

# Week 3 - Learn

## Key Questions

- What is the difference between a schema and an ER diagram?
- Why would we want to create one before the other? Which one should we make first?
- Who is the audience for a schema? Is it different than an ER diagram?
- What is the process for converting from an ER diagram to a schema?

## Week 3 Textbook Readings

- Continue readings from Week 2
- Chapter 4 pages 55-88
- Chapter 5 pages 89-105

## Review: ER Diagrams

You would want to review the content from [Learn page for Week 2.](https://canvas.oregonstate.edu/courses/1825733/modules/items/20221728)  
(<https://canvas.oregonstate.edu/courses/1825733/modules/items/20221728>)

## Schemas

You learned about ER diagrams in the last module. These help us visualize how our data is related. We can see all the entities in our model and how they relate to each other. These are good for talking with clients to make sure everyone agrees about what the data is.



Schemas will give us our first look at the tabular structure of a database. Instead of abstractly looking at entities or relationships, we will actually describe the table names and attribute names we will use to model our data.

However, we still stop short of doing things that would bind us to a particular database engine. For example we don't usually do things like describe what specific data types we will use because the types of strings in PostgreSQL might be different than the options in MySQL.

[Schemas, The Basics](https://media.oregonstate.edu/media/t/0_ocvmr5qw) ([https://media.oregonstate.edu/media/t/0\\_ocvmr5qw](https://media.oregonstate.edu/media/t/0_ocvmr5qw)) (09:01)



**Some confusing definitions**

- A Relation – A table and all its entries (can be a table of Entities or Relationships)
  - Rows are not ord
  - Rows are unique
- A Relation Schema – ... set of constraints on the relation

Again, a Relation is a table and all its entries.



[View the Slides PDF \(https://canvas.oregonstate.edu/courses/1825733/files/82859994/download?wrap=1\)](https://canvas.oregonstate.edu/courses/1825733/files/82859994/download?wrap=1) [https://canvas.oregonstate.edu/courses/1825733/files/82859994/download?wrap=1\)](https://canvas.oregonstate.edu/courses/1825733/files/82859994/download?wrap=1)

## Schema Components

The things we want to capture in a schema are

### Table Names

We want to get all the names of our tables. Pretty straightforward for entities but we also need to name tables, which will be used to describe relationships.

### Column Names

We want to, at this point, also make sure all of our columns have names. Straightforward for entities, but we will need to look at relationships.

### Keys

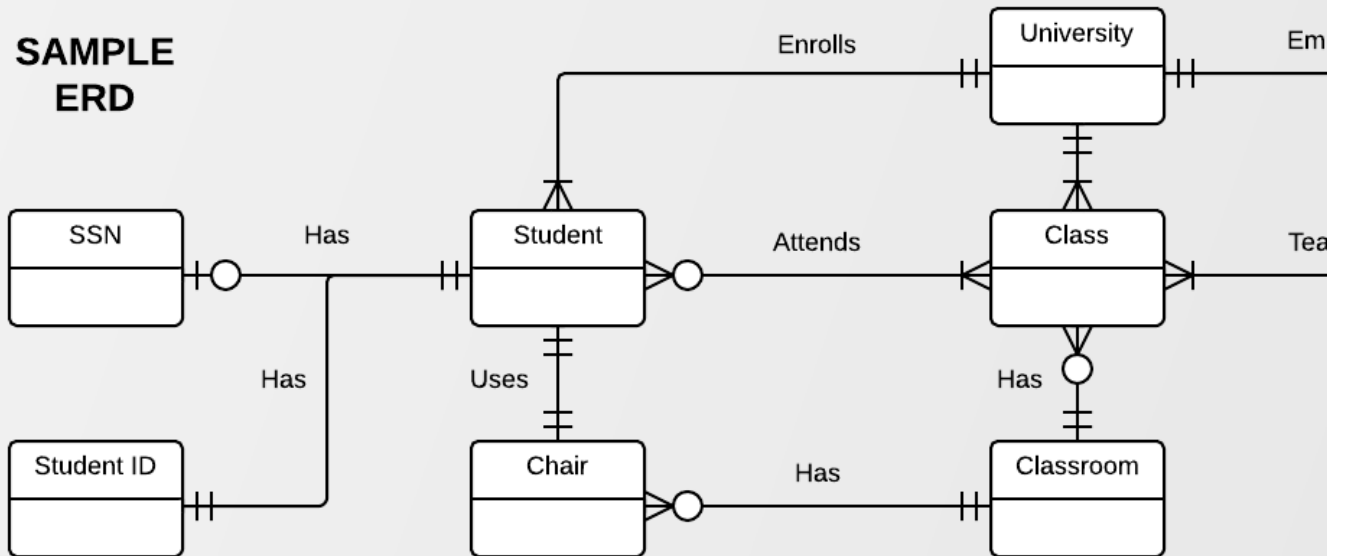
We need to capture primary keys. Depending on our ER notations, we probably already know these. But we also need to know about **foreign keys**. These are the mechanisms by which tables can refer to each other. We will talk about them in some detail in the upcoming explorations.

