

# Week 2 - Learn

## Introduction

---

This module will look at the general underpinnings of relational database design. These concepts will apply to essentially any relational database you work with and are quite "platform agnostic". Relational database design is primarily concerned with how to accurately describe the real world data that you are modeling.

In particular we will be looking at 1) the entity relationship model and 2) schemas. We will address *why* we care about having a good model of our data as well as *how* to model it.

Think about a simple thing you might want to model in a database. Maybe the food you have in your kitchen, or your home book collection. Take a quick shot at thinking about how you might organize this data and write it down on a piece of paper.

## Key Questions

---

- What are the benefits of having a well-constructed database model?
- What is the ER model?
- How can one communicate an ER model using crow's foot notation?
- What are the different ways entities can be related in an ER model?
- What sorts of things can an ER model NOT capture?
- What are common uses of ER diagrams?

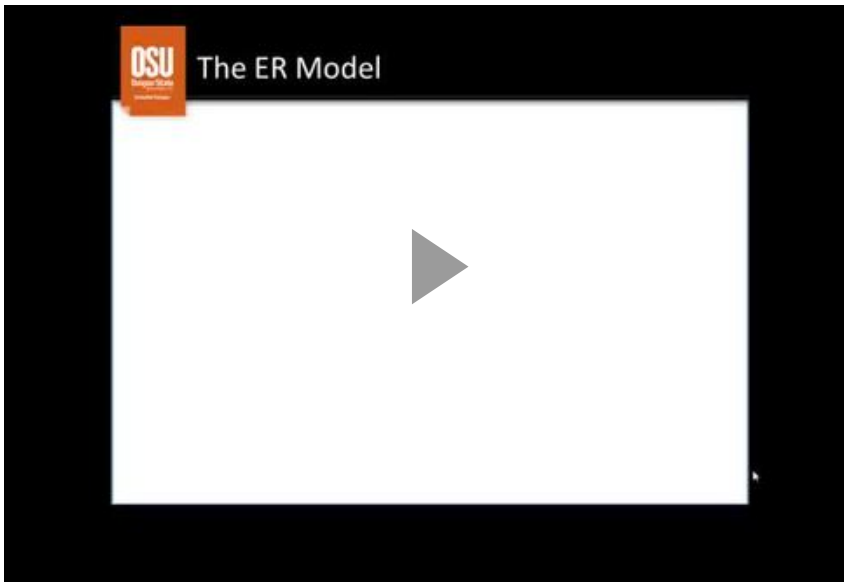
## The ER Model

---

Let us now introduce the primary data model we will be using in this class: the *Entity Relationship model* or the *ER model*. Unsurprisingly, this consists of both entities and relationships, which we will discuss in a bit. It is important to note that the ER model is not the only kind of data model. Another common data model you have used is the object data model where data are described as objects. Another common set of data models is the NoSQL data model, which forgoes most or all relationships in pursuit of better performance.

Following is a video lecture from the old version of this course. I would recommend reading the notes below in addition to watching the lecture.

**The ER Model** ([https://media.oregonstate.edu/media/t/0\\_vhvnhgysy](https://media.oregonstate.edu/media/t/0_vhvnhgysy)) (16:11)



[View the Slides PDF \(https://oregonstate.instructure.com/courses/1746998/files/74604151/download?wrap=1\)](https://oregonstate.instructure.com/courses/1746998/files/74604151/download?wrap=1)  <https://oregonstate.instructure.com/courses/1746998/files/74604151/download?wrap=1>

## Entities

---

Entities are the *things* in your database. Consider a database modeling the ships, roles, and people of the *Battlestar Galactica* universe. Entities could be concrete things, like ships, which would be a collection of attributes like the size of the ship, the number of crew members, the speeds at which it can travel, etc. Entities can also be more abstract, like roles or ranks. A role like "fighter pilot" might have attributes like a name, required hours of annual training, a compensation rate, or a number of work hours required each week.

## Attributes

Entities are not *directly* stored in the database. One can't literally point to a ship inside a database. Instead a collection of attributes following a particular format are stored to represent an entity. So a ship might look like `[23, "Battlestar Pegasus", 5872, 1750, True]`. You would need to see the definition of the ship schema to know what these values meant, but you would be able to look that up and see that it has a unique identifying number, name, length in feet, crew capacity and whether the ship is currently operational.

So more specifically, what is an attribute? An attribute is a piece of data. There are multi-valued and single-valued attributes. For this course we *only* want to deal with single-valued attributes. If an attribute consists of multiple values, we will have to break it down.

