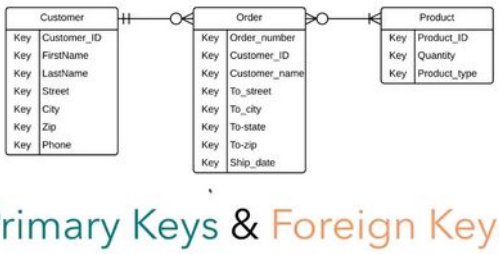
Entity Relationship Diagram (ERD) Tutorial – Part 2 [1]

1. Hi, my name is Taylor and I'm going to walk you through some ***advanced*** **ERD** ***concepts***. Today we're going to be going over primary keys, foreign keys, bridge tables, and then really understanding how ER diagrams help you see exactly what's going on in your database.

2. So let's start where we left off – with our ***very simple diagram*** from the previous tutorial. So we've got entities for customer, order, and product, and the correct cardinality between. You'll notice that next to each attribute is the word "key". This column is here so we can distinguish certain attributes as either *primary keys* or *foreign keys*.



3. Let's start with ***primary keys***. This is probably easiest to understand if we step out of the diagram and into the table it represents. Remember, each entity in your diagram represents a table within your database. So let's look at the *customer entity* as a table with rows and columns. We've got all these different instances of the customer here. And if you are *Amazon* or something, you've got millions of customers all within this table. Wouldn't it be nice if you could use a single trigger to quickly and accurately distinguish who's who in this massive list?

4. That's where a primary key comes in. A primary key is an attribute (or field) that uniquely identifies every record in a certain table. And since a single attribute can accomplish all that, it makes sense that you'll just need ***one primary key per entity***. So for this customer table, the primary key is going to be a value that makes it so that this customer is unlike any other customer in our table.

5. For one of these attributes to qualify, there are ***a few rules***.
   - First, a primary key has to be *unique* so that it identifies only one record in your table.
   - Second, it needs to be *never-changing*; it would be pretty difficult to keep an accurate record of your customers if you're using an attribute that might not stay consistent.
   - And finally, a primary key needs to be never-null; that just means there wouldn't be an occasion where the value could be left blank.

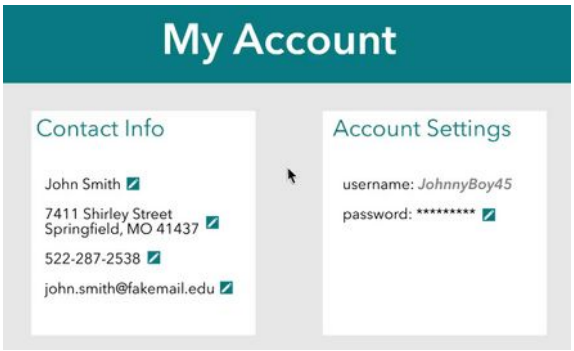
1 https://www.youtube.com/watch?v=-CuY5ADwn24

6. So let's look at ***a specific customer***, like John here, and determine what data might uniquely identify him. Well, you can't rely on names because two totally different customers could very easily share the same first and last name. There's a John Smith here and a John Smith here… but there are two completely different customers. So it's not a unique record in this table. You can't even rely on an address because you could have two separate customers living at the same place… or a customer could move and then their address changes. So an address isn't unique and breaks the rule of never changing. Same thing for a phone number.

7. So that leaves us with ***customer*** **ID**. By design, any sort of ID is typically programmed to increment for each addition to the table. This customer signs up and is assigned customer ID 3-0-0-1-6. Then this customer signs up and is assigned 3-0-0-1-7, and so on. You can see how customer ID passes all our rules. John's customer ID will completely identify him as a particular instance in our database, and that value won't ever be repeated in this table. So we're going to make that our primary key.

