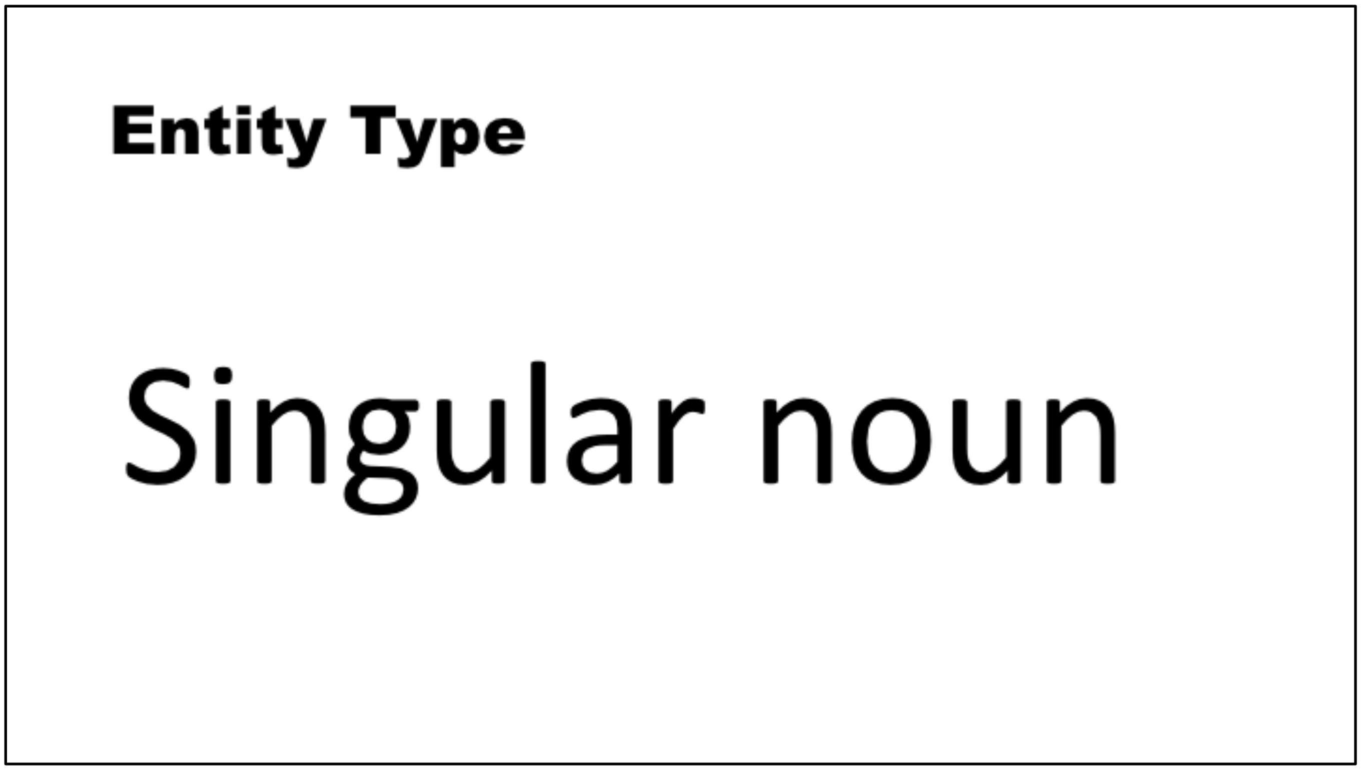
Instructor

instructor_id	firstname	lastname	dob
10	DARIUS	Bokman	7/8/1957
20	CAREY	Mccament	1/3/1984
30	FIDEL	Blondell	4/5/1976
40	ELWOOD	Farb	9/1/1986

The slide shows the entity type “Instructor” on the left.

On the right we see a table containing data about four instances of the entity type.

Note how each instance has its own values for each of the attributes like instructor_id, firstname, lastname and dob.

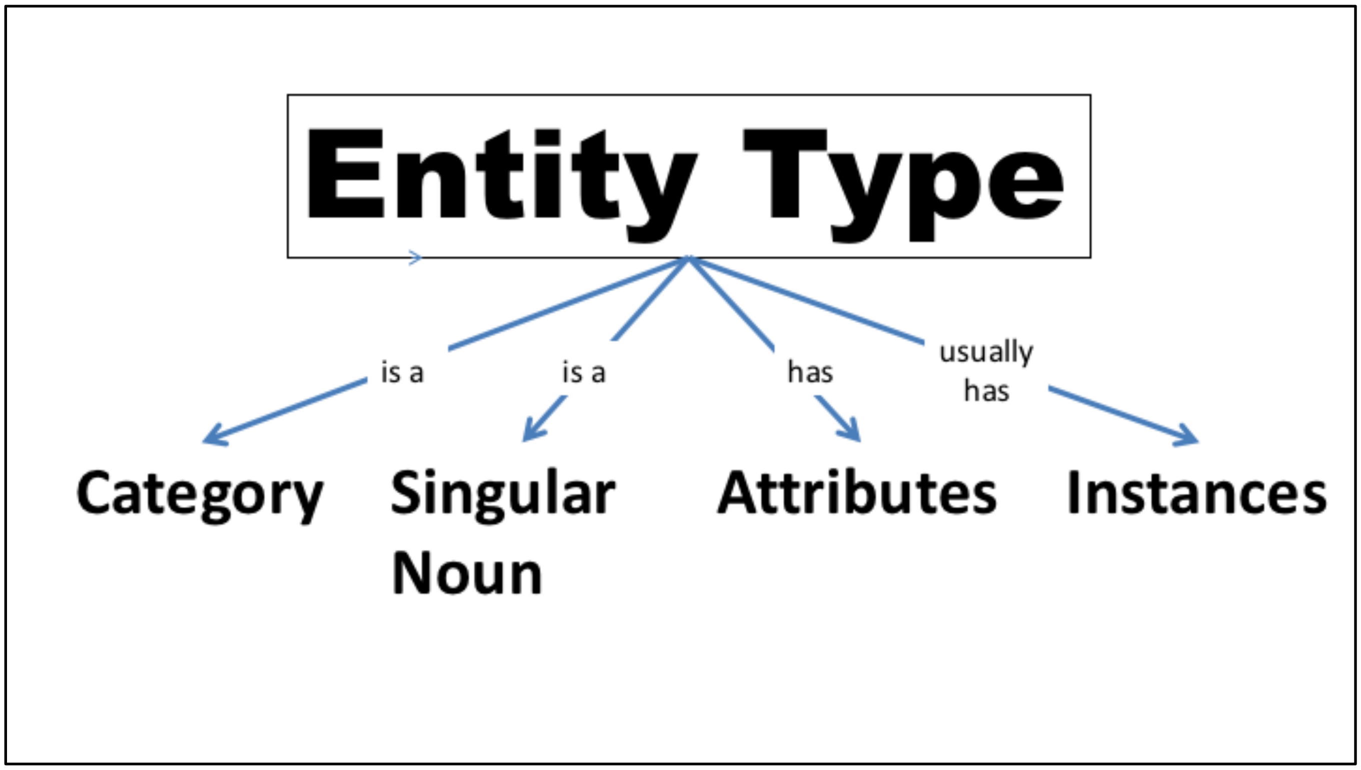


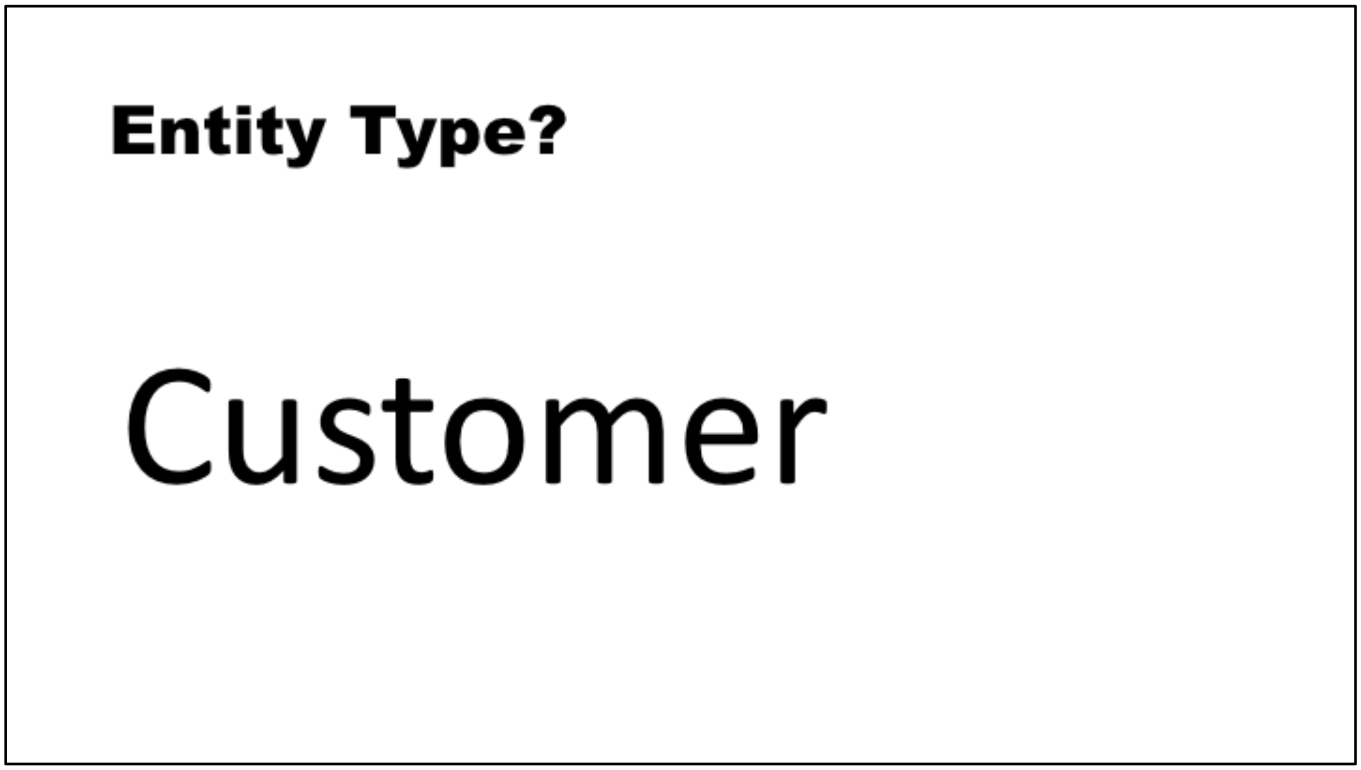
Entity Type

Singular noun

Since an entity type represents a category or concept, it is always expressed as a singular noun.

Do not think of an entity type as a collection of instances. That misunderstanding can hamper you a lot as you try to learn ER diagramming.

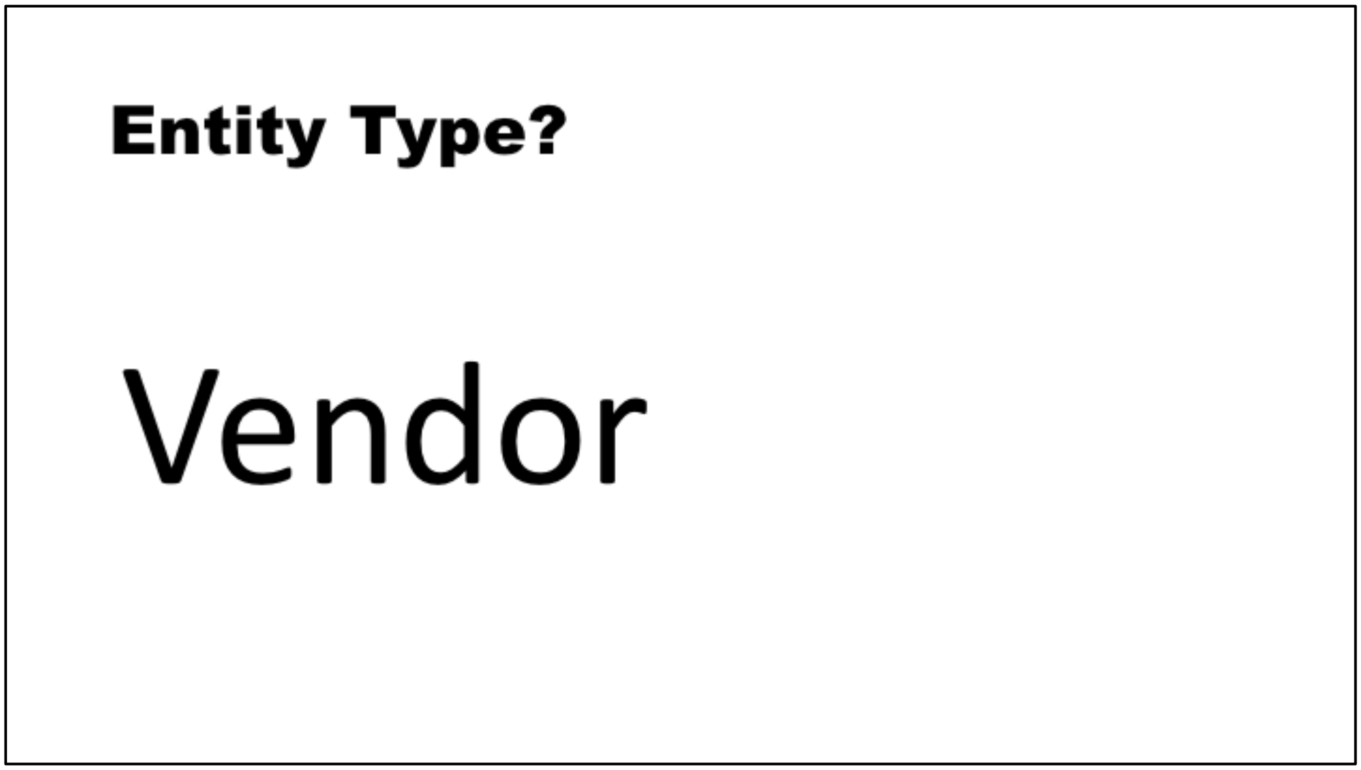

Entity Type
is a
is a
has
usually has
Category
Singular Noun
Attributes
Instances



Entity Type?

Customer

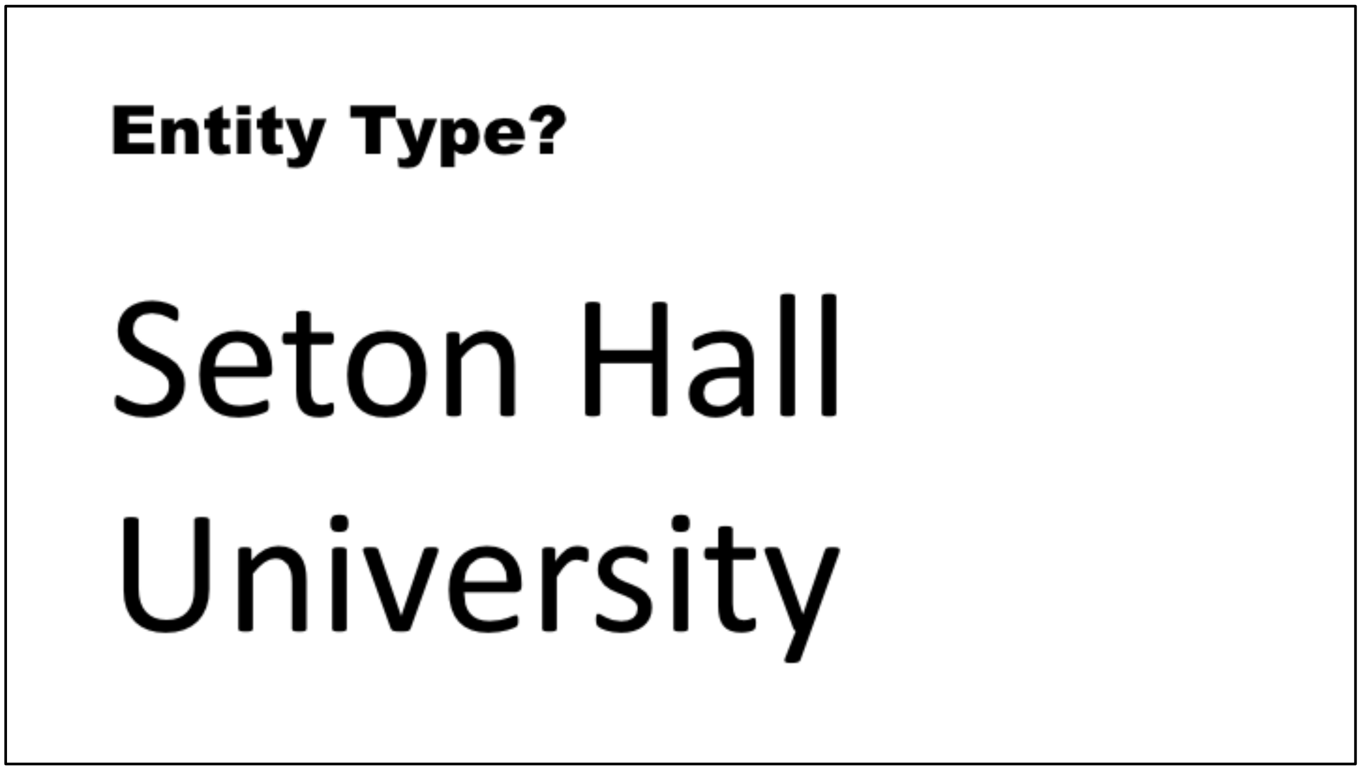
Represents a category and can have instances, for example, customer A, customer B, etc.
Can have attributes like name, address, etc.
Can be an entity type.



Entity Type?

Vendor

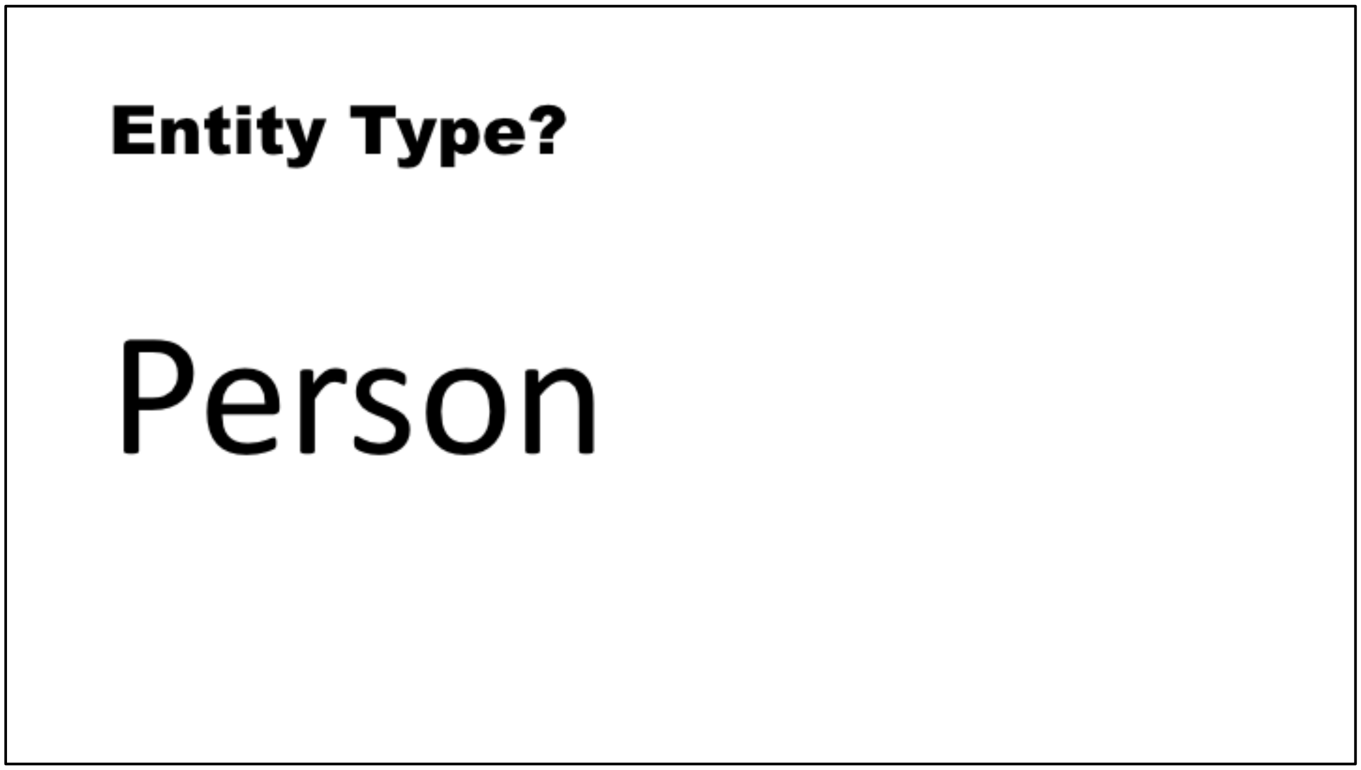
Represents a category and can have instances like vendor A, vendor B, etc.
Can have attributes like name, address, etc.
Can be an entity type.