## Multi- vs Single-Value Attributes

Ok, so what is the difference between the two? Lets take a name *William Adama*. Is that a single value or multiple values? This all comes down to meaning. It is easy to see that this *can* be broken down into two pieces. A first name, *William*, and a last name *Adama*. Now, how about these? Can the name *William* be broken down further? We could say it is a collection of letters. First letter *W*, second letter *i* and so on, but at this point we have gone too far in almost all cases. No one has any real interest in the fact that the third letter of someones first name is "i". It also isn't really clear if it still has much meaning left.

So first name and last name are distinct values and have value on their own. I may want to print out a list ordered by last name, followed by first name. There are plenty of reasons you would want to access one of these and not the other. So I would feel safe defending a first name as a single-valued attribute, but a full first and last name would probably be a multi-valued attribute.

Now lets look at something a little more complicated: phone numbers. The classic thing to argue about in terms of single or multi valuedness. For example: (123) 456-7890. Is this one value or is it more? This depends a lot on your use case. In general I am going to make the argument this is a single-value attribute.

In most use cases you are going to use a full phone number and not have any use for its constituent parts. If, on the other hand, you were a telecom company and had a specific division that handled issues related to Oregon area codes and a different division that handled California area codes, you are going to need to break this down. If you then need to figure out what city an exchange is related to (the 2nd set of 3 digits) that needs to be a separate attribute.

So there are certainly cases where this could be broken down into multiple values. There are also times where it makes sense to keep it a single-valued attribute. The question to ask is, in your application and any reasonably foreseeable changes to it, do you have need to access the constituent parts of the attribute? Would a person using the data agree that it is not made of multiple parts? If people understand it to be a single value and you don't need to access specific parts of it, treat it as a single value. Otherwise break it down until it is a single value.

## Relationships

---

So we talked about the *things* in our database. Now we need to talk about how these things are *related*: the Relation part of the ER model.

Where it was fairly easy to identify what a thing is and to point to its constituent attributes in a list, it is a little harder to describe and find relationships.

## Kinds of Relationships

### One-to-One

This one isn't really all that common and is a bit hard to differentiate from just being an attribute. It means that things are related in exactly one pair. An example might be water meters and tax lots. Some cities

might only allow there to be a single water meter for a tax lot and every tax lot has only one water meter. But maybe you want to keep track of those two things separately in your database.

## One-to-Many

This is a pretty common relationship where one thing is related to many of another type of thing. As an example consider an airline that wanted to keep track of what airport its aircraft are currently at. An airplane can only be at one airport. However, an airport can have many aircraft currently located there. So it is a one-to-many relationship.

## Many-to-Many

In a many to many relationship is the extension of a one-to-many to both sides. So a student can be enrolled in many classes at the same time and a class can have many students enrolled at once. So we say there is a many-to-many relationship between students and classes.

So relationships exist between entities. And they can even exist within a particular kind of entity. For example, you could just have an entity type *Employee* with a one to many (or maybe even a many to many) relationship of management. Where an employee is managed by a different employee. These can get a little tricky to conceptualize but are not that uncommon.

## Activity

Consider street addresses. Can you think of situations in which it would be better to break addresses into multiple components? What components would you break them into? Can you think of situations in which you might want to keep an address as a single attribute?

This should all familiarize you with the basic terminology we will be using to describe the various conceptual components of the data we are modeling. In particular we introduces the terms *entity*, *relationship* and *attribute*. In addition we talked about the various kinds of relationships we are likely to see when modeling data.

## Keys

---

We are going to look at keys. Not like the keys to your house, more like the keys to... Well... Actually, they are a bit like the keys to your house. Your house key, ideally, should only work in your house. Even if there is another house in a different city that has the same address of 1428 Elm Street, your key only works in your house.

In databases a key uniquely identifies a row in a table. Even if there are one or more other rows that have all the same properties, you can still tell which is which because they have a unique key.

## Superkeys

Lets start with the biggest key of them all: a superkey. A superkey is any set of attributes that uniquely identify a row in a database. The easiest superkey is the entire row. In relational algebra, (the sort of math that describes databases, which we will talk about later), one important rule is that there can be no rows which are total duplicates. So if we used every attribute in a row as the superkey, that superkey is guaranteed to be unique.

This would not be very useful. It would be like searching for a book at a library and having to first know the book, the shelf it is on and if it is checked out or not. If you knew all that info already you would not be searching for the book! What we are really interested in are candidate keys, which are a minimal subset of superkeys.