	Customer_ID	FirstName	LastName	Street	City
12	30011	Linda	McGrath	7249 N. Bow Ridge St.	Ft Mitchell
13	30012	Iris	Edmunds	7135 North Rocky River Court	Yorktown
14	30013	Chandra	Parsons	847 Tanglewood Dr.	Calhoun
15	30014	Ranee	Peters	696 Fawn Court	Albany
16	30015	Steven	Langdon	64 Pennington Ave.	Jacksonville
17	30016	John	Smith	7411 Shirley Street	Springfield
18	30017	Ben	Chapman	6 James Ave.	Hopkinsville
19	30018	Jeremy	Nash	76 Strawberry Court	Billerica
20	30019	Rhett	Buckland	243 Mayflower St.	Watertown
21	30020	Carmen	Jones	8318 Mammoth Ave.	Lorton
22	30021	Marylynn	Smith	7411 Shirley Street	Springfield
23	30022	Dorothy	Taylor	845 South Bay Meadows Dr.	Trumbull
24	30023	Lena	Clarkson	50 Westport Rd.	Valparaiso

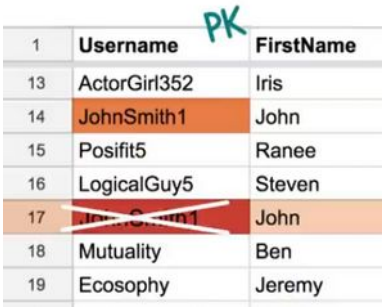
8. Here's something interesting to think about while we're on this topic.



- Have you ever tried to create a ***username*** for an account and then later wanted to change your username to something else because it was hard to remember or stupid or something… but then you can't because the site won't let you? Well, that's probably because your username is being used as a primary key in that site's database, and primary keys can never change. That's how the site is linking you, the customer, to your account. They don't let you change that primary key because their system relies on it for accurate records.

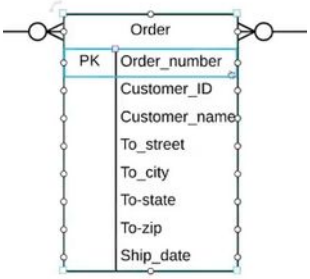
- Or how about when you're setting up a new account and you try to create a username but get an error message saying that it's already taken. Again, this could be happening because the username is being used as the primary key, and a primary key can't be repeated.

In our example though, we're just using a ***randomly assigned*** *customer* ID number as our primary key. We know this will always to be unique and never repeated. So let's jump back to our diagram and note this by putting "PK"next to our customer ID. And I'll go ahead and quickly clear out all this other text to make things clearer.

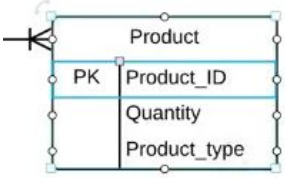


	Username	FirstName
13	ActorGirl352	Iris
14	JohnSmith1	John
15	Posifit5	Ranee
16	LogicalGuy5	Steven
17	[redacted]	John
18	Mutuality	Ben
19	Ecosophy	Jeremy

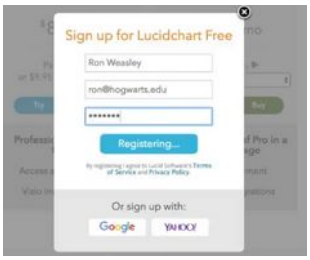
9. Now let's apply those same rules to find primary key for our other entities. Let's quickly jump into the ***order table*** that this order entity represents, and zero in on a specific instance. Here, it looks like our customer John placed an order to go to this address and it shipped at this time. Looking at the data, what could be our primary key? We already know names and addresses are no good and ship date isn't going to work either: two orders could be shipped at the exact same time, so that's not a unique attribute. It's pretty clear that *order number* is the primary key, so we put PK next to that attribute in our diagram.



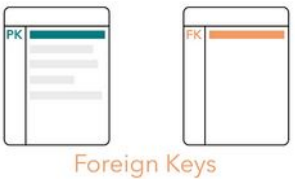
10. And finally we can look at the table that this ***product entity*** represents. Let's say our customer John purchased this product here, a beautiful *Snuggy*. You know by now the *product* ID is going to be the primary key in this table. So we note that in our diagram too. Each of these primary keys is unique, never changing, and never-null.