Entity Type?

Seton Hall University

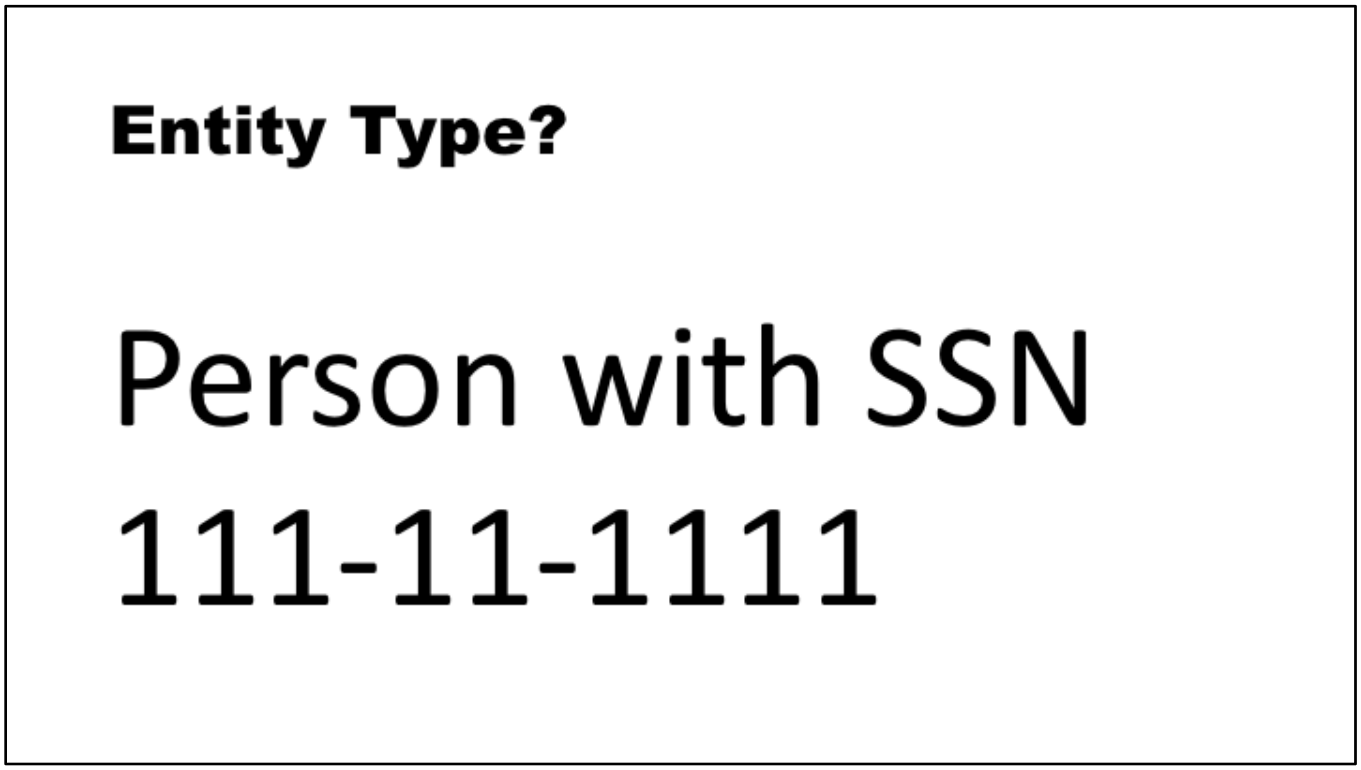
Does not represent a category because it cannot have its own instances.
Looks like an instance of an entity type like University, or Organization or some such.



Entity Type?

Person

Represents a category and can have instances which would be specific persons like you and me.
Can have attributes like name, address, etc.
Can be an entity type.

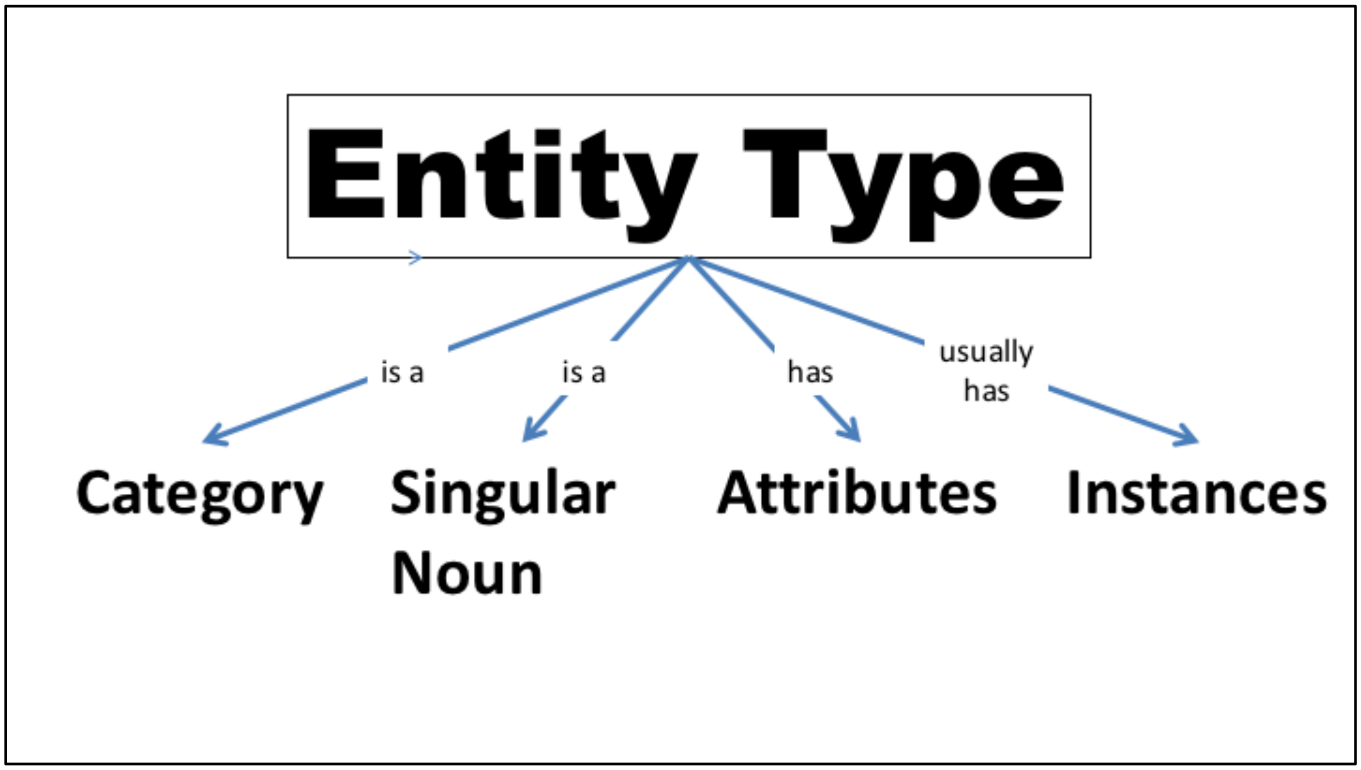


Entity Type?

Person with SSN 111-11-1111

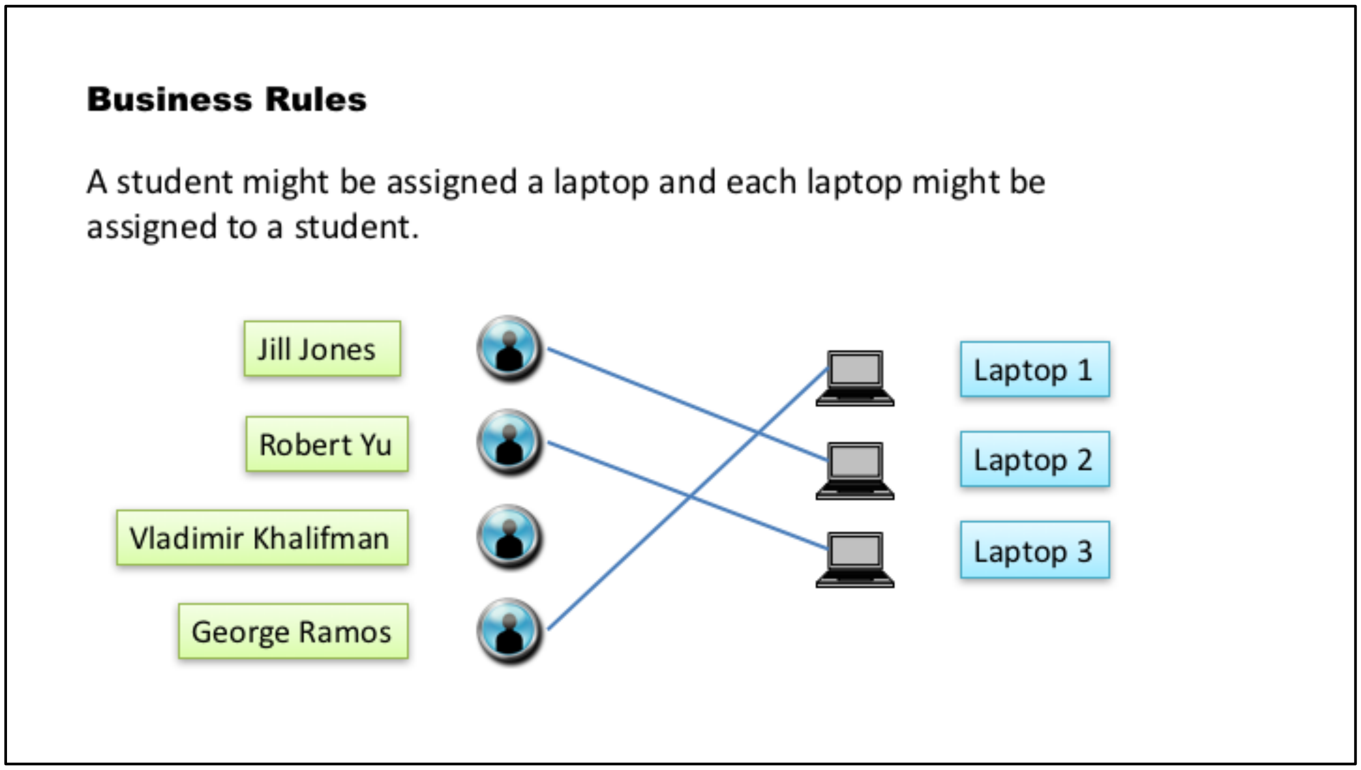
Does not represent a category and hence cannot be an entity type.

Could be an **instance** of an entity type like Person.



In any given situation, we can identify infinitely many entity types. Which ones should we actually model?

For example, while studying an organization, we will surely come across entity types like chair, table, insect, light and so on. Whether we include these in our model or not depends on whether the business wants to track these as part of its operations. While mostly quite clear, we will encounter situations where this decision could be subjective.



Business Rules

A student might be assigned a laptop and each laptop might be assigned to a student.

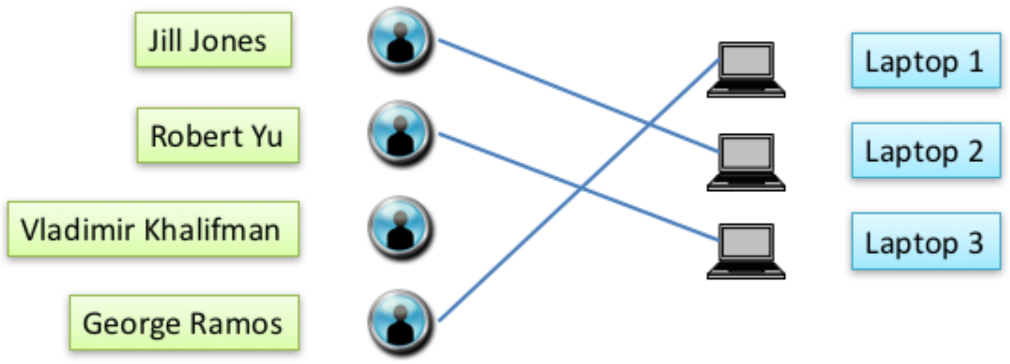


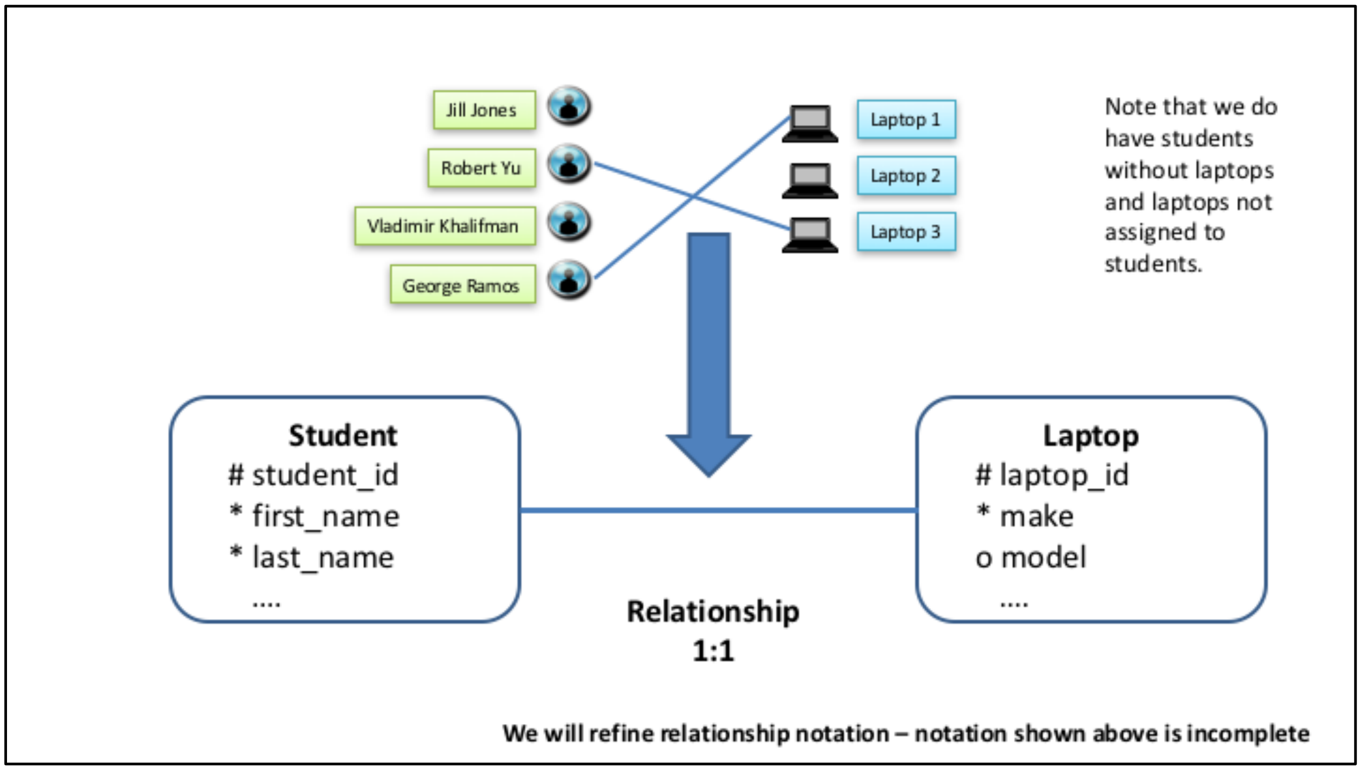
Figure shows instances of entity type student and shows possible relationships between those and instances of entity type laptop.

The above is not an ER diagram. It does not show entity types, instead it deals with entity instances.

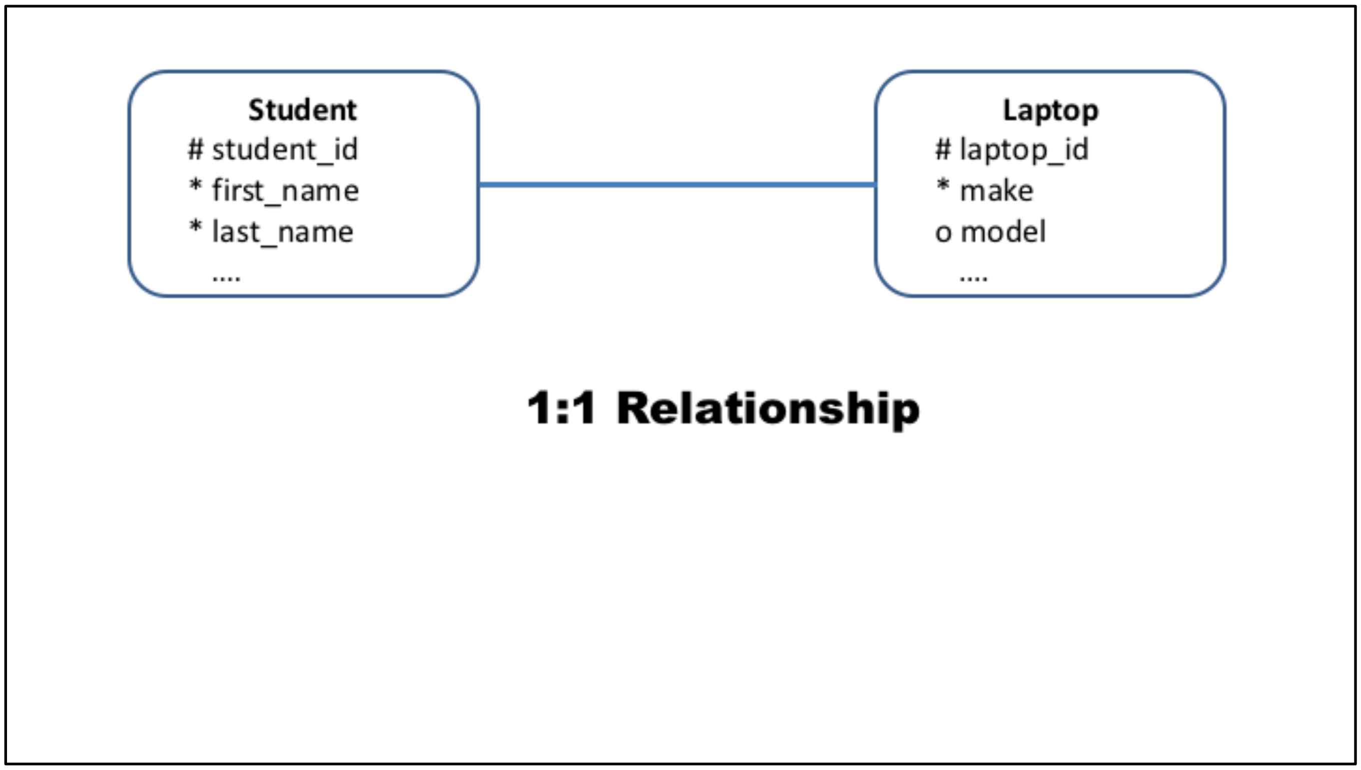
Relationship

Relationships capture the various ways in which entities could be related in the situation under consideration.

As with entity types, we can potentially identify infinitely many relationships. Which ones we include in our model depends on what the organization considers significant.



Note the correspondence between the figure showing connections between entity instances and the more general ER diagram showing the generic relationship between the two entity types.