## Candidate Keys and Primary Key

The only requirement for a superkey is that you can use it to uniquely identify a row. Candidate keys also uniquely identify a row, but they use only the required attributes to do so.

Person

SSN

First Name

Last Name

Lets pretend we live in a world where identity theft does not exist and people in the USA can be identified uniquely by their SSN. But they might have the same first and last names as other people. The super keys would be as follows:

- {ssn, First Name, Last Name}
- {ssn, First Name}
- {ssn, Last Name}
- {ssn}

The candidate key in this case is only the SSN. Any other superkey has additional properties.

It is somewhat difficult to contrive a situation in which there are multiple candidate keys. But suppose for a moment that some people had the same SSN but that no two people had the same SSN *and* first name. In addition no two people had the same SSN *and* last name. In that case we would have two candidate keys to pick from {ssn, First Name} and {ssn, Last Name}.

Whichever candidate key gets chosen is the *primary key*.

Keys are used to specifically identify any row in a database. It may seem odd that we are distinguishing between candidate keys and primary keys. But as we get deeper in the class we will start to discover that having too many keys for an entity can be problematic and that we will go to a fair amount of work to try to make sure there is only one candidate key. But for now, just be aware of what a key is used for and what the difference is between a super, candidate and primary key.

## Future Key Discussion

In the latter portion of this class we will talk about what keys mean for our data design. Often times you end up making significant changes to the way data is laid out in order to 'normalize' it so that we help prevent data inconsistencies. Picking good keys is critical in this process so we will revisit them later when we talk about database normalization.

## ER Notation

---

We have covered a bit about what relational models are and why we are using the ER model. Now let's go into some detail on the actual process of creating and reading ER diagrams. In particular we will be looking at Barker's notation. This is probably the most popular notation for ER diagrams currently.




Another diagram style you may see and want to research yourself is Chen's notation which is able to capture a bit more detail but is much less efficient when it comes to space taken up by the diagram on a page.

## Entities

Recall that Entities are just *things*, which will need their own notation. Entities are represented within a rectangle. The entity name is at the top as a header. Each line below has a property name. Property names with a star next to them indicate that the name is the unique identifier or *primary key* of the entity.

In general this is going to be a single property but on occasion one can have a composite identifier that is made up of multiple properties. This means that it is the combination of those properties that make up the unique identifier. It does not mean that either one can be used alone as the identifier.

There are a variety of things we cannot capture with this notation. One is the domain of properties. Another is whether an attribute is required or optional.

There are a few variations of Barker's notation. Ones not used in this class sometimes use a  symbol to indicate a unique identifier, the  to indicate a required property and the  to represent optional properties. We will not be using this extended notation in the class, but you still might see them occasionally on linked or referenced material.

## Notation for Kinds of Relationship

Entities are one of the things captured by ER diagrams (the 'E' in ER); the other part is Relationships. The notation for relationships is a little trickier but not too much worse. Lines connect related entities; this

is pretty straightforward.

The end of the line represents the participation and cardinality of one entity to another. I find the easiest way to remember what means what with Barker's notation is to think of 3 symbols. The **|** representing a 1, the **0** representing 0 and the **>** representing more than 1.

When following the line from entity A to entity B you will run into two symbols attached to entity B. The first one you run into represents the *minimum* number of B that A must be related to. This will either be **0**, meaning it can be related to 0 Bs or it will be **|** representing that if it exists it must be related to at least one B.

The next symbol, the one closest to B, represents the maximum number of things an A can be related to. It will either be **|** representing a maximum of 1 or **>** representing no maximum. The following table summarizes the combinations.

### **One to Many** [\\_ \(https://media.oregonstate.edu/media/t/0\\_uvldzhro\)](https://media.oregonstate.edu/media/t/0_uvldzhro) (03:48)

This relationship has a **|** on one side and a **>** on the other. This signifies that in one direction a thing can be related to many other things. But in the other direction the relationship is limited to only a single instance of an entity. Teachers and classes are an example of this. A class is taught by one teacher, but a teacher can teach several classes.




### **Many to Many** [\\_ \(https://media.oregonstate.edu/media/t/0\\_rvky4lfy\)](https://media.oregonstate.edu/media/t/0_rvky4lfy) (03:34)

This relationship has a **>** on each end of the relationship line signifying that multiple things can be related in each direction. Students in classes are a classic example. A student can be in several classes and a class has several enrolled students.