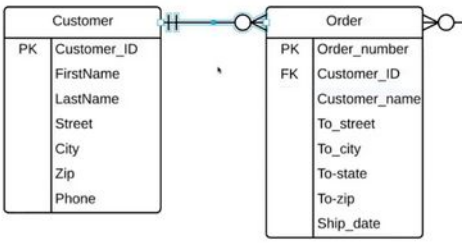
11. So you probably noticed that I'm using a ***diagramming software*** to modify these ERD's, but just know that the same concepts apply if you're drawing them out on pen and paper. But if you want to make things easier on yourself, you can sign up to the diagramming software that I'm using – for free actually. It's called *Lucidchart*. All you have to do is pause the video, click on the card in the top-right corner and sign up for an account. All it takes is an email address and you can follow along with me.



12. So now that we have our primary keys nailed down, let's talk about ***foreign keys***. A foreign key is the same as a primary key, but just located in a foreign place. Maybe you have a primary key in one entity, but it would be really helpful to pull that data into another entity. That's where you get a foreign key. And we want to make note of these foreign keys so we can better understand how our entities relate to one another.

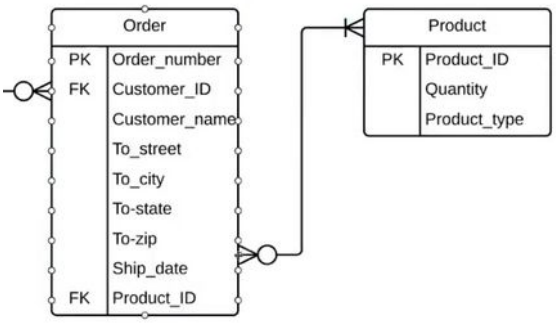


13. Let's see ***how this plays out*** in our diagram. We've already established *customer* ID is the primary key for the *customer entity*, but that same attribute is also over here in the *order entity*. Why? Because for each order we record, we want to know exactly which customer placed that order. The order entity is simply referencing the *customer* ID from the *customer entity*. That makes it a foreign key here, so we'll mark it as such. And we can further show this relationship on our diagram by adjusting this relationship to line up with a primary and foreign keys: just move these crow's-feet to line up with the PK and FK attributes. It reinforces the fact that this foreign key in the *order entity* is referencing the primary key of the *customer entity.*



14. Let's review how this foreign key works by looking at it in the *order table*. So here are all our orders. If we look at a specific order, it has a *customer* ID of 3-0-0-1-6. This is a foreign key because it references the primary key in the *customer table*. Now we know exactly which customer we're dealing with when we look at this specific order. So John made this order here but look farther down. The same *customer* ID is repeated because John placed another order. Unlike a primary key, ***foreign keys don't have to be unique***. They can be repeated in the table. If John is an avid shopper, his *customer* ID is going to be repeated a lot.

15. Another difference between primary and foreign keys is that there can be ***multiple foreign keys in one entity***. Let's say for each order, we also want to know what product is being sold. We'd add *product* ID to our table, and this is what it would look like in our diagram. We just add another field, *type* in *product* ID, and since *product* ID is a primary key over here, that makes it a foreign key in this entity. Now we have two foreign keys in our *order table*.



16. There's also something called a ***composite primary key***. Composite primary keys are used when two or more attributes are necessary to uniquely identify every record in a table. So let's say we created a *shipment entity*, and it's got these attributes here. And then let's take a look at what the corresponding table might look like in order to explain this better. Let's say John ordered a regular *Snuggy* and then a zebra-pattern *Snuggy*. It's all one order, but they're getting sent in two different shipments. When we look at these two rows in each of these attributes, we wouldn't be able to rely on any single one to give us a unique record.