Since a student can have at most one laptop and a laptop can be assigned to at most one student, this is a 1:1 relationship.

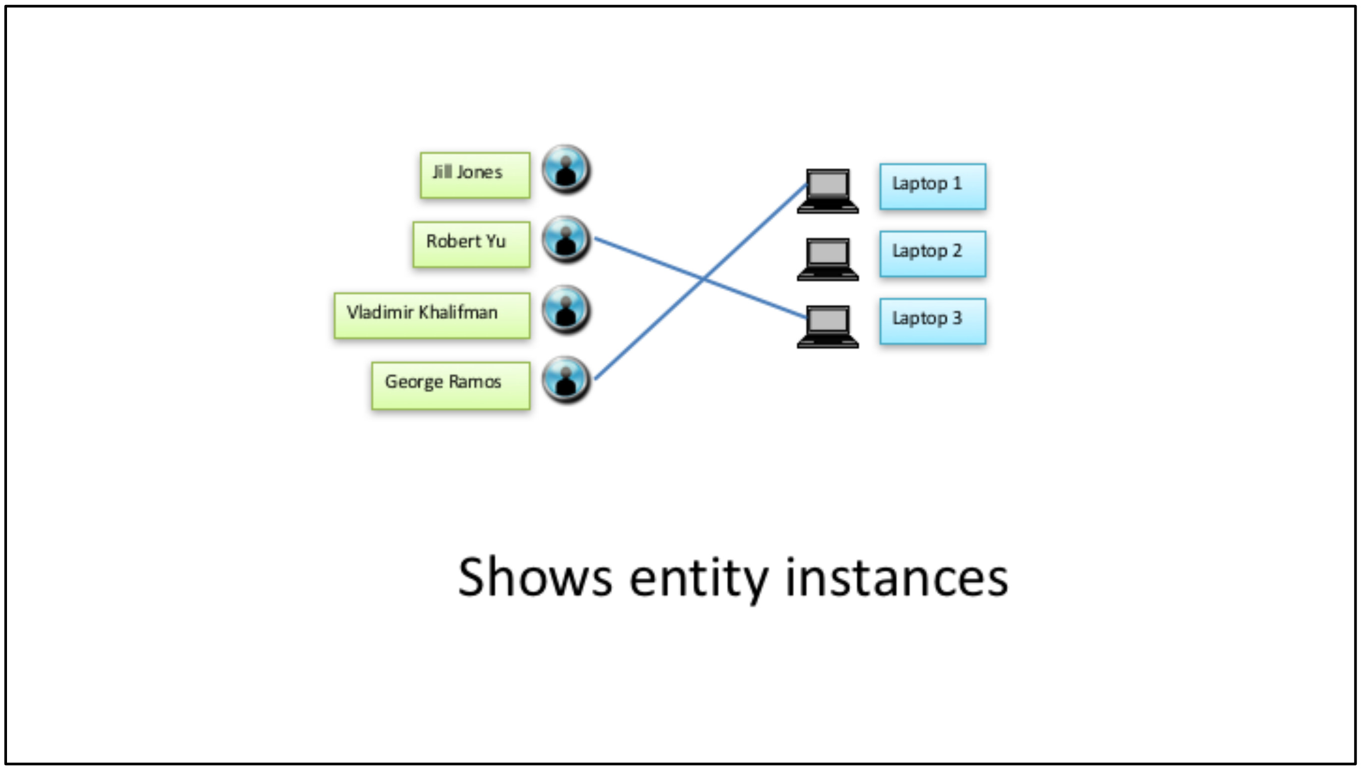
In determining the degree of a relationship, we consider each entity type in turn and ask at most how many instances of the other entity type the first one can be related to. We are concerned only with whether the answer is “one” or “many” – if the answer is more than 1, then the specific number does not matter, it is just “many.”

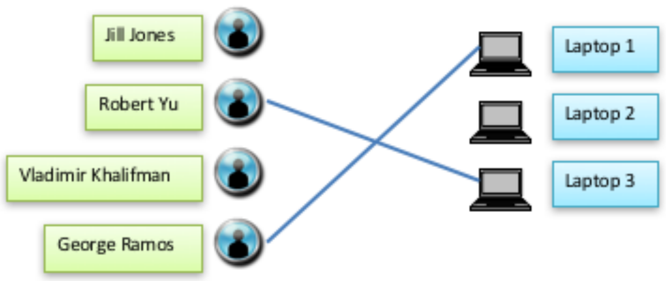
In the above situation we ask:

Each student can be associated with at most how many laptops? Our earlier description indicated that a student might be assigned a laptop. Thus the upper limit is 1.

Each laptop can be assigned to at most how many students? Again we see that the description indicates that this is also 1.

Therefore the relationship is 1:1.

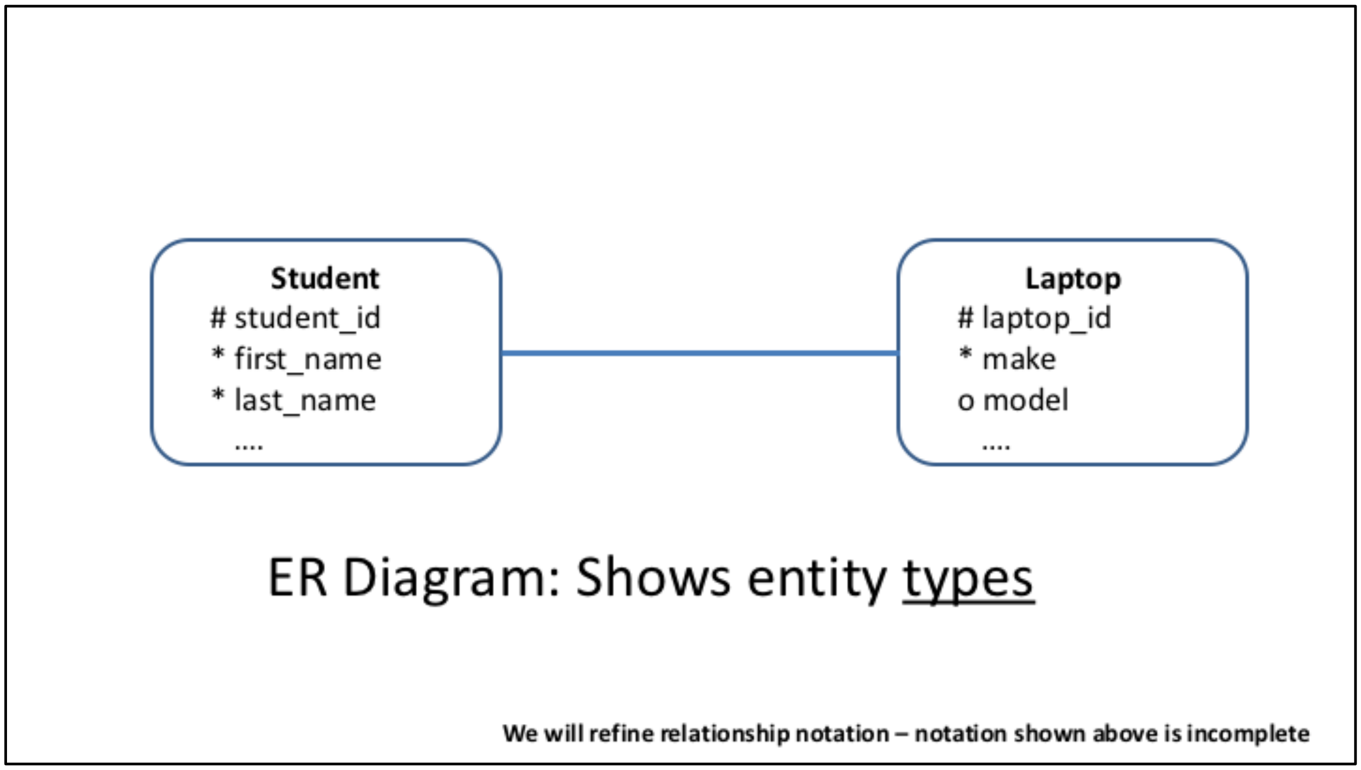


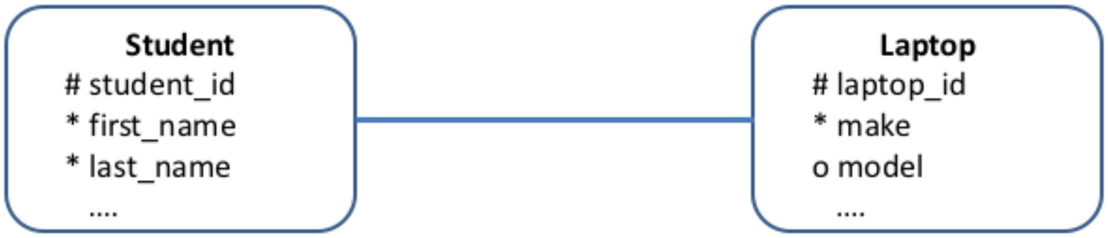


Shows entity instances

The above figure represents instances. It is not an ER diagram and I have shown it only to clarify the difference between ER diagrams and those that show specific instances.

ER diagrams show only entity types. As an enterprise operates, specific instances come and go – new customers are added, old ones are removed and so on – but the entity type "customer" stays on.

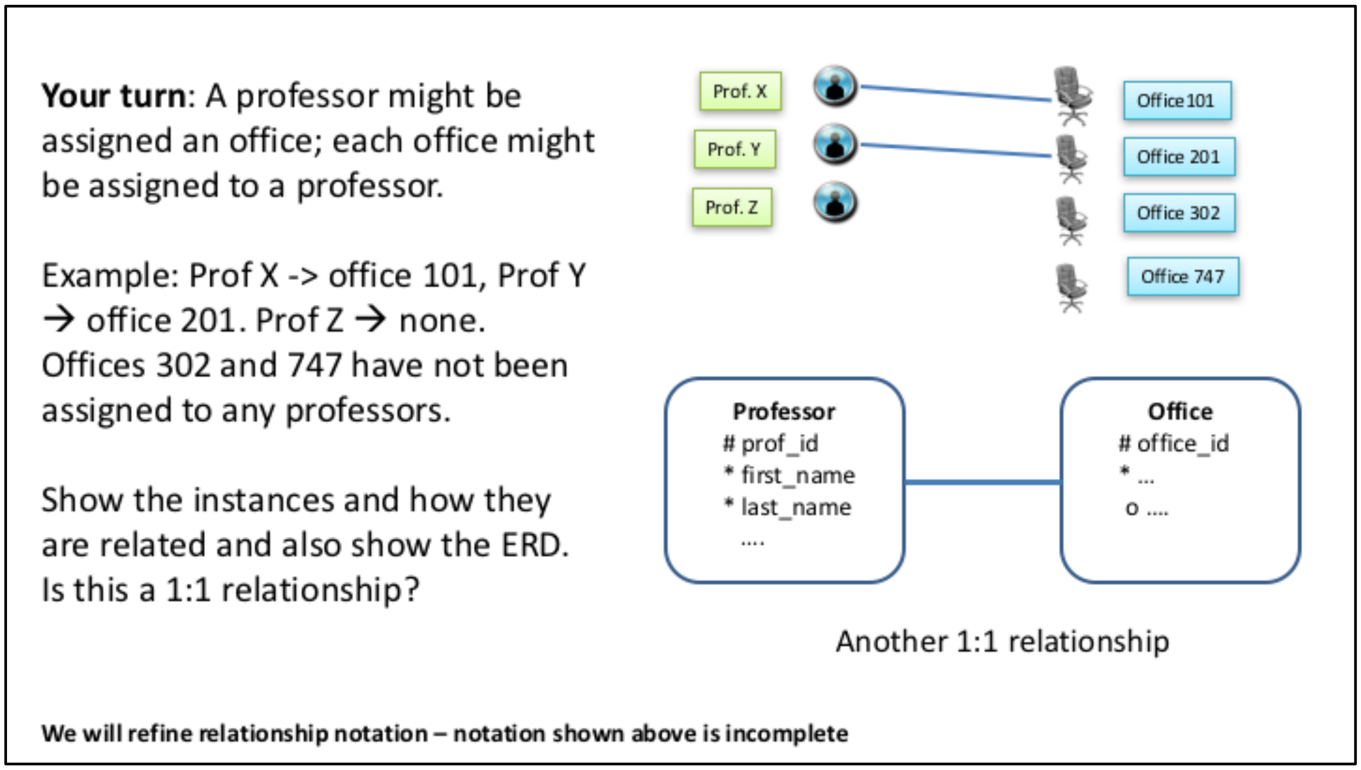




ER Diagram: Shows entity <u>types</u>

We will refine relationship notation – notation shown above is incomplete

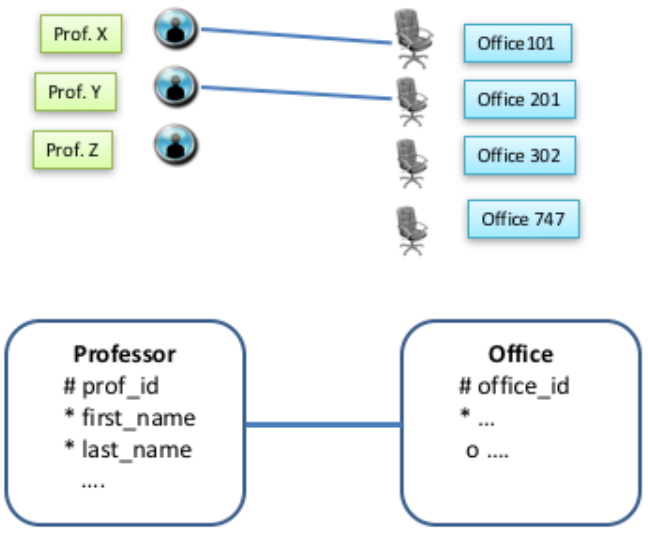
The above is an ER diagram – although the relationship notation has not yet been refined. We will discuss additional nuances of the notation shortly.



Your turn: A professor might be assigned an office; each office might be assigned to a professor.

Example: Prof X -> office 101, Prof Y → office 201. Prof Z → none. Offices 302 and 747 have not been assigned to any professors.

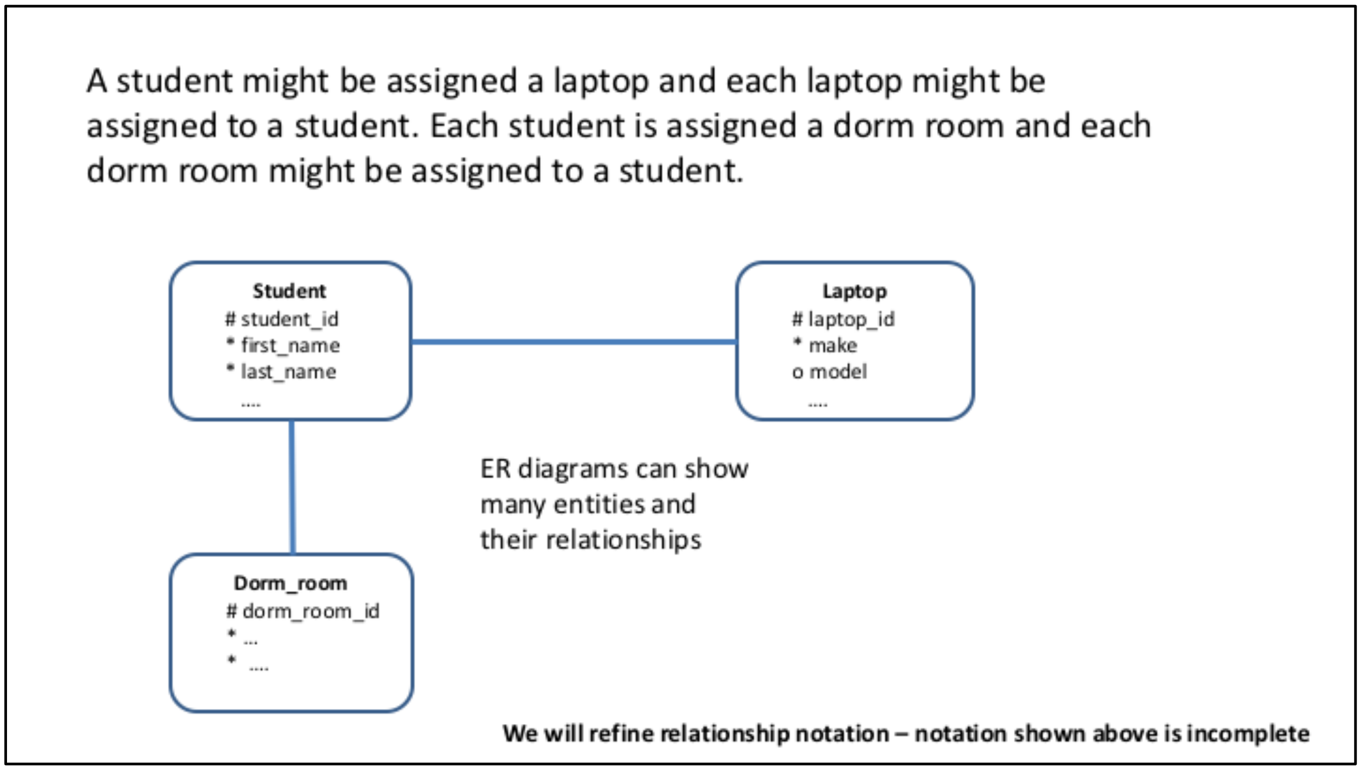
Show the instances and how they are related and also show the ERD. Is this a 1:1 relationship?



Another 1:1 relationship

We will refine relationship notation – notation shown above is incomplete

ERDs show relationships between entity types by connecting related entity types by a line. The above slide does not show the full picture yet. We will refine the notation as we progress.



A student might be assigned a laptop and each laptop might be assigned to a student. Each student is assigned a dorm room and each dorm room might be assigned to a student.

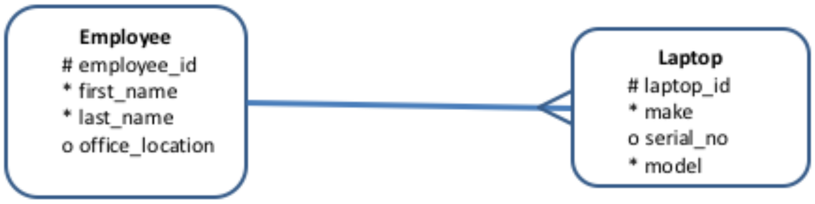
ER diagrams can show many entities and their relationships

We will refine relationship notation – notation shown above is incomplete

An ERD shows many entity types of interest to an organization, and depicts their relationships.

You can therefore expect to see that many entity types have relationships with many others, as with Student above, which relates to Laptop and Dormroom.