The following video is from an older version of this course which also contains SQL statements to create the keys. It uses the Chen notation. You can use this cheatsheet to know how to convert from Chen from Crowfoot when watching this video.

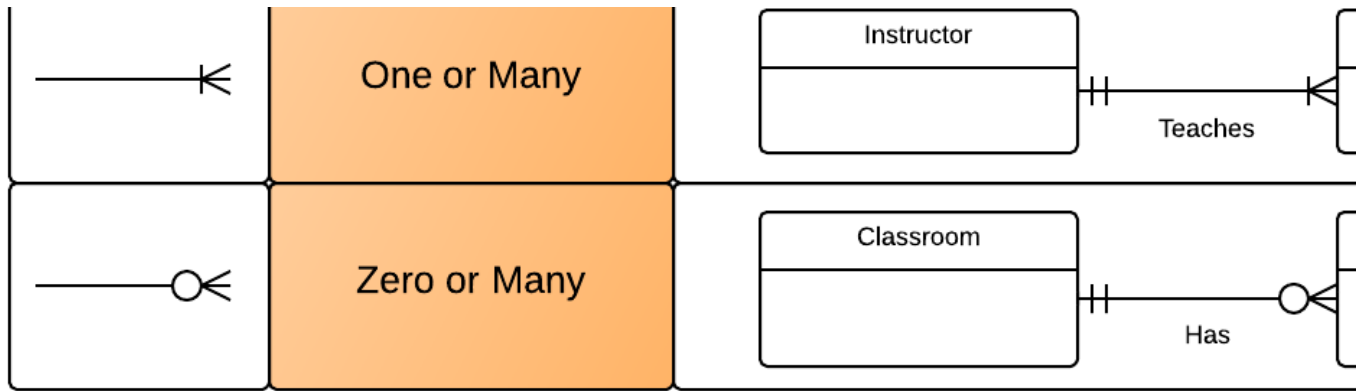
# ERD "Crow's Foot" Relationship Symbols [Quick]