Shipment Table

	Product_ID	Order_number	ChargeCardTime	PackingTime	ShipOrderDate
13	49225	252349915	6/1/2017 9:13:34	6/2/2017 10:14:46	6/3/2017 11:15:52
14	40807	252349916	6/1/2017 9:14:16	6/2/2017 10:15:02	6/3/2017 11:16:03
15	76342	252349917	6/1/2017 9:14:01	6/2/2017 10:15:26	6/3/2017 11:16:13
16	96893	252349918	6/1/2017 9:14:21	6/2/2017 10:15:39	6/3/2017 11:16:19
17	69246	252349919	6/1/2017 9:14:34	6/2/2017 10:15:41	6/3/2017 11:16:47
18	69253	252349919	6/1/2017 9:14:34	6/2/2017 10:15:45	6/3/2017 11:16:47
19	99002	252349920	6/1/2017 9:15:07	6/2/2017 10:16:07	6/3/2017 11:17:11
20	64382	252349921	6/1/2017 9:15:14	6/2/2017 10:16:07	6/3/2017 11:17:23
21	91514	252349922	6/1/2017 9:15:33	6/2/2017 10:16:28	6/3/2017 11:17:23
22	64244	252349923	6/1/2017 9:15:33	6/2/2017 10:16:50	6/3/2017 11:17:41
23	94251	252349924	6/1/2017 9:16:17	6/2/2017 10:17:05	6/3/2017 11:18:00
24	69253	252349925	6/1/2017 9:16:21	6/2/2017 10:17:17	6/3/2017 11:18:04
25	94166	252349926	6/1/2017 9:16:38	6/2/2017 10:17:29	6/3/2017 11:18:10
26	44199	252349927	6/1/2017 9:16:41	6/2/2017 10:17:30	6/3/2017 11:18:16

- The *product number* is duplicated when someone else buys that same zebra-pattern *Snuggy*; so it's not a unique record in this table.
- An order could be parsed into a couple of different shipments, like John's was; so *order number* is also not unique.
- *Charge card time* would be duplicated if someone else pays for their shipment at the exact same time.
- And same for *packing time* and *ship date* – all not unique.

So it doesn't look like any of these attributes will give us the primary key we're looking for.

Shipment Table

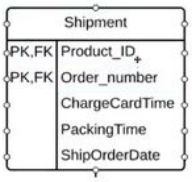
	Product_ID	Order_number	ChargeCardTime	PackingTime	ShipOrderDate
13	49225	252349915	6/1/2017 9:13:34	6/2/2017 10:14:46	6/3/2017 11:15:52
14	40807	252349916	6/1/2017 9:14:16	6/2/2017 10:15:02	6/3/2017 11:16:03
15	76342	252349917	6/1/2017 9:14:01	6/2/2017 10:15:26	6/3/2017 11:16:13
16	96893	252349918	6/1/2017 9:14:21	6/2/2017 10:15:39	6/3/2017 11:16:19
17	69246	252349919	6/1/2017 9:14:34	6/2/2017 10:15:41	6/3/2017 11:16:47
18	69253	252349919	6/1/2017 9:14:34	6/2/2017 10:15:45	6/3/2017 11:16:47
19	99002	252349920	6/1/2017 9:15:07	6/2/2017 10:16:07	6/3/2017 11:17:11
20	64382	252349921	6/1/2017 9:15:14	6/2/2017 10:16:07	6/3/2017 11:17:23
21	91514	252349922	6/1/2017 9:15:33	6/2/2017 10:16:28	6/3/2017 11:17:23
22	64244	252349923	6/1/2017 9:15:33	6/2/2017 10:16:50	6/3/2017 11:17:41
23	94251	252349924	6/1/2017 9:16:17	6/2/2017 10:17:05	6/3/2017 11:18:00
24	69253	252349925	6/1/2017 9:16:21	6/2/2017 10:17:17	6/3/2017 11:18:04
25	94166	252349926	6/1/2017 9:16:38	6/2/2017 10:17:29	6/3/2017 11:18:10
26	44199	252349927	6/1/2017 9:16:41	6/2/2017 10:17:30	6/3/2017 11:18:16
27	40759	252349928	6/1/2017 9:16:49	6/2/2017 10:17:44	6/3/2017 11:18:25
28	39668	252349929	6/1/2017 9:16:52	6/2/2017 10:17:53	6/3/2017 11:18:31
29	71292	252349930	6/1/2017 9:16:59	6/2/2017 10:17:58	6/2/2017 11:18:43

17. But what if we took two of these attributes, and combine them to create a new unique value? Like *product* ID and *order number*. Take those two values for any instance, combine them together, and you've got a value that won't be repeated. That's a ***composite primary key***. Now the primary key for the shipment is different than the primary key for this shipment. You could technically call this a "compound key" because we're using two foreign keys, but usually people just say *composite key* as an umbrella term.