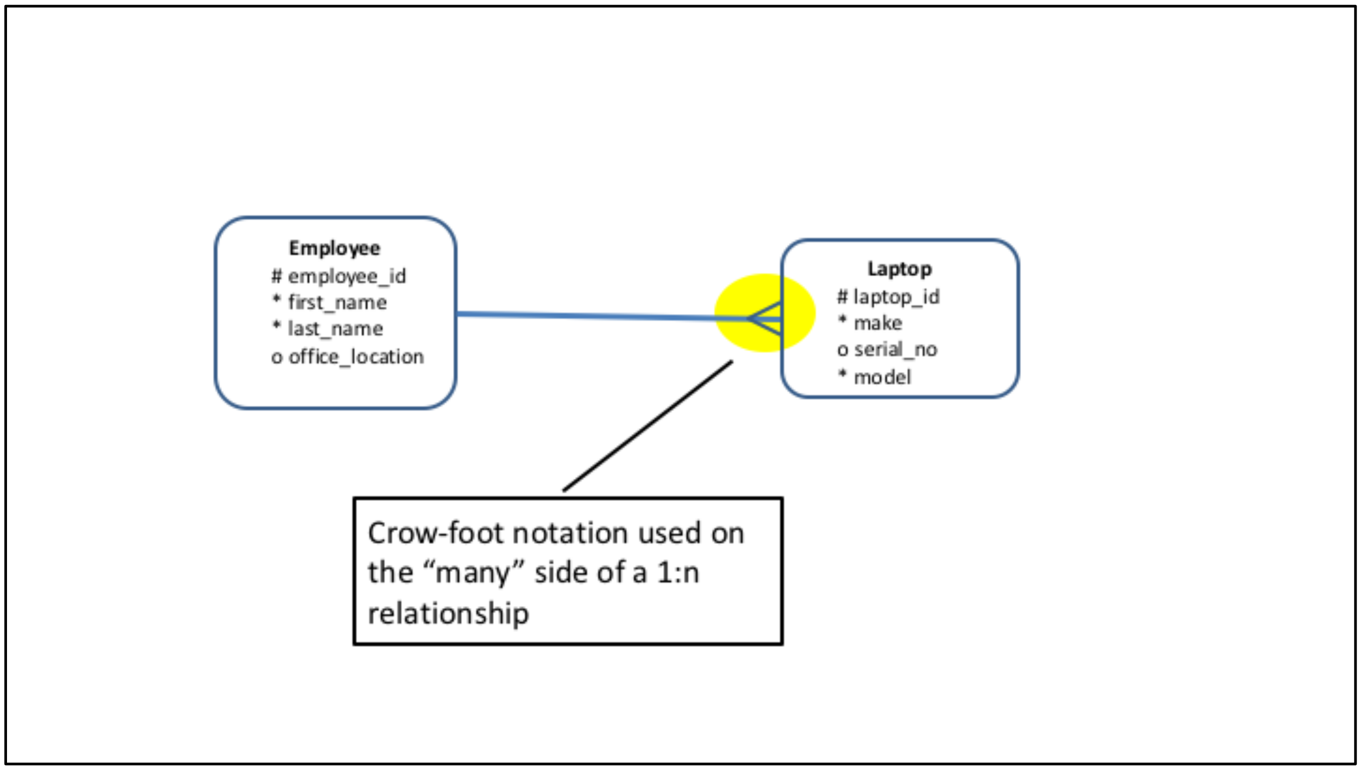
Each laptop might be assigned to an employee for maintenance and each employee might be assigned to maintain many laptops.



1:many or 1:n relationship

We will refine relationship notation – notation shown above is incomplete

In the above, we see that each laptop might be assigned to at most one employee, but that each employee might be responsible for many laptops – we thus have a relationship of degree 1:n.

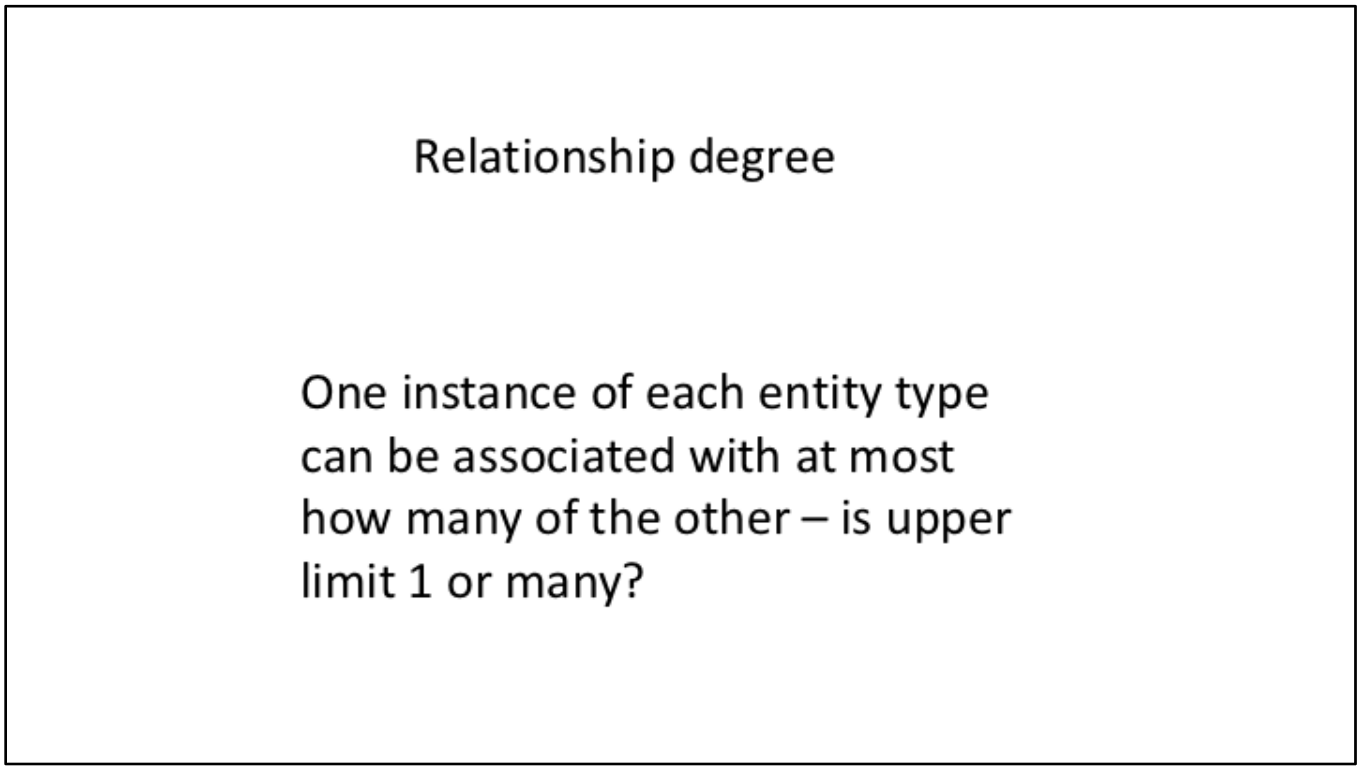


We use the crow-foot notation on the “many” side of the 1:n relationship.

Since one employee can be associated with many laptops, we have the crow-foot on the side of Laptop.

Since each laptop can be associated with at most one employee, we do not have a crow-foot on Employee.

Be sure you get the logic of the placement of the crow-foot.



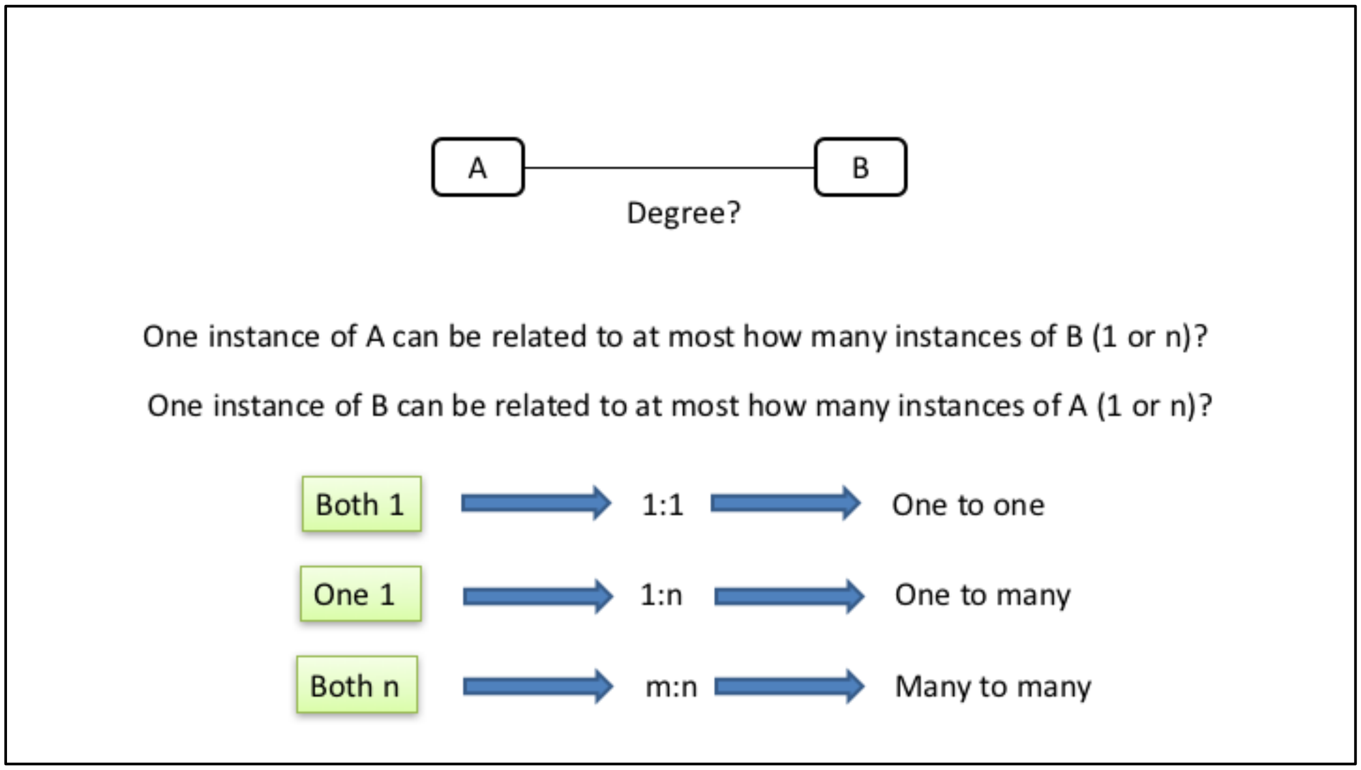
Relationship degree

One instance of each entity type can be associated with at most how many of the other – is upper limit 1 or many?

In determining the degree of relationship between entity types A and B, we are concerned with the following two questions:

One instance of A can be related to at most how many of B: 1 or n?
One instance of B can be related to at most how many of A: 1 or n?

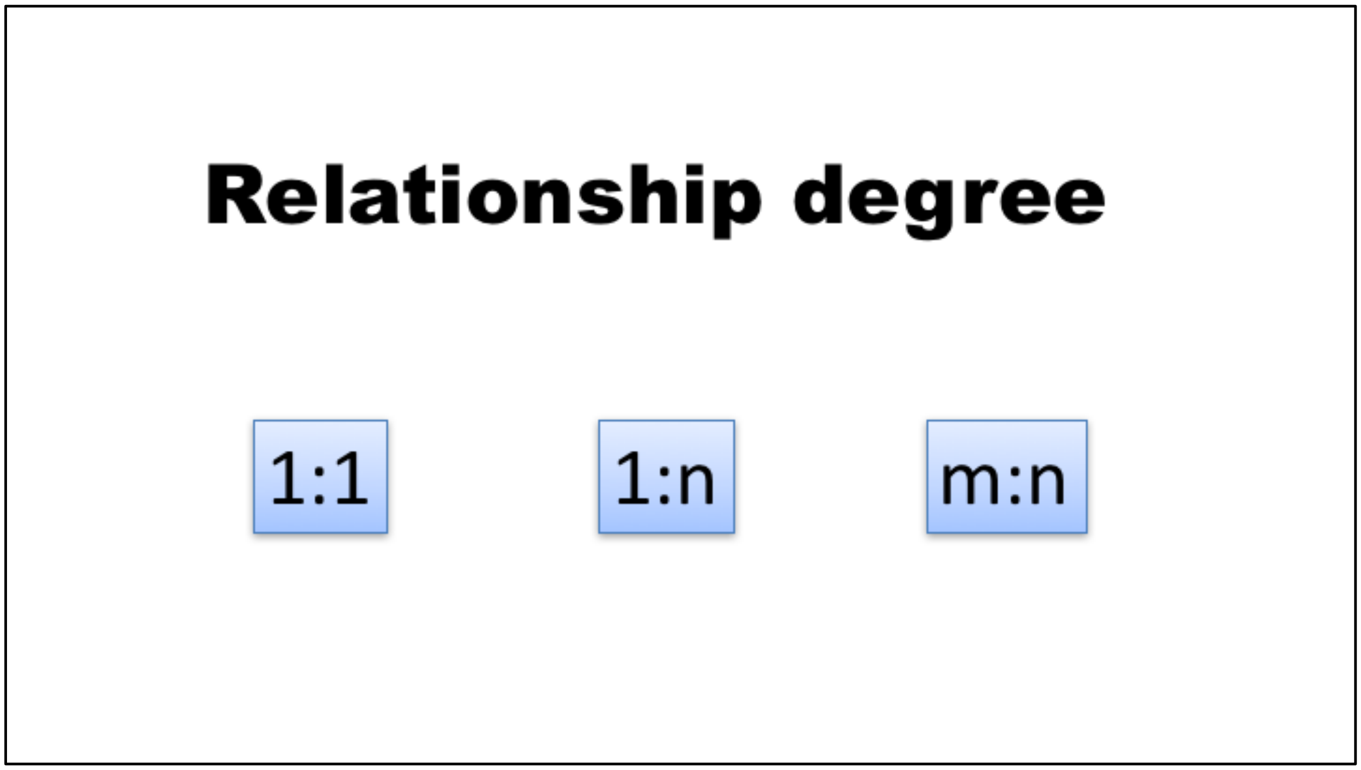
If both are 1, then we have a 1:1 relationship.
If one is 1 and the other is n then we have a 1:n relationship.
If both or n, then we have a m:n relationship.



One instance of A can be related to at most how many instances of B (1 or n)?

One instance of B can be related to at most how many instances of A (1 or n)?

Look at the relationship from both ends to determine the degree.



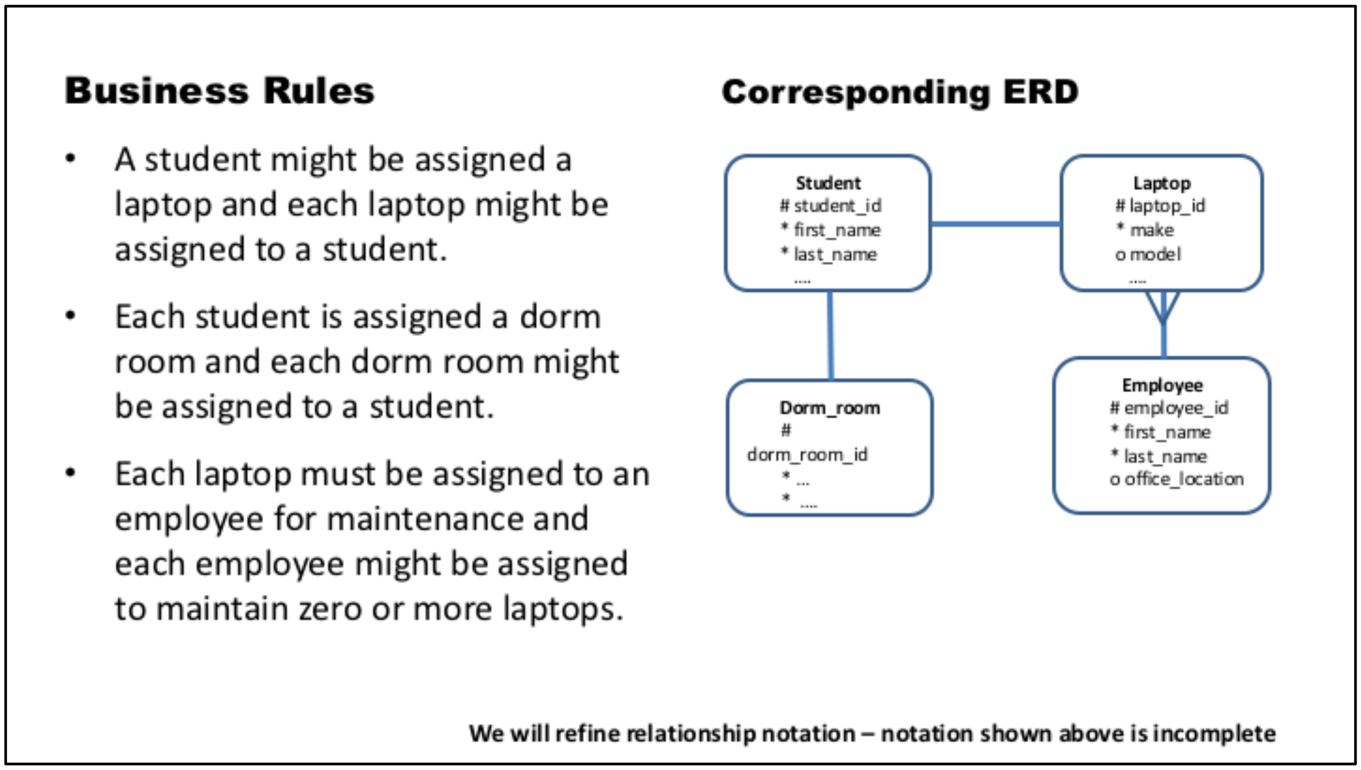
Relationship degree

1:1

1:n

m:n

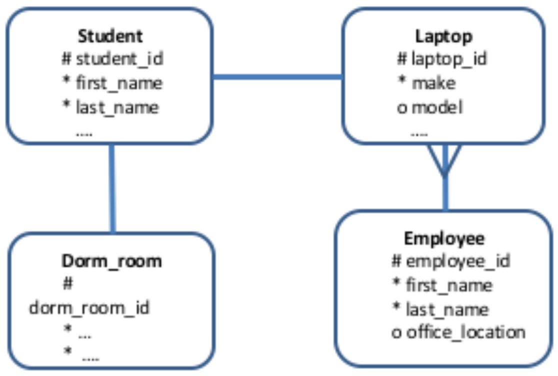
We will say more about m:n relationships later.



Business Rules

- A student might be assigned a laptop and each laptop might be assigned to a student.
- Each student is assigned a dorm room and each dorm room might be assigned to a student.
- Each laptop must be assigned to an employee for maintenance and each employee might be assigned to maintain zero or more laptops.

Corresponding ERD

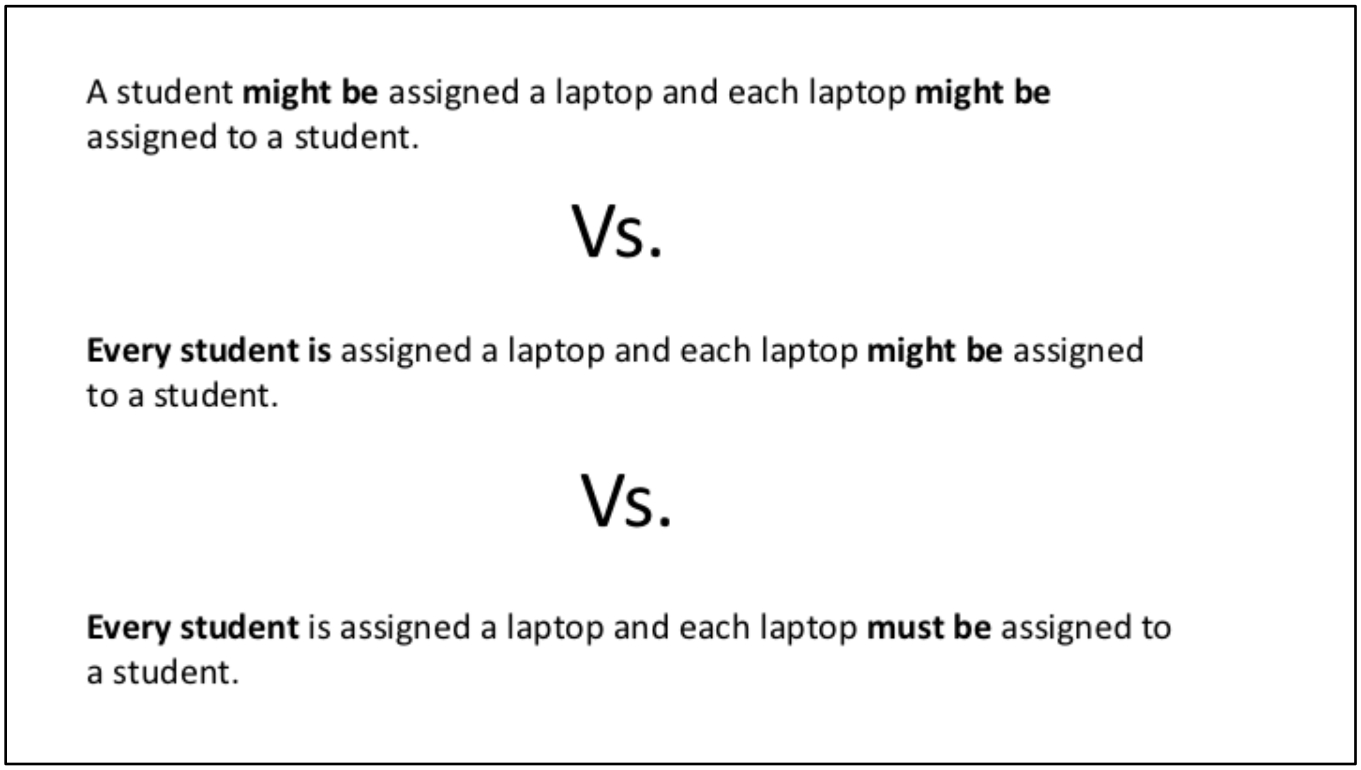


We will refine relationship notation – notation shown above is incomplete

Database design involves identifying the various entity types involved in a business situation and capturing the relationships.

