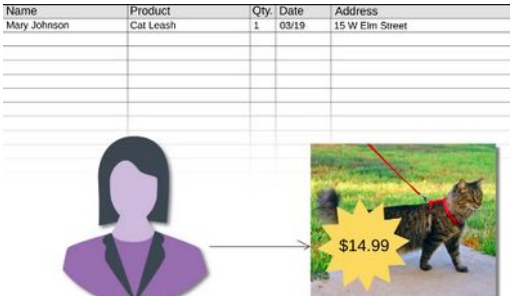
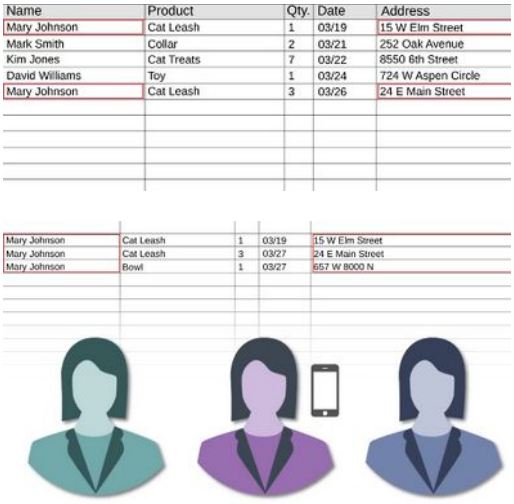
Database Tutorial for Beginners [1]

1. Hi, my name's Taylor and I'm going to give you a ***high-level overview of database architecture***. Understanding the big picture and principles of database management will help you as you're getting into the more technical concepts like ERD further down the road.

2. So let's say you're opening your own online store for something like specialty cat accessories, and you want to be able to keep track of all sorts of information surrounding your sales. Most people would just open a ***spreadsheet*** and start putting in things as orders come in. Maybe it would look like this. Mary is our customer, she buys a cat leash and you capture all this information. And you just record this information for each of your customers that trickle in.

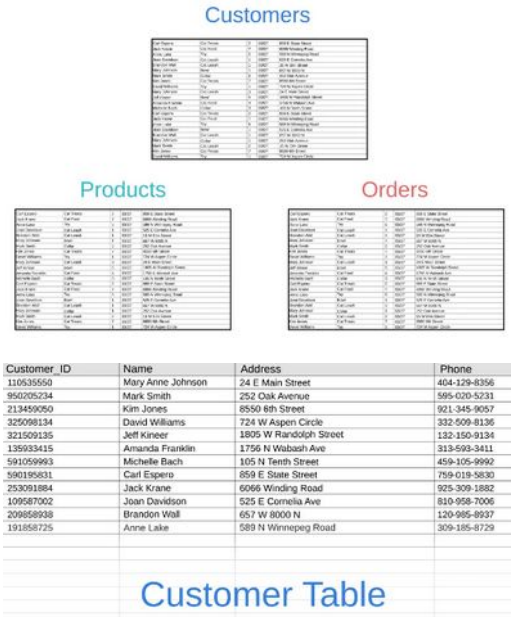


3. But maybe later Mary buys something else, like three cat leashes, and at this point she has moved to a bigger place to have room for all her cats, so her address is different. Now you've got redundant information, some contradicting values for your customers address, and if your specialty cat store got enormously popular, these issues would just escalate. Mary calls to ask about one of her orders, and when you pull up her information, you get three different addresses, all these disorganized orders, and you're not even sure you're looking at the correct customer because there are three different Mary Johnson in your spreadsheet! You could see how this might lead to a ***messy situation***. Shipments could get sent to the wrong place, customers might get mixed up, and the wrong products could get sent to the wrong people. So how would you resolve this?



4. Instead of having just one massive spreadsheet, you would ***separate the information*** into different bite-sized tables. So with our cat store for example, you might start by creating a table that lists out all of your customers, and then a separate table that lists out all your products, and then another table that records each of your orders. This separates the data you're pulling in a much more efficient way.