Created by Vivek M. Chawla | @VivekMChawla

## SAMPLE ERD

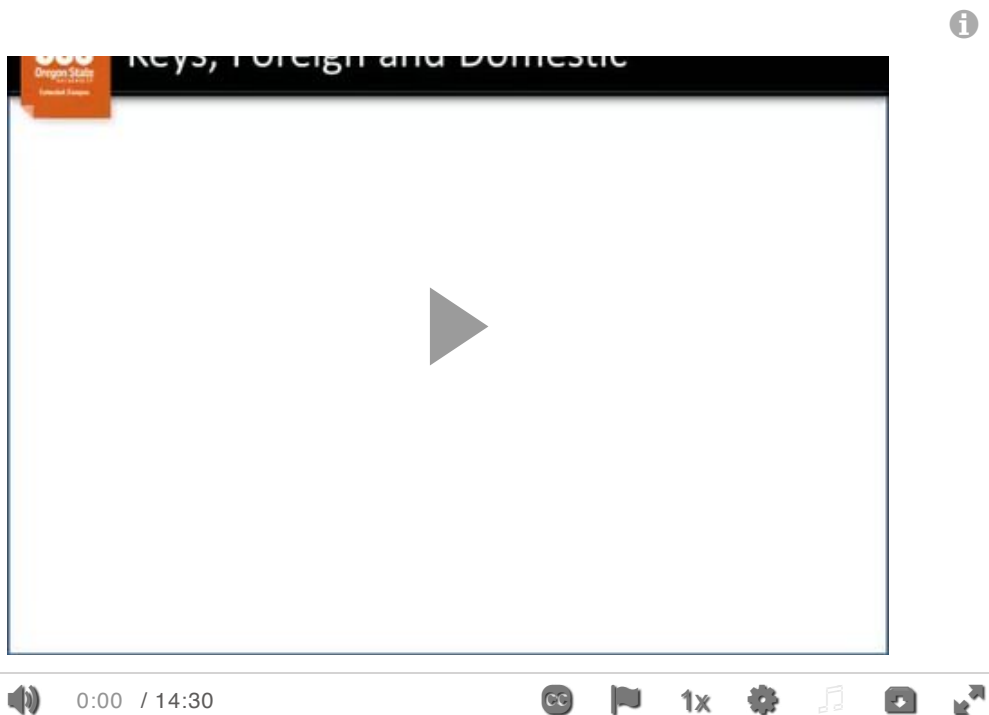


Notation	Meaning	Example
_____	Relationship	
_____+	One	
_____>	Many	
_____++	One and ONLY One	
_____o+	Zero or One	



You can ignore the SQL from it for now as it's not relevant to this week. We will come back to it in later weeks.

**Keys** [\\_\(https://media.oregonstate.edu/media/t/0\\_rdg7epc\)](https://media.oregonstate.edu/media/t/0_rdg7epc) (14:30)



**View the Slides PDF** [\\_\(https://canvas.oregonstate.edu/courses/1825733/files/82860022/download?wrap=1\)](https://canvas.oregonstate.edu/courses/1825733/files/82860022/download?wrap=1) [\\_\(https://canvas.oregonstate.edu/courses/1825733/files/82860022/download?wrap=1\)](https://canvas.oregonstate.edu/courses/1825733/files/82860022/download?wrap=1)

So, in a quick summary, what is new with schemas is really the capturing of relationships. We had ER diagrams which capture entities and properties. And we knew which relationships exist. But we never described how we would represent that relationship. The schema really captures that.

## What Can We Do with Schemas?

With the schema a developer can basically get started with writing queries. They will have access to all the table names and property names. When a programmer is working on software that queries a database they will often have a copy of the schema out in front of them. The ER diagram captures too much information and the actual queries to make the database are too verbose and capture more information than is generally needed. So the schema represents a good compromise that is easy to reference when programming.

Generally you will find that schemas will easily convert from one relational database engine to another. So you can pretty reliably use a schema to create a database in MySQL or PostgreSQL. This is the last stop in the land of the abstract for awhile. Moving forward we are going to dive into more technical topics looking at how to actually convert schemas into tables in a database and how to manipulate that data in a database. The upcoming explorations will get into some more depth in terms of the notation we will use and in particular we will look at how relationships are captured in schemas.

## Relationships in Schemas and Foreign keys

When discussing the ER model we talked about 3 major types of relationships: many-to-many, one-to-many and one-to-one. We also talked about total and partial participation. Now we are going to look at how we can capture those relationships using tables.

The main topic of interest in this section is relationships. How can we go about capturing which things are related to each other? For example, how do we indicate in a database of students and classes that a student is enrolled in a class?

There are a wide variety of approaches to this problem, all of which, in some way or another, solve it. But a lot of them bring some problems of their own.

### How many related things?

One of the biggest issues is trying to deal with multiple related things. Imagine you run an online shop and have customers with mailing addresses. At first you might assume customers have one or maybe two addresses--maybe a residential address and a business address. There are solutions that would accommodate this with no problems.

But what happens if you later end up with a customer who has many addresses? Maybe they frequently travel for business so they have apartments in many cities. You need to make sure that your system has the flexibility to handle more relationships than you might have originally anticipated.

But we can also look at the other side of things. What if you want to make sure that something is related to *only* one thing? Is there a way we could set up our relationship to handle that? For example you may want to say that a car, at any given time, is allowed to have only one license plate number or that a particular edition of a book has only one publisher.

## One thing related many times

How do we handle the situation where a single item is related to many things? For example there might be 100 students enrolled in a single class. Any sort of system for handling relationships should be able to handle this situation as it is very common.

In addition to handling this situation, consider what happens when that thing changes. If the room where the class is held changes, we want to make sure that all 100 students can see that change. It would be quite problematic if different students saw different versions of what is intended to be the same class.