During this process we do not explicitly consider data retrieval requirements. We only try to unearth entities and relationships of interest. The sheer beauty of the relational database concept allows us to then use SQL to retrieve pretty much anything we need.

We first understand the business rules by talking to people who run the business and then convert these into ERDs.



A student **might be** assigned a laptop and each laptop **might be** assigned to a student.

Vs.

Every student is assigned a laptop and each laptop **might be** assigned to a student.

Vs.

Every student is assigned a laptop and each laptop **must be** assigned to a student.

We now look at another aspect of relationships – whether or not an entity type has optional or mandatory or obligatory participation in a relationship. The three situations presented above bring out the main issue.

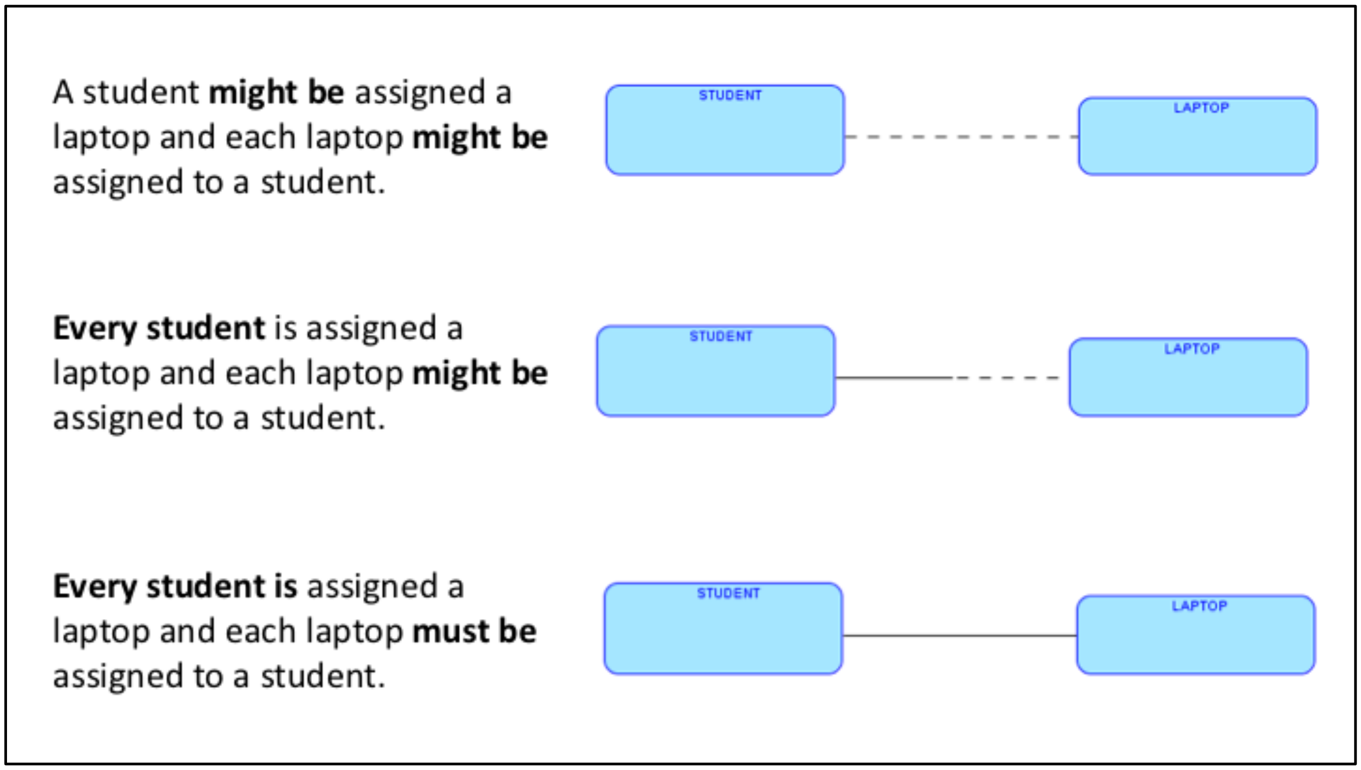
In the first sentence, we see that students and laptops have a relationship, but that the business rules allow for some students to not be related to any laptops and some laptops to not be related to any students. This means that all students and all laptops are not obliged to participate in the relationship.

That is, in general students and laptops re related, but not all students and laptops need to participate – participation is **optional**.

In the third sentence we see that all students and all laptops must participate – participation is **obligatory** or **mandatory.**

In the second sentence, we see that students must participate but laptops need not.

Note: Will not show attributes in some of the following few ERDs


A student **might be** assigned a laptop and each laptop **might be** assigned to a student.

Every student is assigned a laptop and each laptop **might be** assigned to a student.

Every student is assigned a laptop and each laptop **must be** assigned to a student.

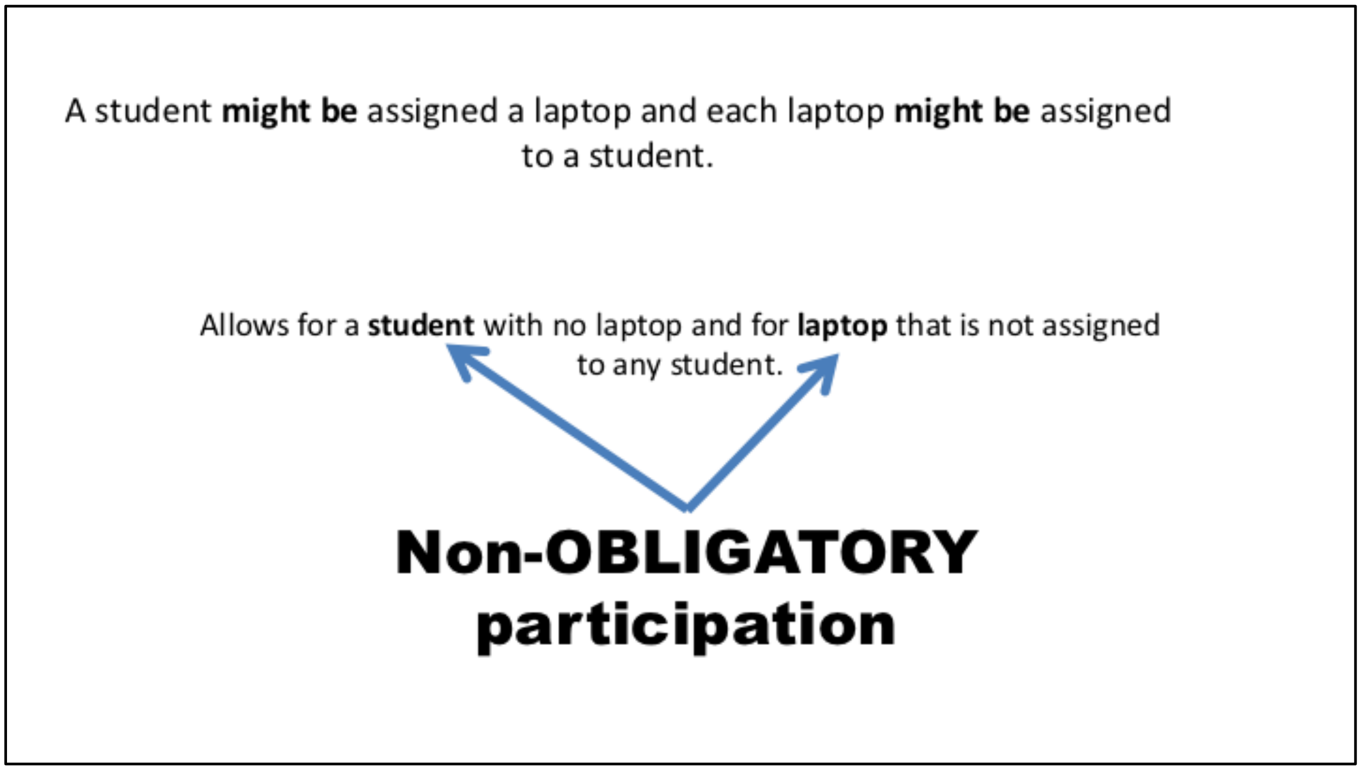
The relationship line connecting two entity types can be seen as comprising two halves – one half close to each entity type.

If an entity type has obligatory participation (that is, it must participate and cannot remain unassociated with the other entity type) on a relationship then the half line close to it on that relationship is solid. Otherwise that half line is dashed.

We look at each of the above three examples in the following slides.

When we say that a student need not necessarily be related to a laptop, we are really answering the question: “is it possible that at some point in time w could have an instance of student that is unrelated to any instance of laptop?” If this is true, we have optional participation.

On the other hand, if at no point in time were it allowed for a student instance to be unrelated to a laptop instance, then we have obligatory participation.

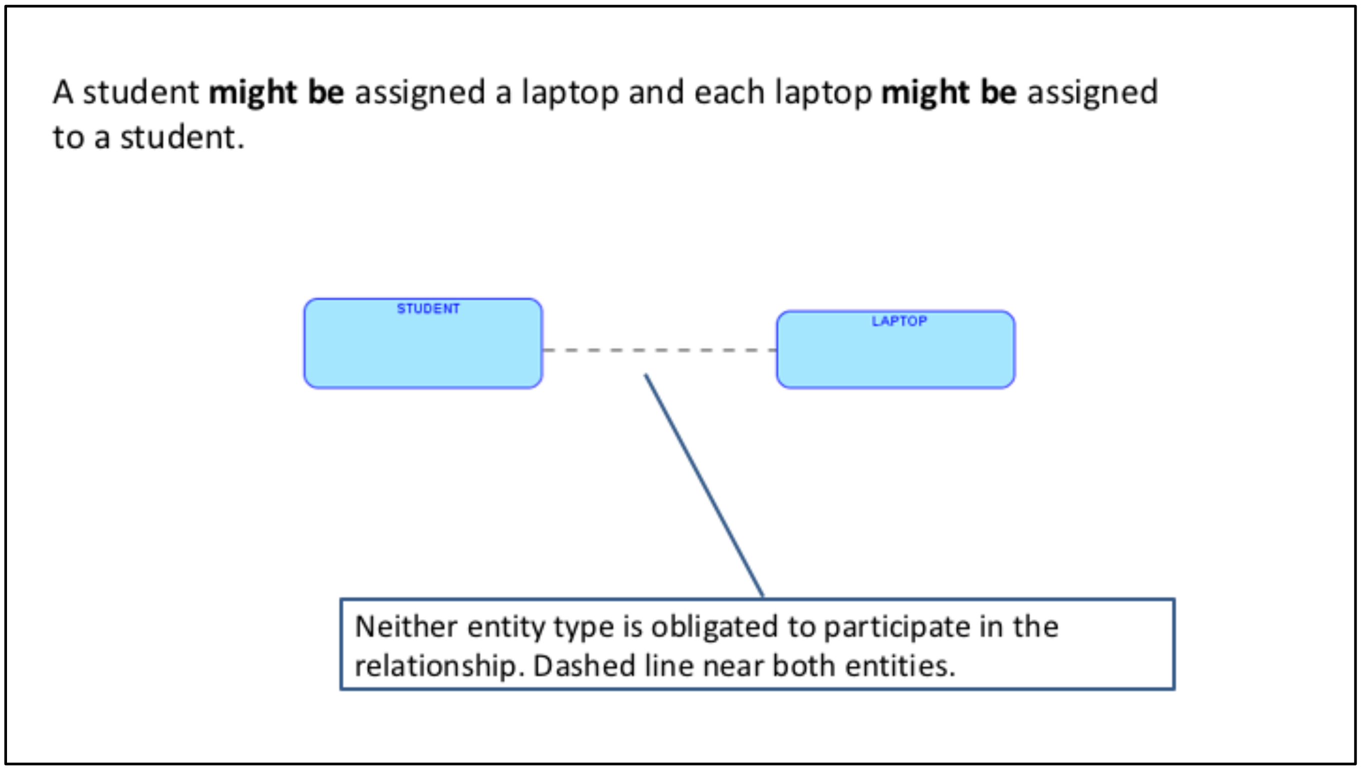

A student **might be** assigned a laptop and each laptop **might be** assigned to a student.

Allows for a **student** with no laptop and for **laptop** that is not assigned to any student.

Non-OBLIGATORY participation

When we have two entity types that have a relationship – like **student** and **laptop** above, we have many possibilities for business rules controlling their participation in the relationship.

The business rule on this slide says that although some instances of student and laptop might be associated, we could still have instances of both of these entity types which are not associated with any instance of the other. That is, these entity types have NON-OBLIGATORY participation in this relationship.



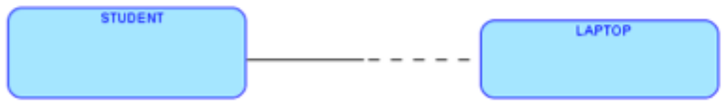
A student **might be** assigned a laptop and each laptop **might be** assigned to a student.

Neither entity type is obligated to participate in the relationship. Dashed line near both entities.

In practice this means that in our database we can have instances of either entity type that has no associated instance of the other entity type.

This allows us to have instances of student and laptop that are related and also allows us to have instances of either that are unrelated to the other type.

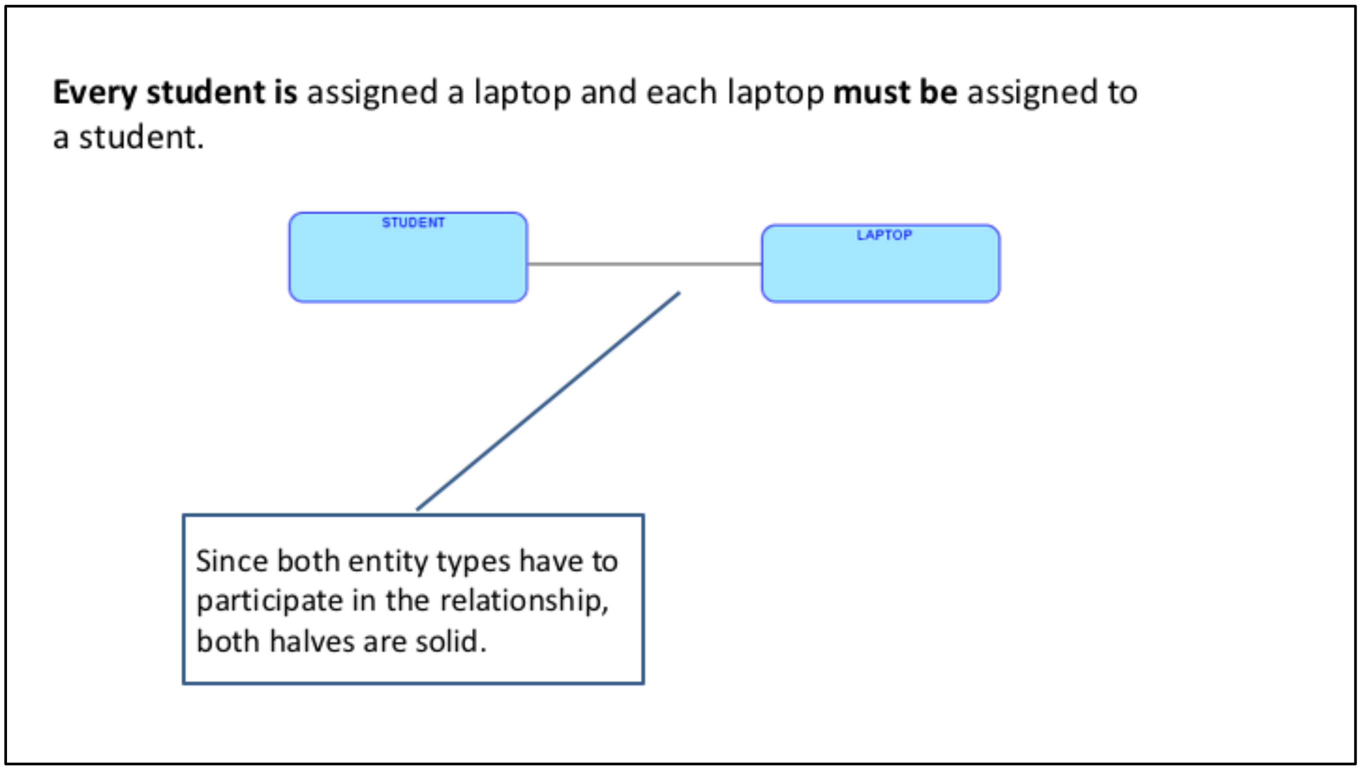
Every student is assigned a laptop and each laptop **might be** assigned to a student.



Every student has obligatory participation. Solid line near Student.

Every laptop need not necessarily participate in the relationship. Dashed line near Laptop.

In practice this means that we are not allowed to create a student instance without specifying a laptop for that student. Each and every student must at all times be associated to a laptop instance. However, we could create laptop instances that have no associated student.

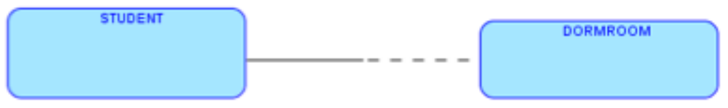


Every student is assigned a laptop and each laptop **must be** assigned to a student.

Since both entity types have to participate in the relationship, both halves are solid.

In practice this means that we cannot create a student without a laptop nor can we create a laptop without assigning it to a student.

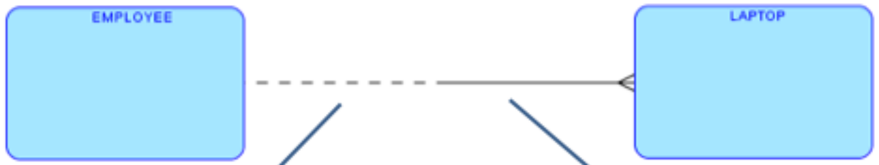
Your turn: Each student **is** assigned one dorm room and each dorm room **might be** assigned to one student.



Students must have dorm rooms assigned -- OBLIGATORY participation. Hence solid near student.