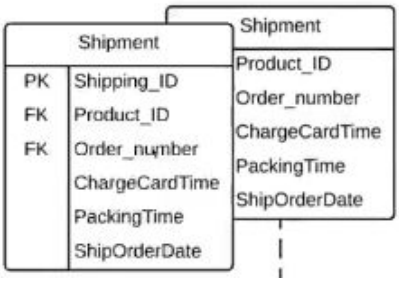
Composite Primary Key

	Product_ID	Order_number
13	49225	252349915
14	40807	252349916
15	76342	252349917
16	96893	252349918
17	69246	252349919
18	69253	252349919

Anyway, jumping back over to our diagram, you note a composite primary key with multiple PK marks. That doesn't mean there are two primary keys and means that both of these attributes are needed to create a composite primary key.

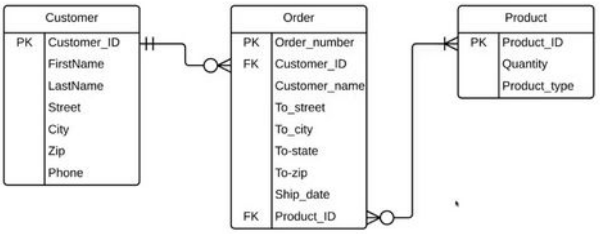


18. A ***couple of rules*** here when making these:
    - One, use the fewest number of attributes as possible.
    - Two, don't use attributes that are apt to change because that can make things messy.

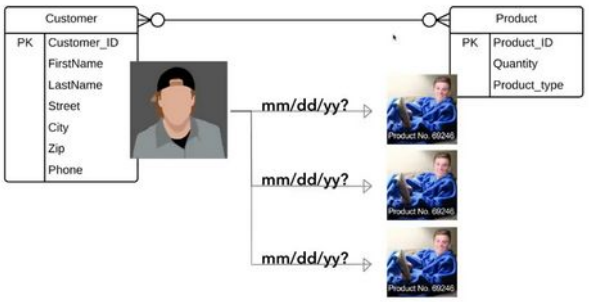
    

    And you might ask yourself, why not just create a *shipping* ID attribute so we don't have to deal with this scenario? Well you certainly could, and that's totally valid. It's actually quite a ***debate*** among some people. if you should even bother with composite primary keys. But it all depends on the database you're creating, and if there are certain scenarios where a composite primary key makes sense.

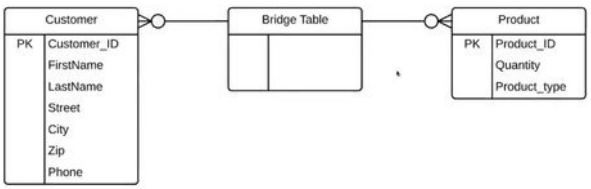
19. Another thing we can talk about here is something called a ***bridge table***. When you're making your ER diagrams, it's important to ask yourself: "Is there anything else that I should be recording into the database?" Sometimes you'll have two entities connected to one another, but there's more going on between them then you're accounting for. That's when you need to use a bridge table.

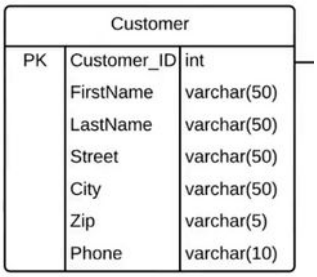
20. To illustrate this point, I'm going to strip down our diagram to just *customer* and *product*. Now couldn't we just create a ***direct relationship*** between these two entities? A customer can purchase zero or many products and a certain product can be purchased by zero or many customers. Sure! Conceptually, this works. But the way this is set up, you're not going to know when a customer purchased that product. You're not going to know if they bought all of them at once, or if they returned to buy more at separate times. You're going to be in the dark on many details regarding the interaction between these two entities. And this issue often occurs when you've got a *many-to-many* relationship.