### One to One [\\_ \(https://media.oregonstate.edu/media/t/0\\_95novhjc\)](https://media.oregonstate.edu/media/t/0_95novhjc) (03:39)

This is a less common relationship where there is a  on each side indicating that one thing is related to one other thing. An example might be student ID photos and students. A student always gets one student ID photo and a student ID photo is associated to one student. (This would assume you can never get your photo retaken, or if you do it deletes the old one).



### Participation [\\_ \(https://media.oregonstate.edu/media/t/0\\_0tylk5dj\)](https://media.oregonstate.edu/media/t/0_0tylk5dj) (04:32)

So far we have talked only about how many things are in each relationship, or the *cardinality* of the relationship. Participation indicates that something is *required* to be in a relationship. This has a lot more



to do with how you set up your database than real-life situations.

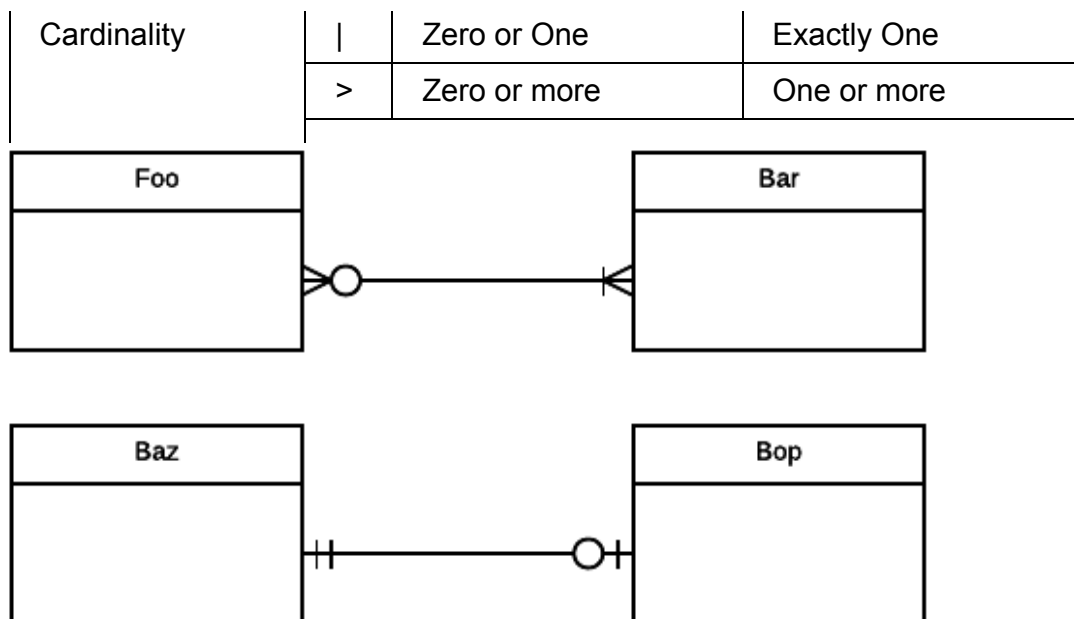


**Special Cases** [\\_ \(https://media.oregonstate.edu/media/t/0\\_t7b43888\)](https://media.oregonstate.edu/media/t/0_t7b43888) (07:04)

There are some edge cases that get a little tricky; this video looks at those special cases.



	Participation	
	O	



This image shows the basic relationships. A **Foo** is related to one or more **Bar**. A **Bar** is related to zero or more **Foo**. A **Baz** is related to zero or one **Bop** and a **Bop** is related to exactly one **Baz**. For the moment this is going to be the limit of our notation, but it can handle most cases.