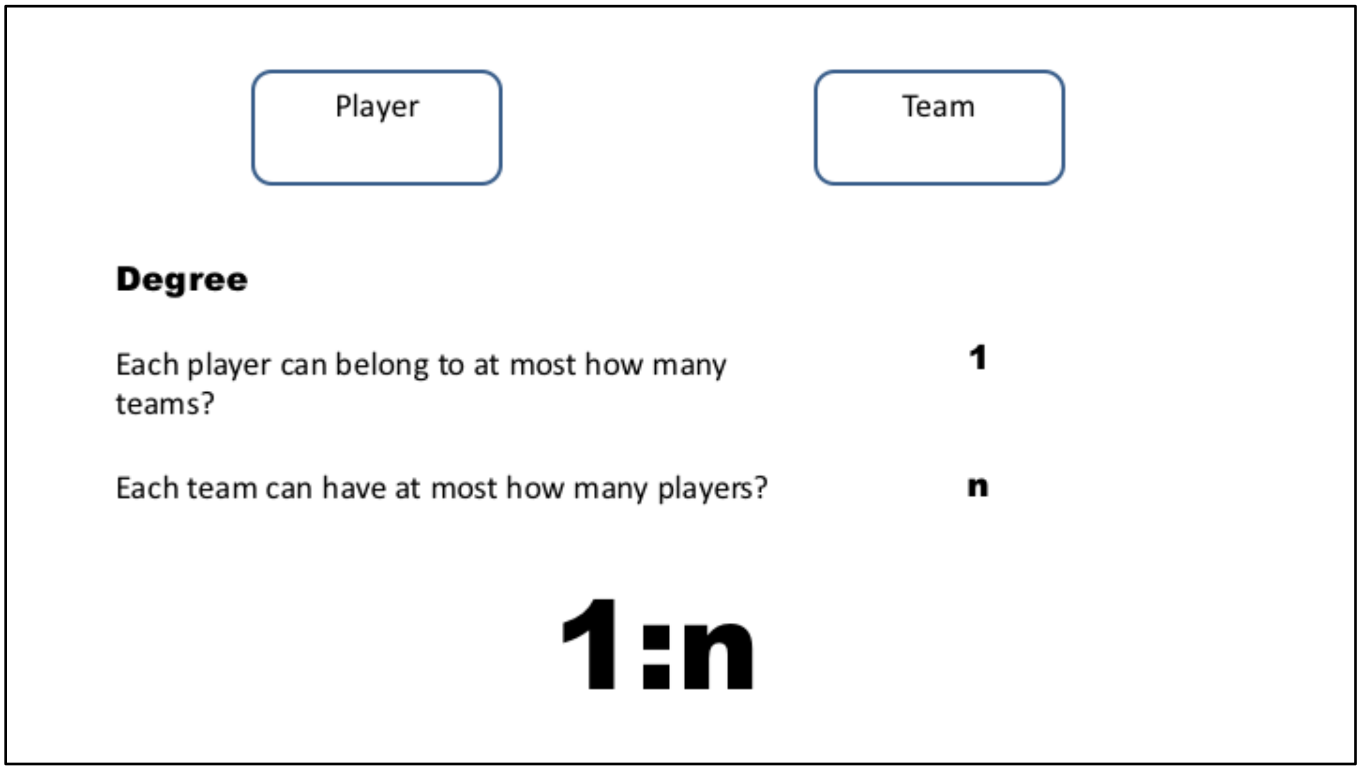
A dorm room might not necessarily be assigned to a student. Every dorm room need not participate. Hence dashed.

Your turn: Each laptop **must be** assigned to an employee for maintenance and each employee **might be** assigned to maintain zero or more laptops.



Can have employees without laptops assigned to them for maintenance – dashed line.

Every laptop must have an employee assigned for maintenance – solid line.


Player
Team

Degree

Each player can belong to at most how many teams?	**1**
Each team can have at most how many players?	**n**

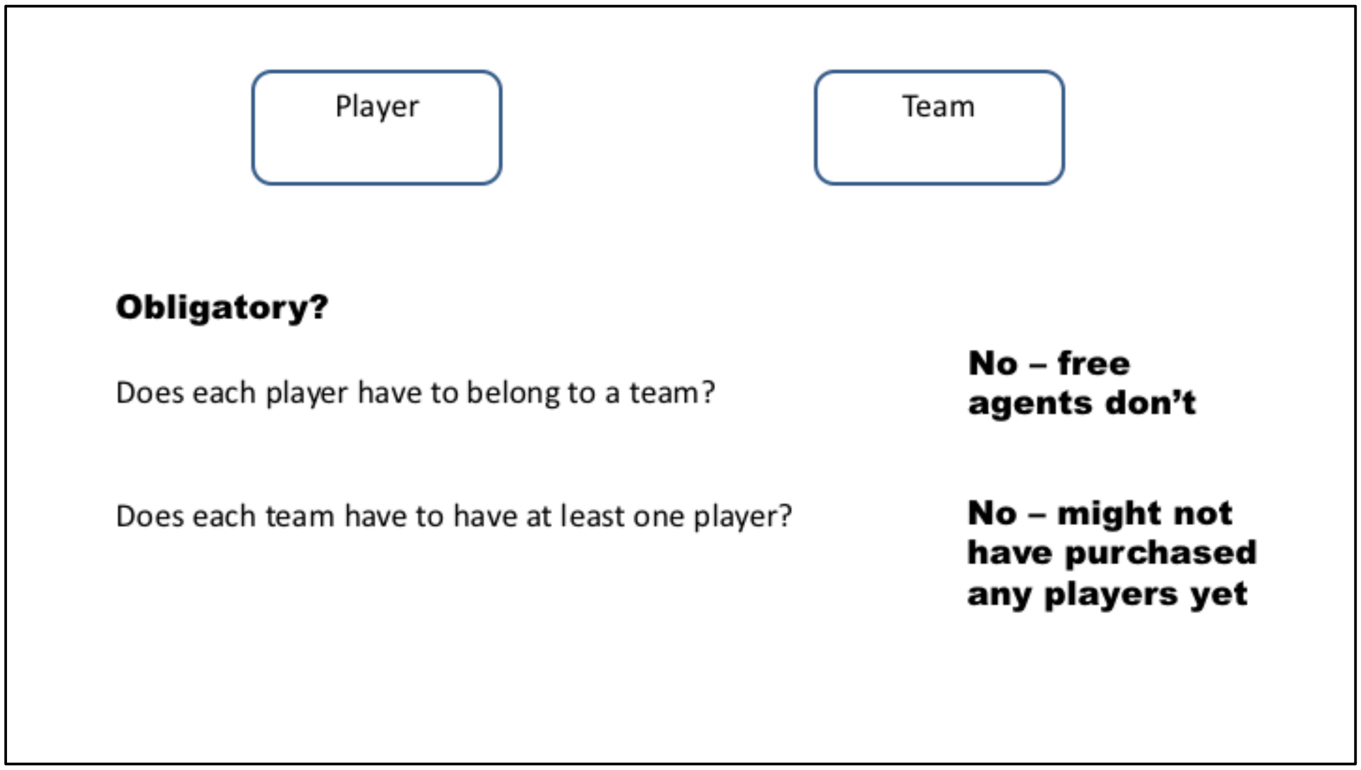
1:n

While drawing ERDs we first identify the entity types of interest. In doing this we apply the what we learned earlier about the properties of entity types.

Once we have the various entity types, we need to consider the relationships.

Properly representing a relationship requires us to identify the degree of a relationship and also the participation rules.

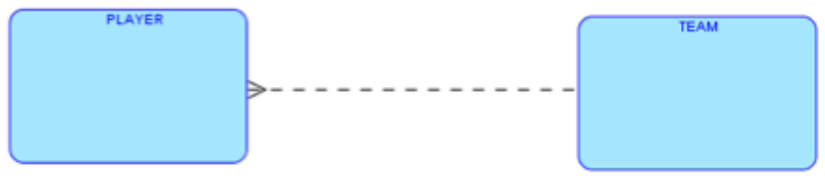
Together they determine the proper **cardinality** notation.

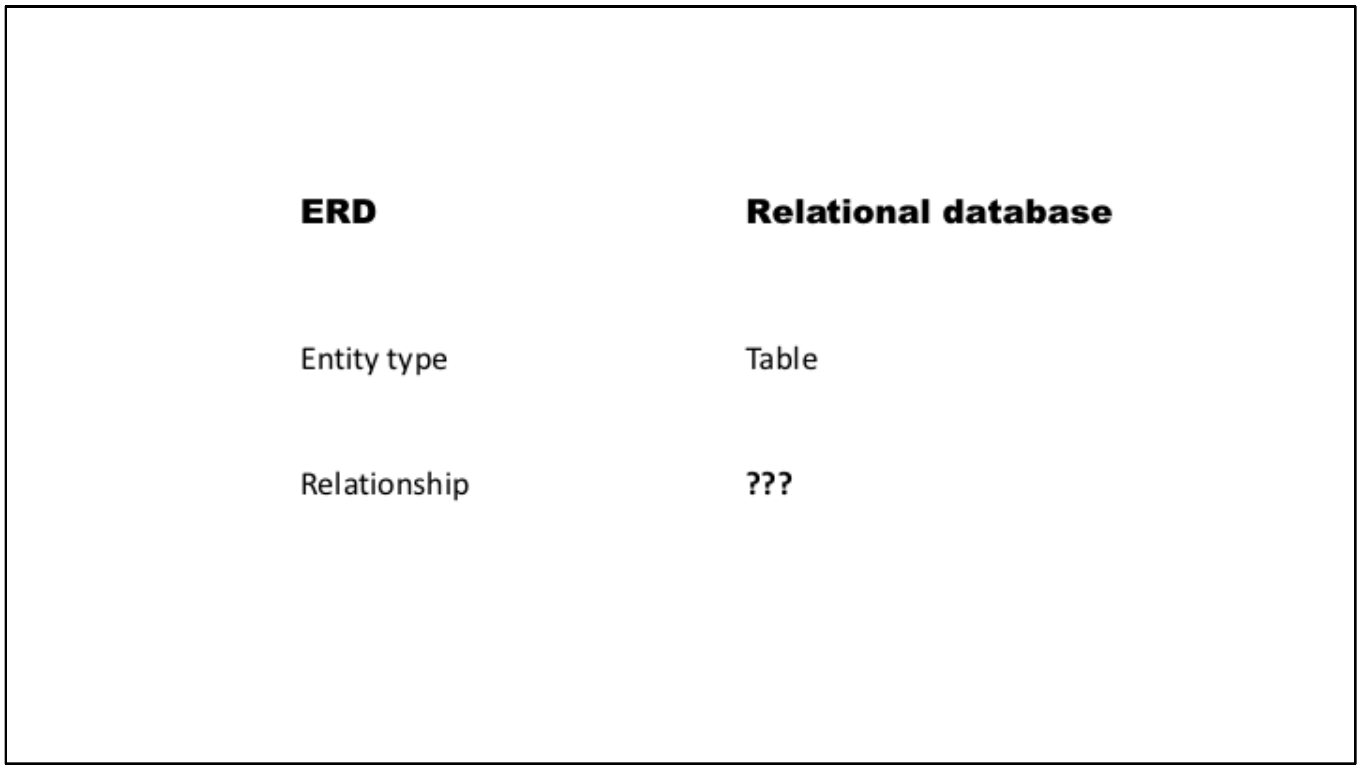

Player
Team

Obligatory?

Does each player have to belong to a team? **No – free agents don't**

Does each team have to have at least one player? **No – might not have purchased any players yet**

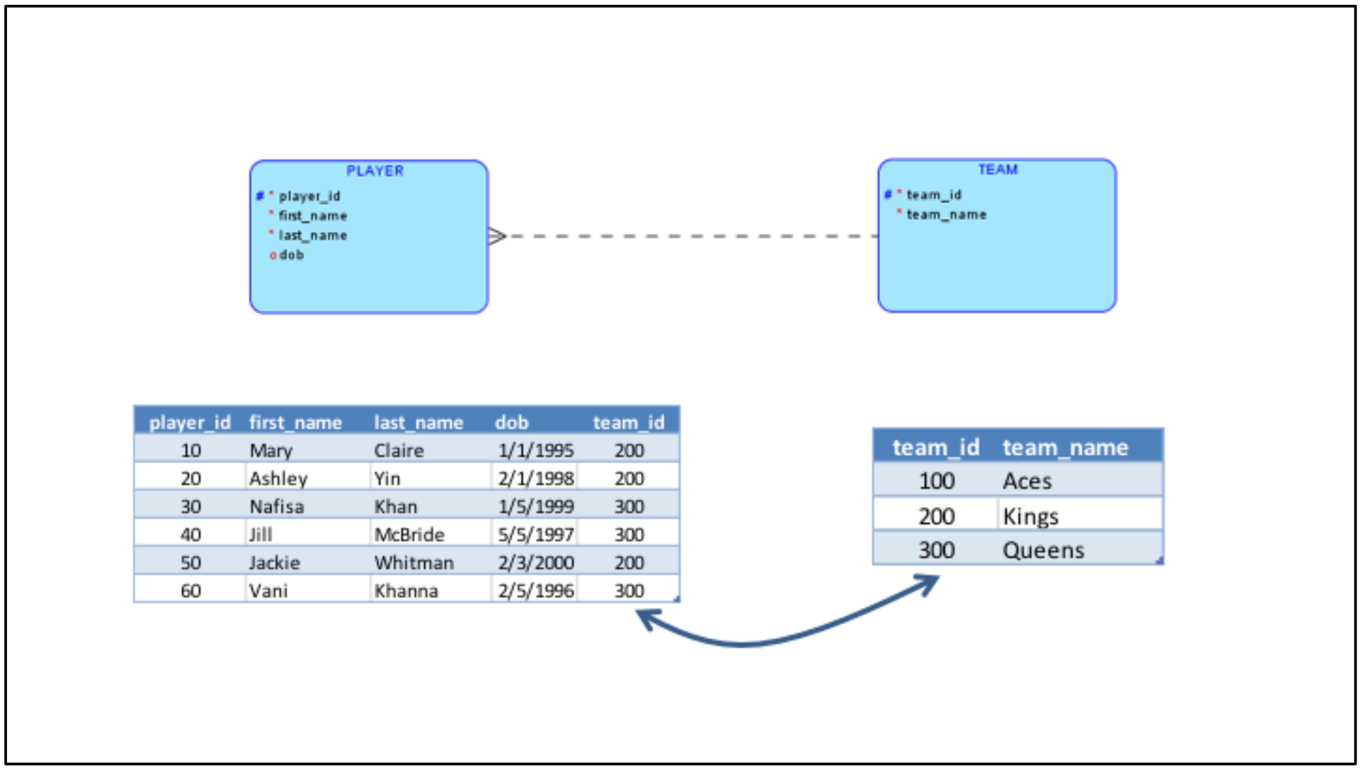

PLAYER
TEAM



ERD	Relational database
Entity type	Table
Relationship	**???**

We have already seen that an entity type on the ERD corresponds to a table on the relational database schema.

What does a relationship on the ERD connote in a relational database schema?



player_id	first_name	last_name	dob	team_id
10	Mary	Claire	1/1/1995	200
20	Ashley	Yin	2/1/1998	200
30	Nafisa	Khan	1/5/1999	300
40	Jill	McBride	5/5/1997	300
50	Jackie	Whitman	2/3/2000	200
60	Vani	Khanna	2/5/1996	300

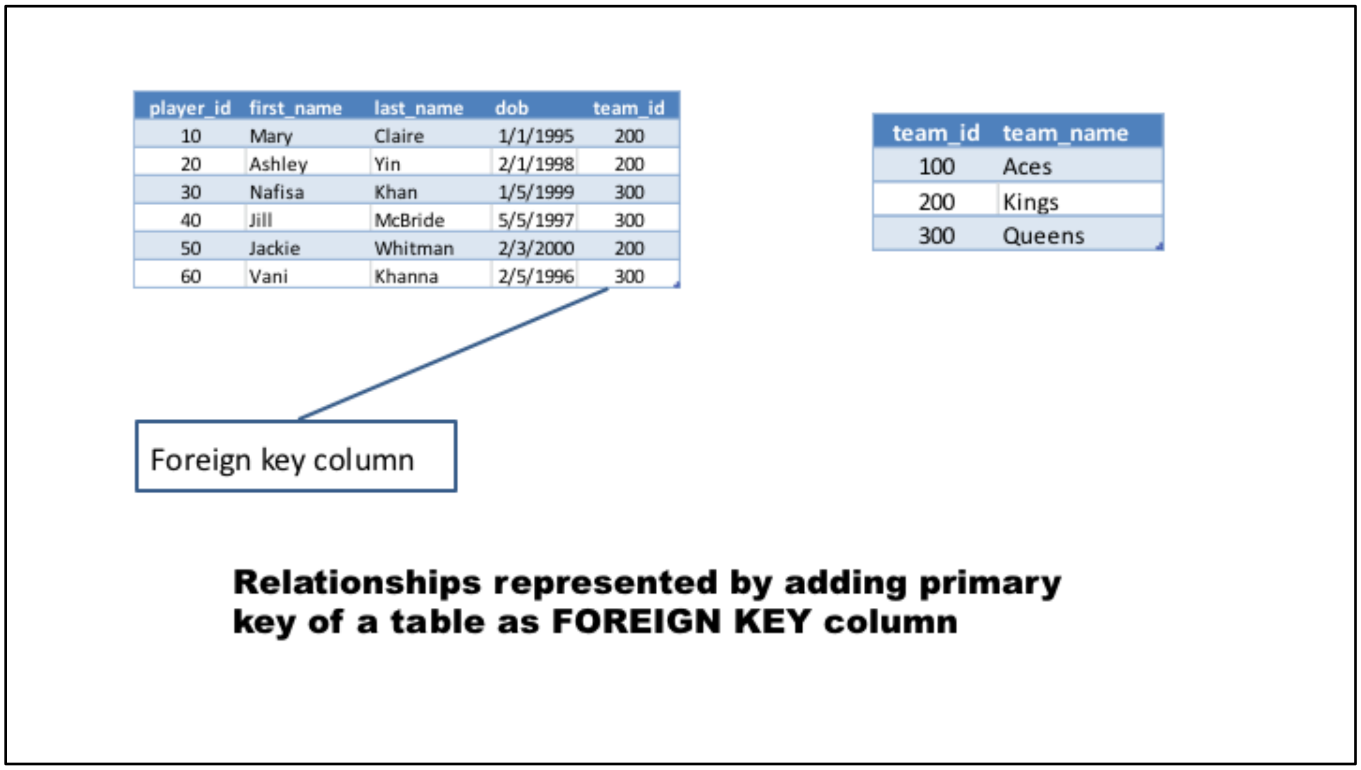
team_id	team_name
100	Aces
200	Kings
300	Queens

From the databases that we have already seen, we know that we can show connections between various tables by adding the primary key of one table as a field or column in another. The slide shows how the connection between players and teams (that is, which team a player belongs to) is represented by adding the team_id to the players table.

In fact this is exactly how we represent relationships in schema.

The added column to represent a relationship is called a **foreign key**. The term makes sense because the field is not a primary key in the table where it is added – like the team_id in the players table, but it is a primary key elsewhere (in the players table); it is the key of some other table.

We will see later that the primary key of a table can sometimes be added to that same table as a *foreign* key.