21. That's where a bridge table comes in. A bridge table allows for an ***intermediary one-to-many relationship*** and gets you the information you're lacking. In this example, a bridge table would be the *order entity*. Let's put that back into the diagram. You can see how it breaks up the *many-to-many* relationship, and now, each time a customer purchases a product, we're going to have a record of that interaction in our *order table*. So as you build or modify your ER diagram, make sure you're asking yourself if you're getting all necessary information, and use bridge tables to capture that data.

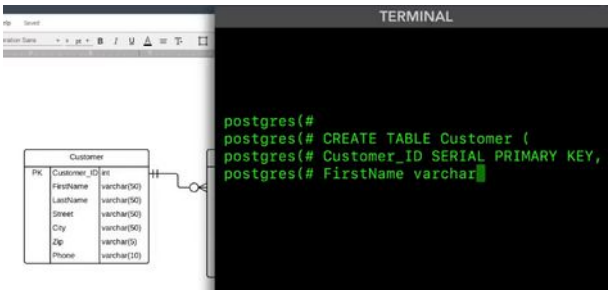
22. If you want to start getting a bit more specific with your ER diagram, you can do so by adding ***data types*** for each attribute. This can be really helpful, especially if you'll be doing some programming out of this diagram. So we use an entity shape with three columns, and we'll specify the type of data we want on the far right side.



- For *customer* ID, we'll return an integer; that's going to give us a number like 5-0-1-4-2-2.
- And all these attributes here will be *varchar*, meaning you can use various characters and then set a character limit in parentheses, like this. So for phone number, for example, it's going to have to be ten digits to fit the standard phone format.

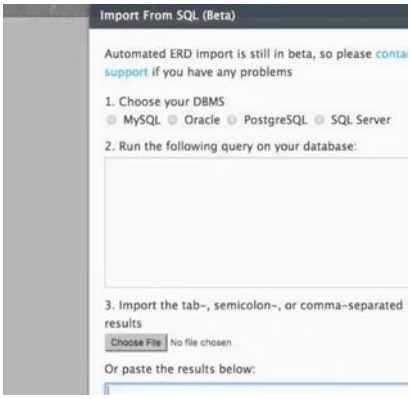
There are several different data types and things we won't get into, but just know that you can include these in your ER diagram if you want to get more technical.

23. Now we've added a lot of detail to this ERD but what are we going to actually do with it? Building ERD's is a great way to conceptualize and visualize before you've actually built your database. But once you've built the whole ER diagram, you're going to need to translate it into a ***database management system*** or **DBMS**. Typically, you would have to just reference this diagram and then manually enter code to create the database. That's a lot of work!



24. But here, *Lucidchart* allows you to ***export*** the diagram and it will automatically generate the code you need for your database. You can see the table commands here with the entity it's referencing. Here are the attributes for this entity with the data parameters that we set. And it also notes our primary and foreign keys. Then you just specify which database management system you're using, copy the code and paste it in. Your database is ready to go. It's a total time-saver... and if you're a student with an assignment to make a diagram in the database, this is like cheating. It's so easy!

25. Conversely, let's say you've already got a database created, but you want to visualize it. Anyone who's worked in a database management system knows that it's not exactly easy to see what's going on. But a ERD of that database gives you a visual layout. Well, you can just ***import*** the database here... and *Lucidchart* populates all your tables. Drag one out and it automatically creates the entity with the appropriate attributes, primary keys, and foreign keys. And if you drag out another entity, the correct relationship will automatically connect between them. This is another huge time-saver: you can quickly visualize your database, make changes in diagram form and then pass those changes back into your database via code.



26. Thanks for watching this tutorial on ER diagrams. Please subscribe to our channel below to get more helpful tutorials heading your way. Also, we're always curious: what other tutorials do you want to see? Please comment below and let us know. And don't forget, sign up for a free *Lucidchart* account so you can start creating your ER diagrams today.