Those are the big constraints, making sure we can either limit relationships to a single item or not have to worry about the limit of related things. We also want to make sure that as something becomes related to a lot of things that we don't break everything or end up with out-of-sync copies if it changes.

## One-to-Many

Let's look at these two tables of hotel guests and hotel rooms. Can you see how the relationship is set up between rooms and guests? How would a database user be able to tell which room a guest is staying in?

hotel_guest				
id	f_name	l_name	check_out	room_number
1	Will	Adama	5/1/2034	101
2	Bat	Man	5/8/2034	103
3	Maverick		5/4/2034	102
4	Goose		5/2/2034	102

hotel_room		
id	cleaned	accessible
101	True	True
102	False	True
103	False	False

The `hotel_guest` table has a column `room_number`. The values in that column correspond to values within the `hotel_room`'s `id` column. So by cross-referencing the tables we can figure out that Maverick and Goose are staying in room 102 and Bat Man is staying in room 103.

This is an example of a one-to-many relationship. We see that more than one person is staying in room 102, but it isn't possible for a guest to be staying in multiple rooms. The row containing Maverick only has one field to enter a room number in so he can't be in more than one room. But nothing prevents a room from being referenced by several different guests.

## Many-to-Many

Now let's say that our hotel client says that this is too simple. In reality there are guests who might rent several rooms, maybe for a family vacation or to throw a party of some sort. Whatever the case, we need to be able to accommodate a guest renting several rooms at once **and** a room being occupied by several guests at once.

There is really only one way to capture this. We need to make a new table, a relationship table. The only data this table will have is a listing of which rooms are checked out by which guests (and inversely which guests are associated with which rooms).

The `hotel_room` table stays the same.

hotel_room		
id	cleaned	accessible
101	True	True
102	False	True
103	False	False

We drop the room number column on the `hotel_guest` table

hotel_guest			
id	f_name	l_name	check_out
1	Will	Adama	5/1/2034
2	Bat	Man	5/8/2034
3	Maverick		5/4/2034
4	Goose		5/2/2034

And now we make a new table, `room_guest` that is a relationship between rooms and guests.

room_guest	
guest_id	room_id

1	101
1	102
2	101
3	103
4	103

So now Will Adama has rooms 101 and 102, Bat Man is sharing room 101 with Mr. Adama and Maverick and Goose are still in room 103. We have people who have checked out multiple rooms and rooms shared by multiple occupants.

This is generally how you will handle these relationships. The place where things can get a little confusing is trying to decide if what you are describing is an entity or a relationship. In this case it is clearly a relationship.


But what if we added check-in and check-out dates as fields in our `room_guest` table? Is that just some additional data about the relationship? Or have we made a new *entity* entirely? From an implementation standpoint this isn't critical to decide. The database will be constructed the same way, but it can help to clarify these things when it comes time to decide what limitations you will put on data and how the user interface will be designed.

## Schema Notation

You can use either of the following schema notations



A)




# Schema Notation

- Your book does this:


students				
<u>ID</u>	Name	Year	GPA	Birthday
- That's ok but I prefer  
Students(  
ID,  
Name,  
Year,  
GPA,  
Birthday)
- It looks more like SQL, you can do either

Notation 1 example  
(without the arrows  
connecting Foreign  
Keys)



Note that though there aren't arrows in the above screen shot of Notation A) you would have to use them to connect foreign key and primary keys if you choose to use this!

B)



## More notation

bsg_people				
<u>Id</u>	Fname	Lname	Homeworld	Age

bsg_planets				
<u>Id</u>	Name	Population	Language	Capitol

Diagram illustrating database notation for two tables: **bsg\_people** and **bsg\_planets**.

- bsg\_people** table attributes: Id, Fname, Lname, Homeworld, Age.
- bsg\_planets** table attributes: Id, Name, Population, Language, Capitol.

The diagram shows a blue arrow pointing from the Id attribute in the **bsg\_people** table to the Id attribute in the **bsg\_planets** table, indicating a foreign key relationship.

- Underlined Attributes – These are the attributes making up the primary key
- The arrow – Signifies one attribute references another
- More on both of these in later lectures

You do *not* need to include data types in the schema.

## Additional Resources

**Harrington Chapters 5** - This chapter goes over the way to represent various relationships in table form rather than in an ER diagram. This is typically the most challenge part of converting to a schema from an ER diagram.