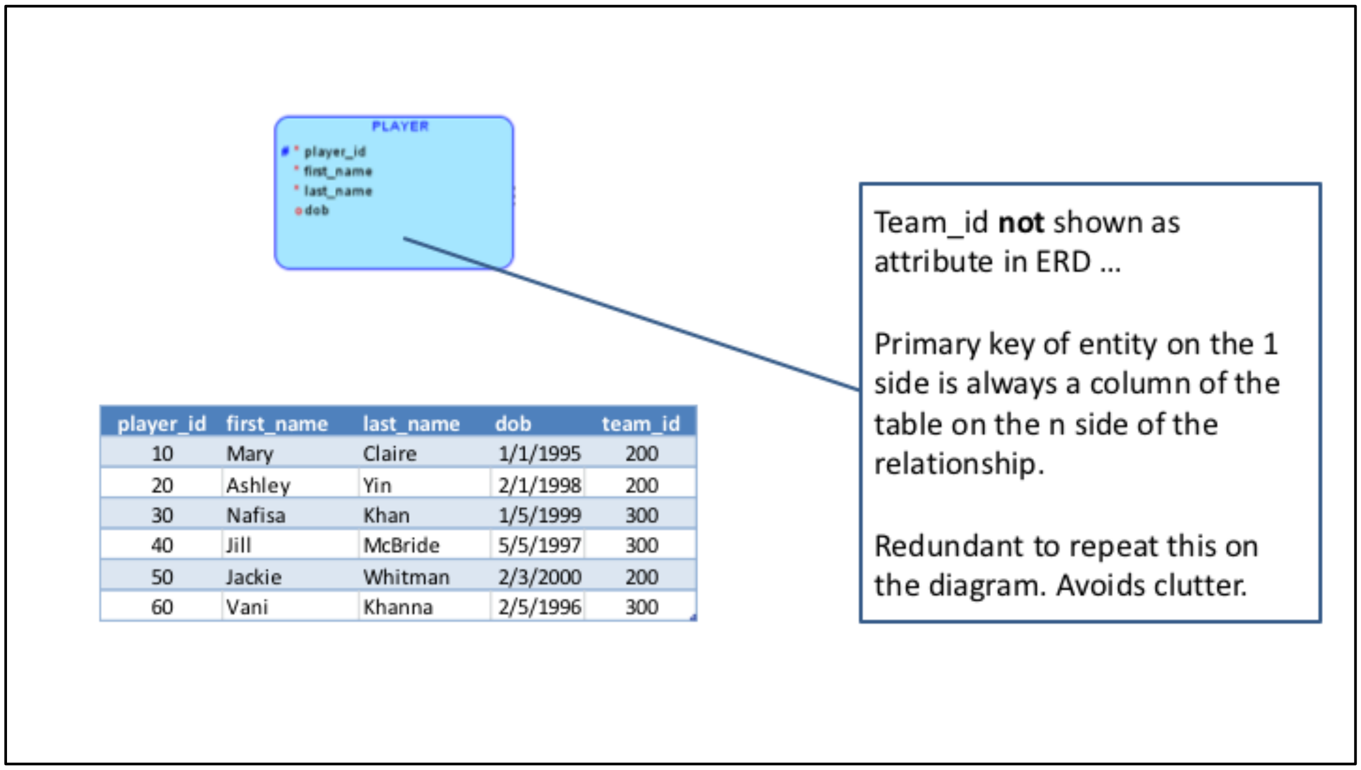


player_id	first_name	last_name	dob	team_id
10	Mary	Claire	1/1/1995	200
20	Ashley	Yin	2/1/1998	200
30	Nafisa	Khan	1/5/1999	300
40	Jill	McBride	5/5/1997	300
50	Jackie	Whitman	2/3/2000	200
60	Vani	Khanna	2/5/1996	300

team_id	team_name
100	Aces
200	Kings
300	Queens

Foreign key column

Relationships represented by adding primary key of a table as FOREIGN KEY column



player_id	first_name	last_name	dob	team_id
10	Mary	Claire	1/1/1995	200
20	Ashley	Yin	2/1/1998	200
30	Nafisa	Khan	1/5/1999	300
40	Jill	McBride	5/5/1997	300
50	Jackie	Whitman	2/3/2000	200
60	Vani	Khanna	2/5/1996	300

Team_id **not** shown as attribute in ERD …

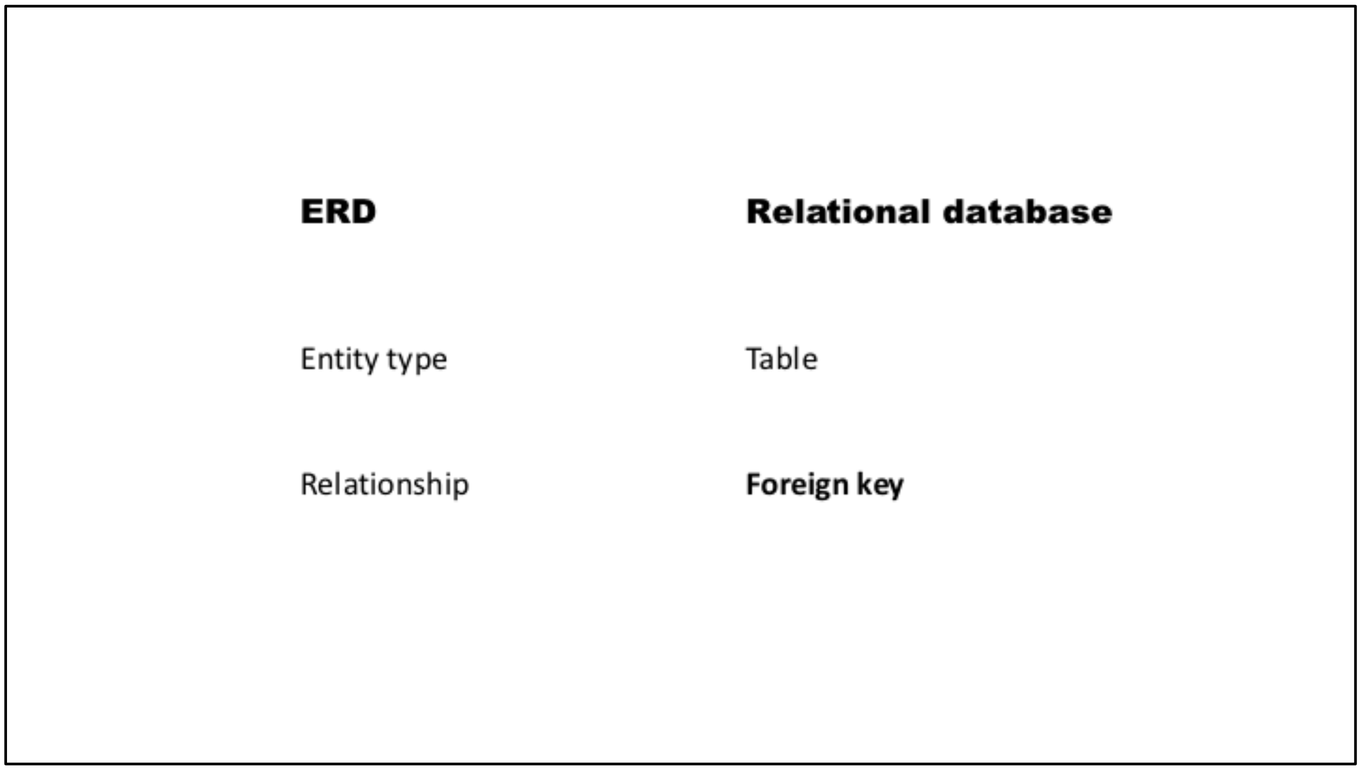
Primary key of entity on the 1 side is always a column of the table on the n side of the relationship.

Redundant to repeat this on the diagram. Avoids clutter.

ERDs can get pretty complex and we want to avoid clutter and redundancy where possible.

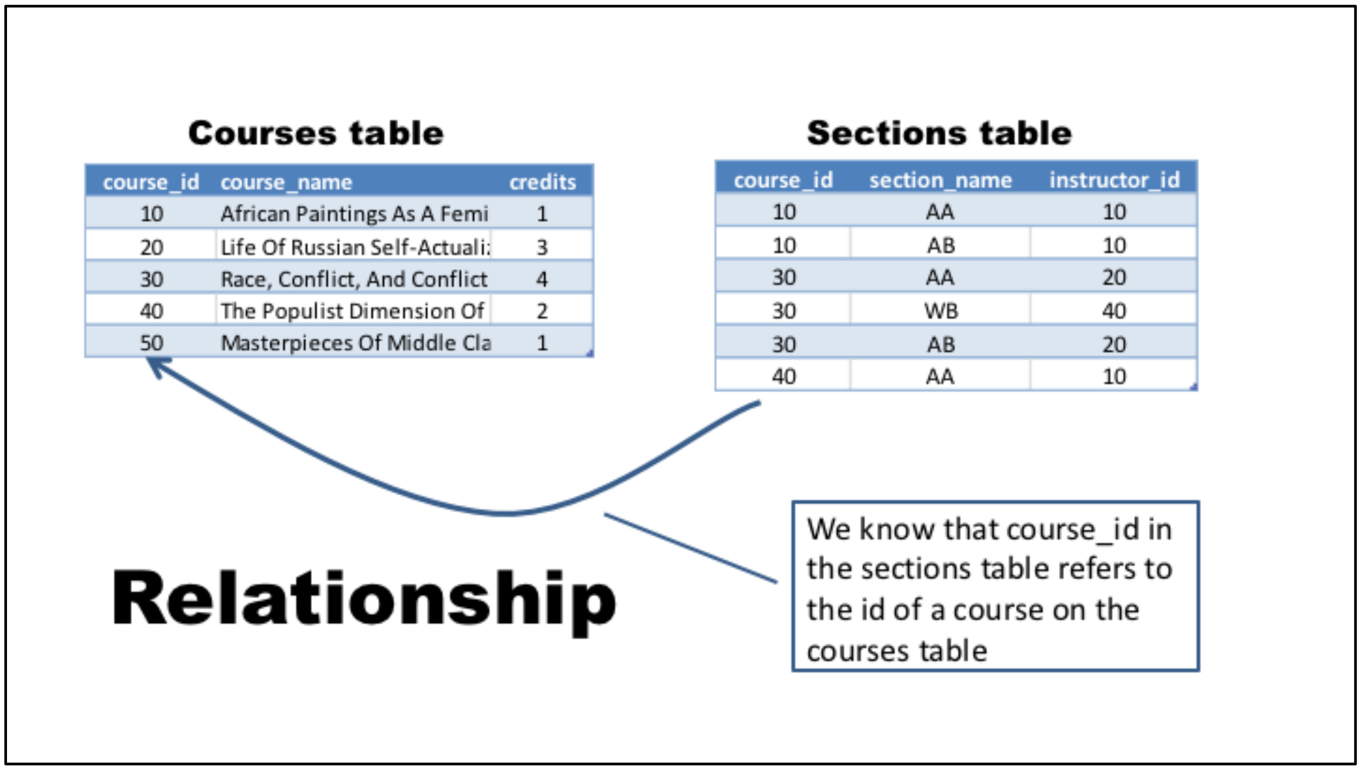
Generalizing from the teams-players example, we can see that to represent a i:n relationship, we add the primary key of the entity type in the "1" side as a foreign key to the entity type on the "n" side of the relationship. This is always true and therefore when we see a 1:n relationship on an ERD, we can always infer this. Therefore there is no need to redundantly show the foreign key attribute on the ERD and add clutter and redundancy.

This is why the above ERD does not show team_id as an attribute in the Player entity type. The ERD on the prior page implies this and so we need not repeat this by showing team_id as an attribute.



ERD	**Relational database**
Entity type	Table
Relationship	**Foreign key**

From the earlier discussion, we see that a relationship on the ERD is represented through a foreign key on the database schema.



course_id	course_name	credits
10	African Paintings As A Femi	1
20	Life Of Russian Self-Actuali:	3
30	Race, Conflict, And Conflict	4
40	The Populist Dimension Of	2
50	Masterpieces Of Middle Cla	1

course_id	section_name	instructor_id
10	AA	10
10	AB	10
30	AA	20
30	WB	40
30	AB	20
40	AA	10

We know that course_id in the sections table refers to the id of a course on the courses table

We have several courses and for some courses we have multiple sections. We now go on to look at the issue of a primary key for the sections table.

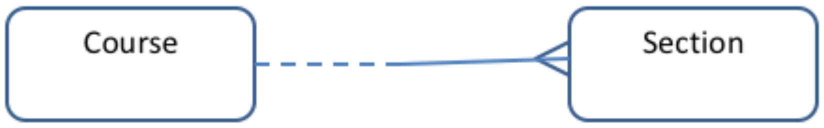
Course
Section

Degree: How many sections can a course have? One or many?

A course can possibly have many sections. Each section is related to one course.

1:n

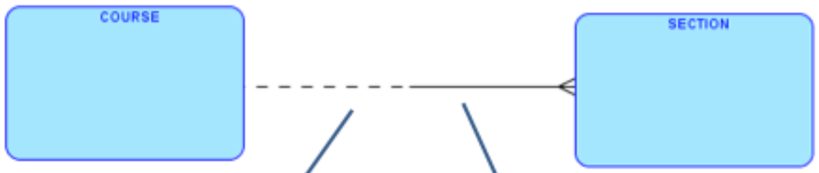
Obligatory Participation?



Does every course need to have at least one section? **No**

Does every section need to be associated with at least one course? **Yes**

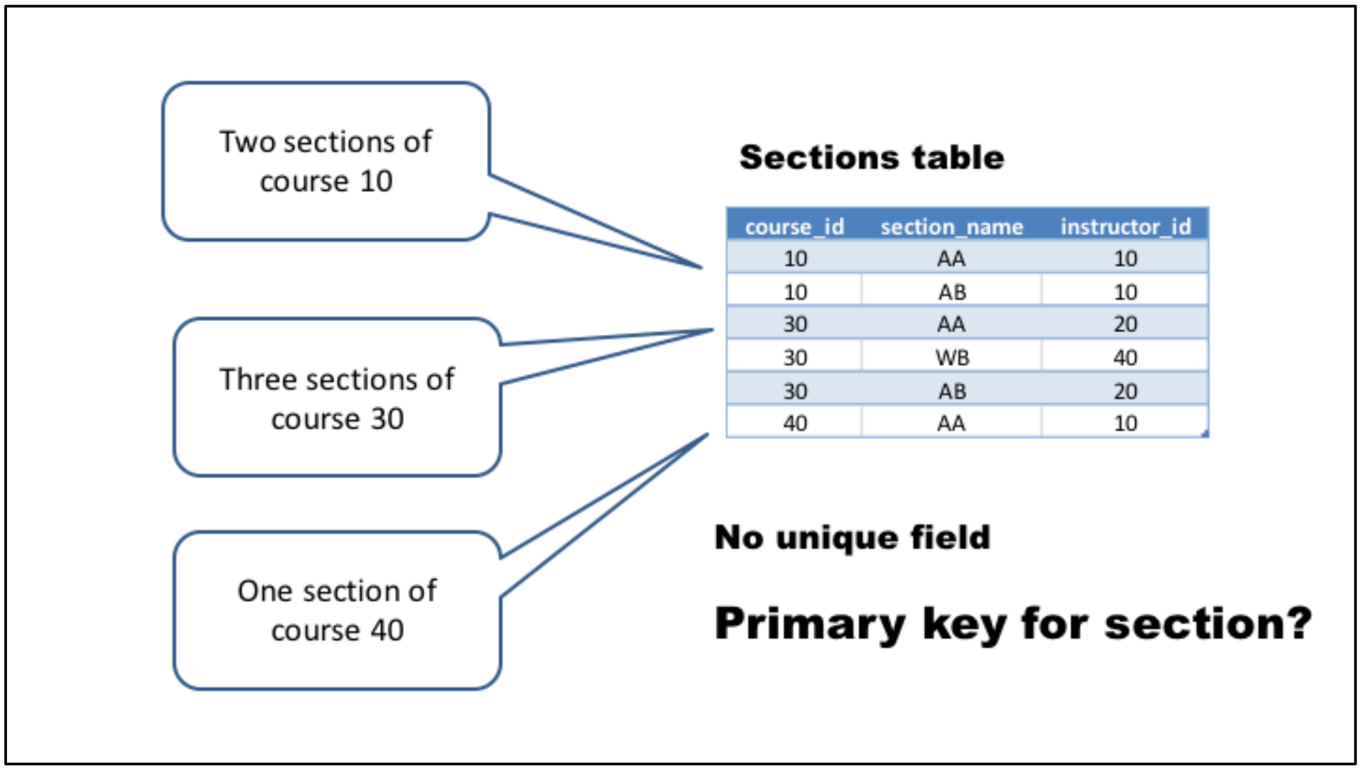
Obligatory Participation?



Every course need not have a section – **non-obligatory**

Every section must be of some course or other – section cannot exist without an associated course -- **obligatory**

Diagram is still incomplete



Sections table

course_id	section_name	instructor_id
10	AA	10
10	AB	10
30	AA	20
30	WB	40
30	AB	20
40	AA	10

No unique field

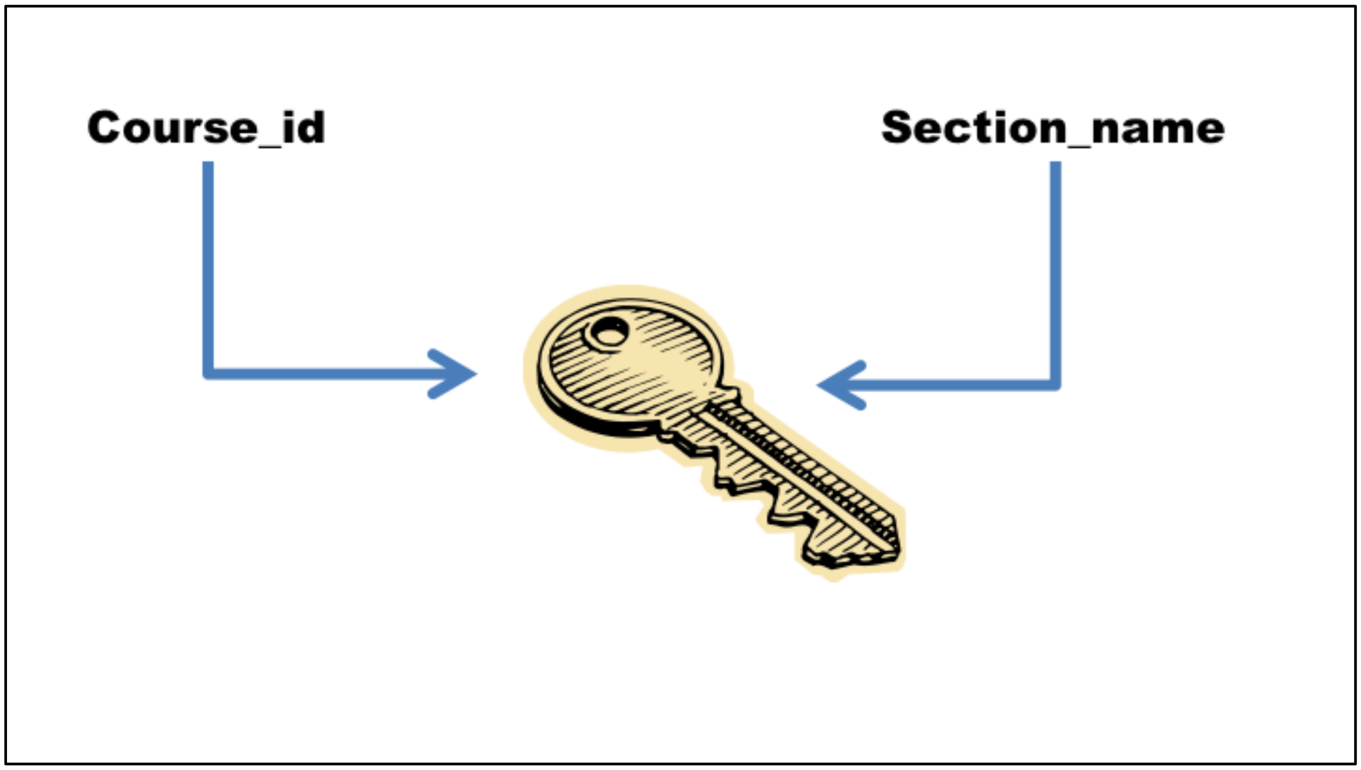
Primary key for section?

We can see that none of the columns in the sections table can be the primary key for the table. We need to think carefully about what uniquely identifies a section. Clearly a course has many sections. To distinguish between the different sections of the same course, we have the section name.

Since the combination of course_id and section_name will surely be unique, we can use this combination as the primary key.

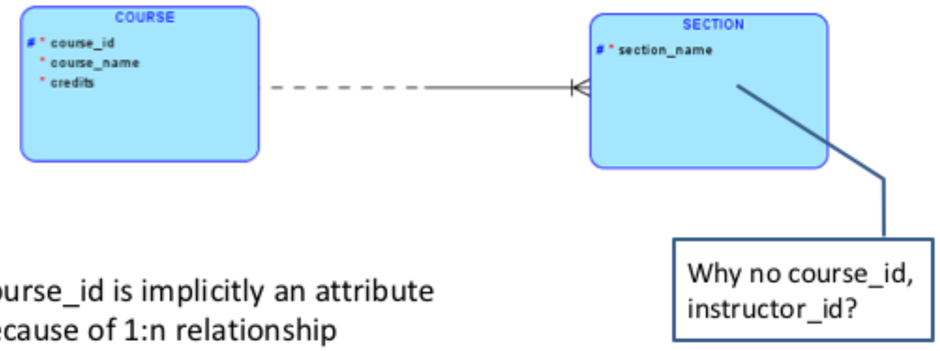
Combination is **unique** -
- can be the primary key

course_id	section_name	instructor_id
10	AA	10
10	AB	10
30	AA	20
30	WB	40
30	AB	20
40	AA	10


Course_id
Section_name

course_id	course_name	credits
10	African Paintings As A Femi	1
20	Life Of Russian Self-Actuali:	3
30	Race, Conflict, And Conflict	4
40	The Populist Dimension Of	2
50	Masterpieces Of Middle Cla	1

course_id	section_name	instructor_id
10	AA	10
10	AB	10
30	AA	20
30	WB	40
30	AB	20
40	AA	10



Course_id is implicitly an attribute because of 1:n relationship

Because of the 1:n relationship between Course and Section, course_id is implicitly an attribute (as a foreign key) in Section. Thus, we do not need to show it as an attribute on the Section entity in the ERD.