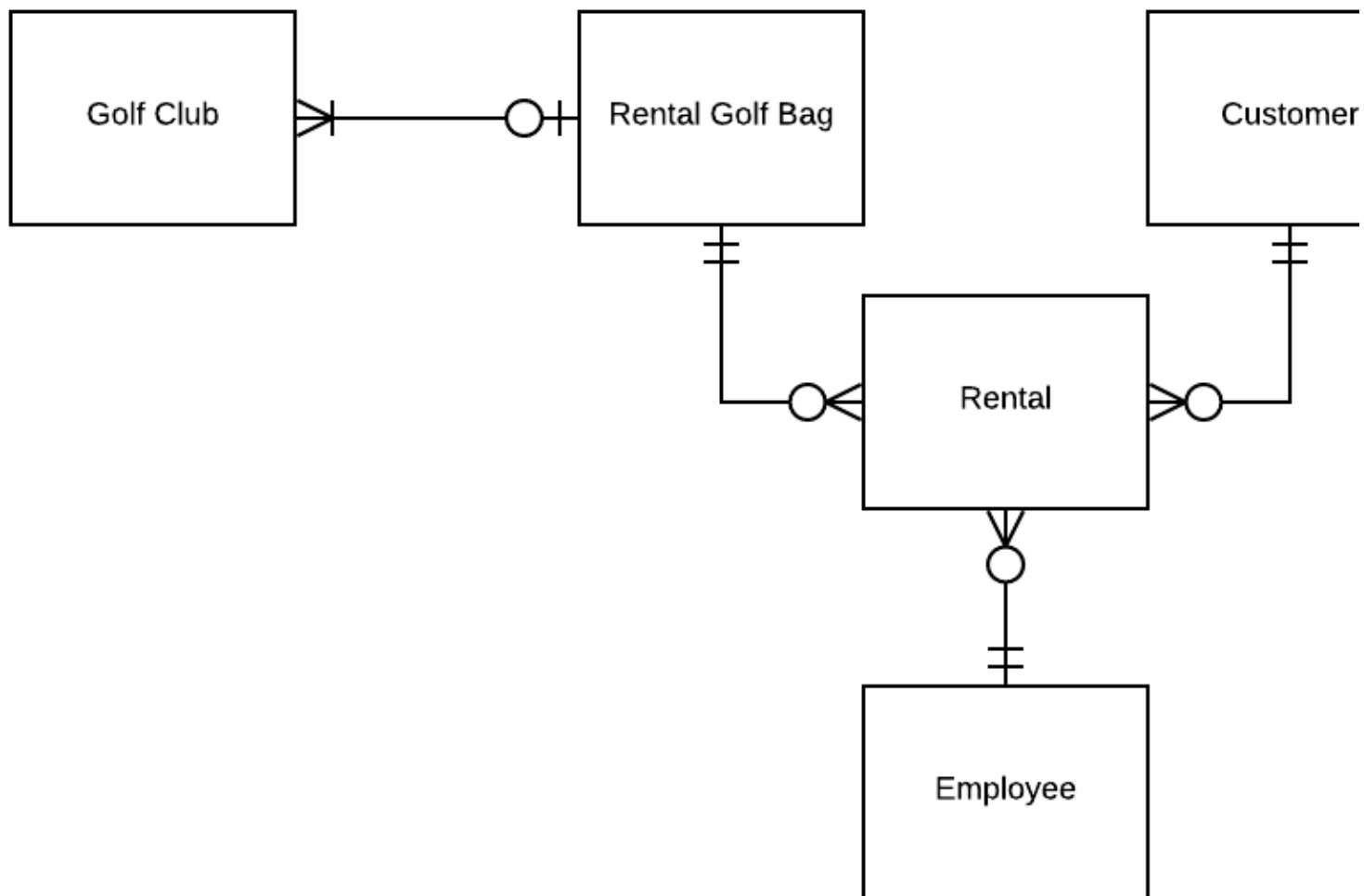
But *only* most cases. Similarly to entities, there are some things we cannot capture using this notation when working with relationships. One thing that really no diagramming tool can capture is business rules. For example, say a client can only be related to an executive account if their balance is over \$9,000. This sort of rule is not really possible to capture using any sort of diagramming notation.

Another thing that is not possible to show using this notation is a ternary or n-ary relationship. Basically a ternary relationship is a relationship between 3 things. As an example, a product, a customer and a coupon might all be related when ordering a product. It isn't possible to show that strictly as a 3-way relationship. Instead you would need to create an additional entity, an order and have an *order* be related to a customer, an order related to a product and an order related to a coupon. In general this works out fine, but it is a limitation of the notation system.

Finally, just as with Entities, there are other, more advanced notations. In particular the  $\diamond$  character can be used in the participation indicator to show that an entity is a subclass of a different entity. This is a challenging thing to implement using the relational database we are using in this class so it is unlikely you will run into it again, but if you see it in the wild, that is what it represents.

## Activity

Describe in words what the following ER diagrams represent. Pay particular attention to how many things are related to each other. Attempt to come up with a list of requirements that you could give to someone else to have them create this same diagram. The attributes are omitted as this is really about figuring out the relationships.



## Week 2 Assignment/Quiz

---

This week you will be given a description of a set of entities and the relationships between them. It will be similar to what a client would give you when explaining the data that might be involved in their potential database. You will need to take that data and correctly convert it to an ER diagram capturing as many of the constraints listed in the description as possible.

## Review

---

Having gone through the material this week, go back to that task from the intro where you took a first attempt at organizing some personal data. What would you change now that you have learned more about database design and the ER model?