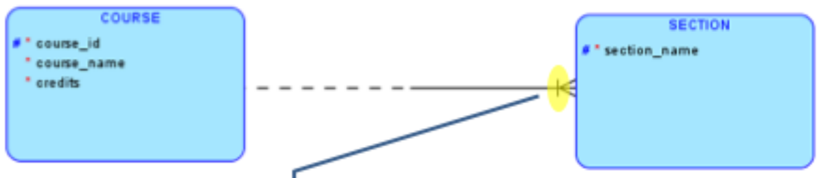
However, this poses a problem because course_id is one of the columns in the primary key of Section. How can we show that the primary key for Section is course_id+section_id without being ale to put course_id as an attribute in Section?

The small vertical bar near the crow-foot is called as the key migration notation and it is used to solve this problem. The key migration notation tells us that the primary key of the entity type on the opposite side of the relationship is part of the primary key of this entity type.

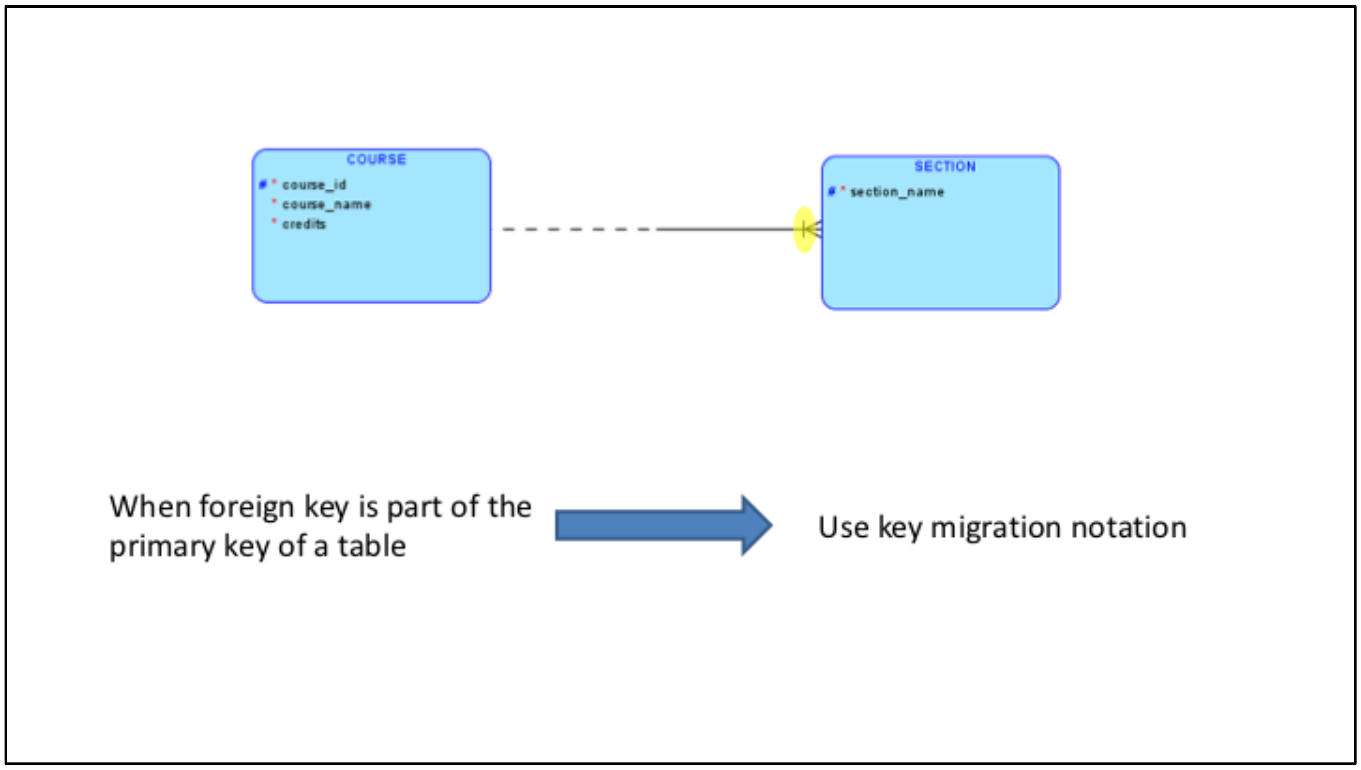
Specifically, because of the key migration notation above, we know that the primary key of Section is not section_id alone, which is all that is shown inside the Section entity type, but that course_id is also a part of the primary key for Section.

course_id	course_name	credits
10	African Paintings As A Femi	1
20	Life Of Russian Self-Actuali:	3
30	Race, Conflict, And Conflict	4
40	The Populist Dimension Of	2
50	Masterpieces Of Middle Cla	1

course_id	section_name	instructor_id
10	AA	10
10	AB	10
30	AA	20
30	WB	40
30	AB	20
40	AA	10



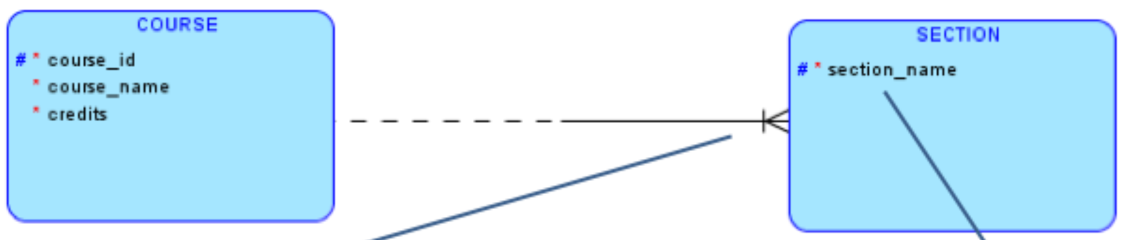
Shows that primary key of entity on the other side is part of the primary key of this entity – **key migration**



When foreign key is part of the primary key of a table → Use key migration notation

Whenever a situation arises that the foreign key is also part of the primary key we should use the key migration notation.

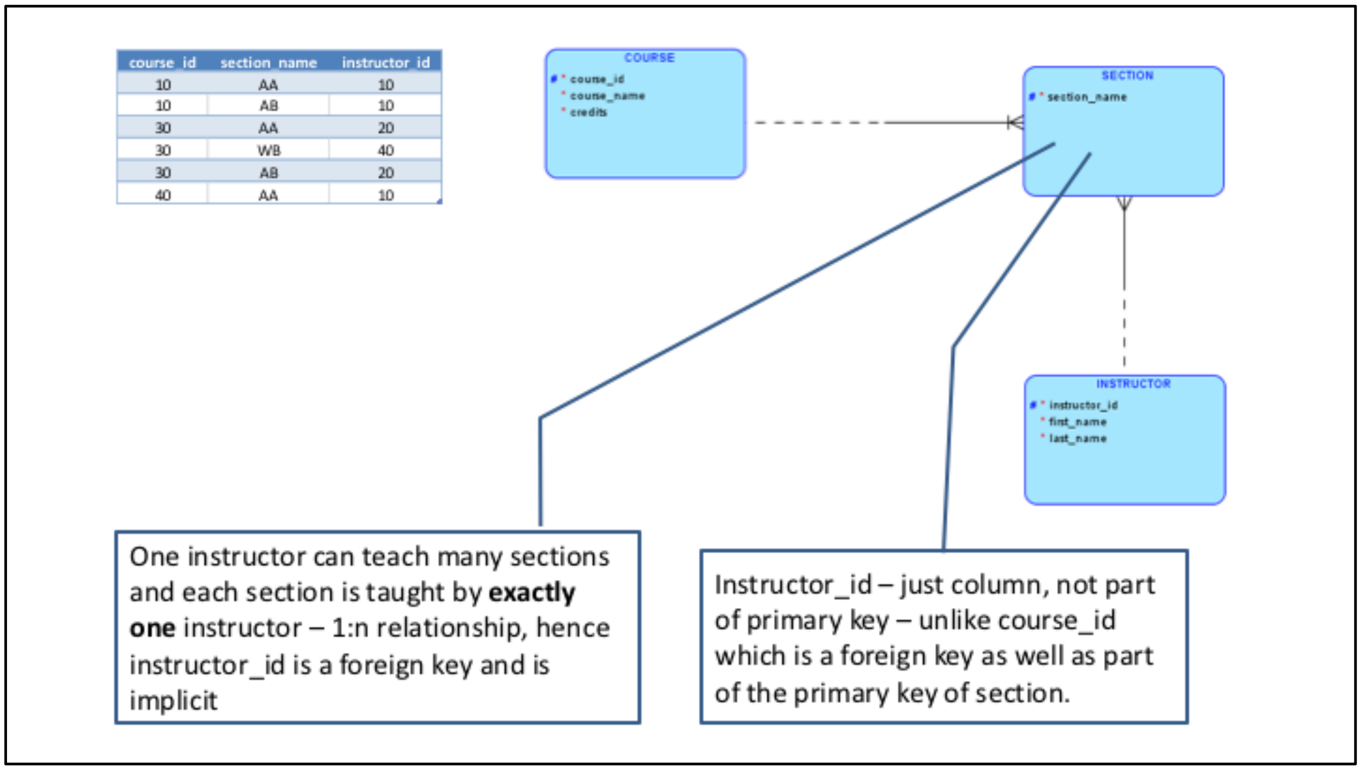
Figure out the primary key for SECTION just by looking at the figure?



Key migration notation tells us that course_id is also part of the primary key

Course_id+section_name

tells us that section_name is part of the primary key



course_id	section_name	instructor_id
10	AA	10
10	AB	10
30	AA	20
30	WB	40
30	AB	20
40	AA	10

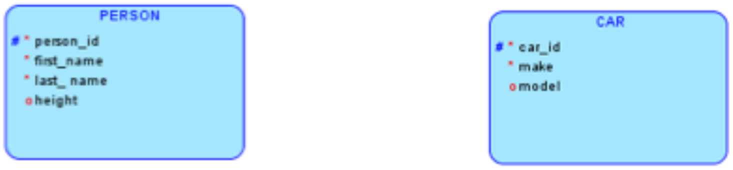
One instructor can teach many sections and each section is taught by **exactly one** instructor – 1:n relationship, hence instructor_id is a foreign key and is implicit

Instructor_id – just column, not part of primary key – unlike course_id which is a foreign key as well as part of the primary key of section.

Instructor_id in the sections table is also a foreign key and hence it is not explicitly shown in the Section entity type. However, the mere presence of a 1:n relationship automatically tells us that instructor_id is a foreign key attribute and does not need to be redundantly specified on the entity type.

Practice 1: A person might own several cars or no cars (car_id is the primary key). Each car is owned by exactly one person (person_id is primary key). Make up two or three attributes other than the indicated primary key.

Step 1: Identify the entity types and their primary keys



Practice 2 (continued): A person might own several cars or no cars (car_id is the primary key). Each car is owned by exactly one person (person_id is primary key). Make up two or three attributes other than the indicated primary key.

Step 2: Identify the degree of the relationship

1:n

Each person can own several cars (upper limit)

Each car can belong to exactly one person (upper limit)

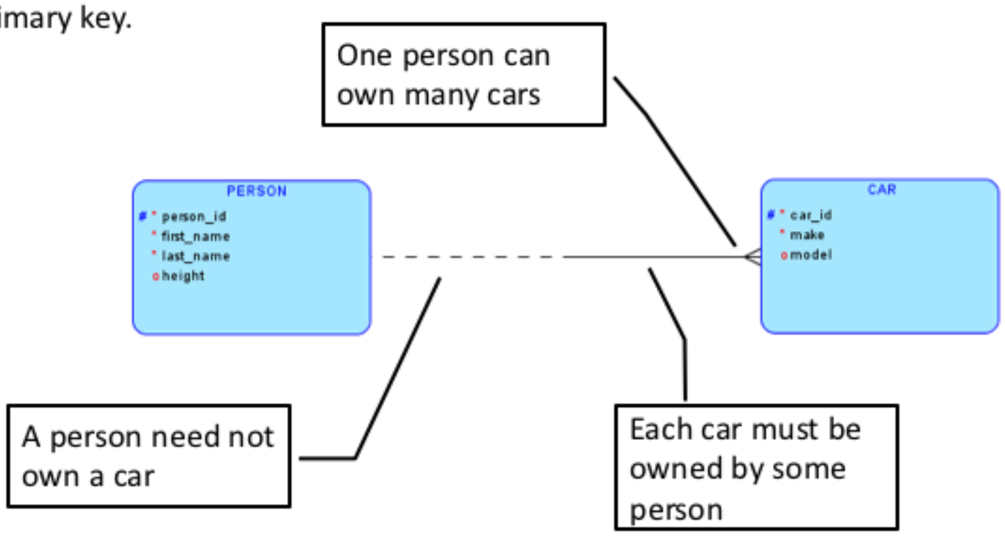
Practice 2 (continued): A person might own several cars or no cars (car_id is the primary key). Each car is owned by exactly one person (person_id is primary key). Make up two or three attributes other than the indicated primary key.

Step 3: Identify participation

Each person can own zero or more cars – Non obligatory

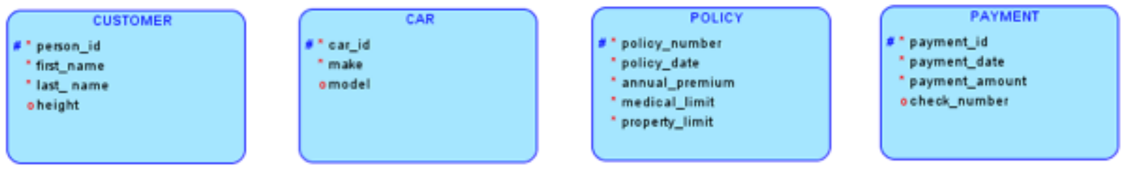
Each car must belong to exactly one person – Obligatory

Practice 2 (continued): A person might own several cars or no cars (car_id is the primary key). Each car is owned by exactly one person (person_id is primary key). Make up two or three attributes other than the indicated primary key.


One person can own many cars
PERSON
# * person_id
* first_name
* last_name
o height
CAR
# * car_id
* make
o model
A person need not own a car
Each car must be owned by some person

Practice 2: Construct an ER diagram for a car insurance company whose customers own one or more cars each. Each insurance policy covers one or more cars, and has zero or more premium payments associated with it. Each payment is for a particular policy and for a period of time and has an associated due date and the date when payment was received.

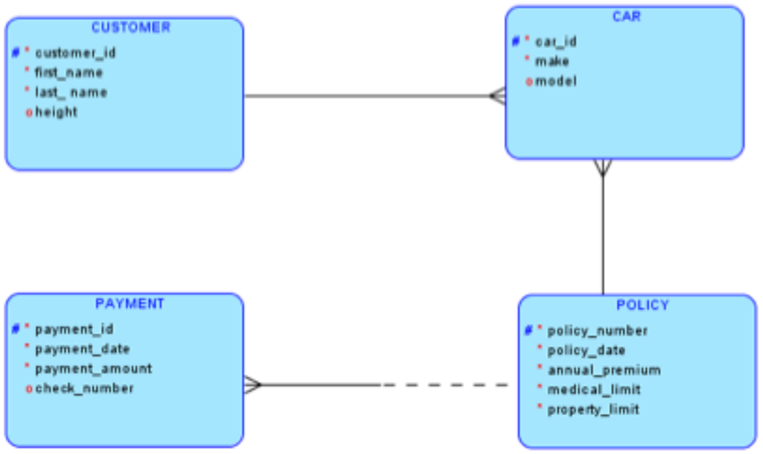
Step 1: Identify the entity types and their primary keys



Practice 2 (continued): Construct an ER diagram for a car insurance company whose customers own one or more cars each. Each insurance policy covers one or more cars, and has zero or more premium payments associated with it. Each payment is for a particular policy and for a period of time and has an associated due date and the date when payment was received.

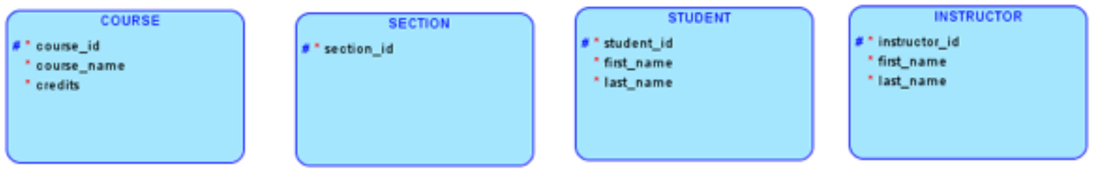
Step 2: Identify relationship degrees		**Step 3: Identify participation rules**
Customer -- Car	**1:n**	Customer, Car: Obligatory
Policy -- Car	**1:n**	Car, Policy: Obligatory
Policy -- Payment	**1:n**	Policy: Non-obligatory; Payment: Obligatory

Customer -- Car	**1:n**	Customer, Car: Obligatory
Car -- Policy	**1:n**	Car, Policy: Obligatory
Policy -- Payment	**1:n**	Policy: Non-obligatory; Payment: Obligatory



Practice 3: A course can have many sections or none. Each section is of exactly one course. A student can be registered for several sections or none. A section could have many students or none and a section must have exactly one instructor. An instructor could be teaching many sections or none. Each student has an instructor as advisor. Each instructor might be the advisor for many students or none.

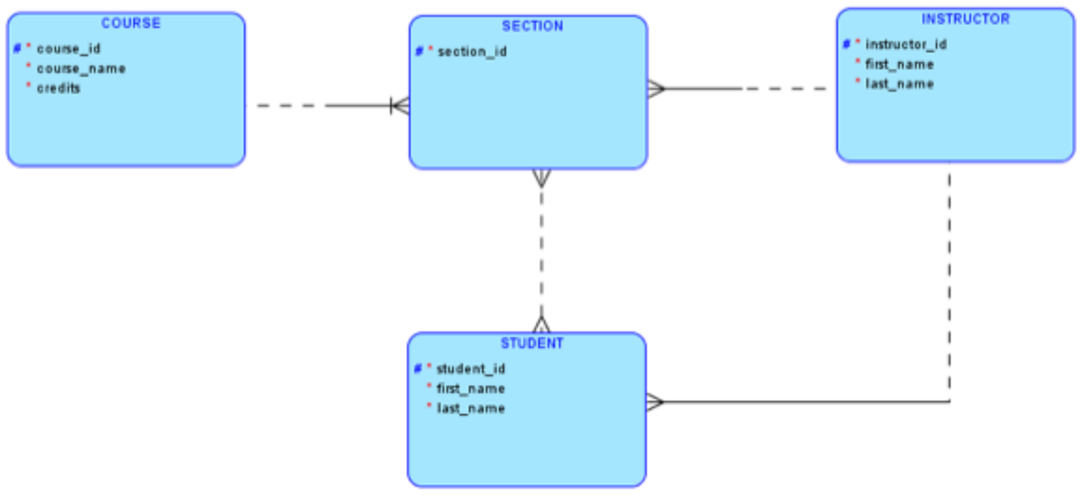
Step 1: Identify the entity types and their primary keys



Practice 3 (continued): A course can have many sections or none. Each section is of exactly one course. A student can be registered for several sections or none. A section could have many students or none and a section must have exactly one instructor. An instructor could be teaching many sections or none.

Step 2: Identify relationship degrees		**Step 3: Identify participation rules**
Course -- Section	**1:n**	Course: Optional; Section: Obligatory
Student -- Section	**m:n**	Student: Optional; Section; Optional
Instructor – Section	**1:n**	Section: Obligatory; Instructor : Optional
Instructor -- Student	**1:n**	Student: Obligatory; Instructor : Optional

Course -- Section	**1:n**	Course: Optional; Section: Obligatory
Student -- Section	**m:n**	Student: Optional; Section; Optional
Section -- Instructor	**1:n**	Section: Obligatory; Instructor : Optional
Instructor -- Student	**1:n**	Student: Obligatory; Instructor : Optional


COURSE
# * course_id
* course_name
* credits
SECTION
# * section_id
INSTRUCTOR
# * instructor_id
* first_name
* last_name
STUDENT
# * student_id
* first_name
* last_name