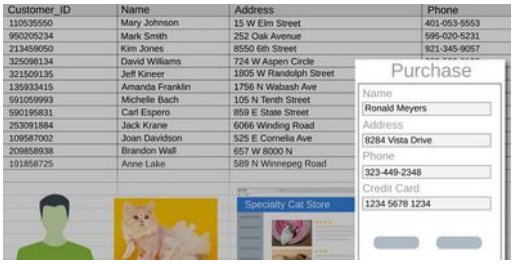
   So here's what your *customer table* might look like: you can see we've got Mary again, but now she won't be repeated into several different rows. Any change to her address, contact info, or even name can be updated in this one consolidated place. The *product table* would have all your cat accessory inventory. As you add or remove items, this would be the one place where you would make those changes. And *orders* would keep track of every single sale you make.



1 https://www.youtube.com/watch?v=wR0jg0eQsZA

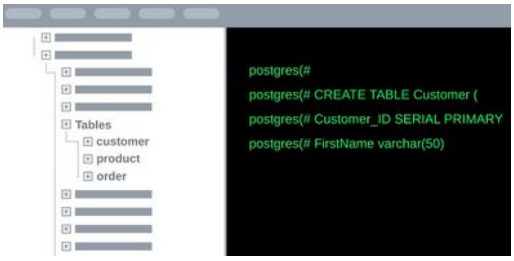
5. Now these tables, although separate, have ***connections*** to one another, and it's these connections that form the database. Let's dive in and take a closer look to see what these interactions look like. We will start in the *customer table*. Let's say someone goes to our online store and makes a purchase. It's a guy named Ronald and he's in the market for a cat costume, and buys one from our store.

   - When he checked out, he entered all his contact info, and we've recorded it in this customer table and assigned him a *customer* ID.



   - Let's move over to the product table. This lists all our inventory, and here's the cat costume he wanted. We keep track of it with a few fields here, like *product* ID, *quantity in stock*, and *product type*.
   - And then when Ronald actually ordered the cat costume, we recorded that specific purchase information in the order table. Here you can see we pulled in the *customer* ID from the *customer table*, so we know it's Ronald. We also pulled in the *product* ID from the *product table* so we know that he purchased this cat costume. And there's other data in here that tells us about the date of the sale, shipping address, quantity, etc.

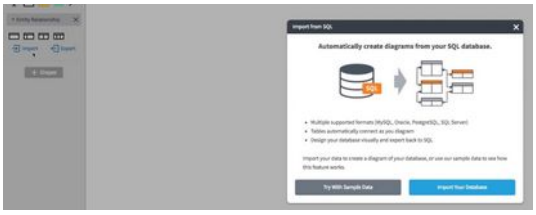
6. It's pretty obvious that this system we're using now is far more organized than our single spreadsheet from earlier – not to mention it's far more robust at handling large amounts of data and information. But database management systems typically aren't very good at visualizing the tables and connections within a database. It's all in the programming language and it's hard to see where the connections are and where improvements can be made.



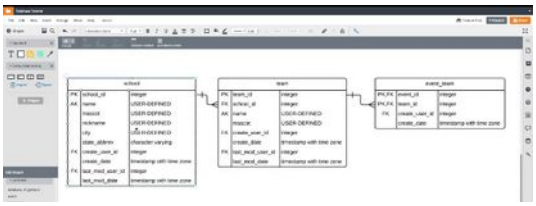
7. That's where ***entity relationship diagrams*** come in. It's a visual way of looking at your database structure. Each table translates into an *entity* and your column categories, like customer name, address, purchase date, etc. are listed as *attributes* in their respective entity. Finally, the programmed connections between your tables, like how Ronald's order referenced a specific *product* ID and his *customer* ID, those are visualized through *relationship lines*.

8. So imagine if your database was far more fleshed out than our simple example, like if you had separate tables for shipping address, billing addresses, credit cards, shipping info, etc. Trying to make sense of a large database when you're in the database can be very taxing.

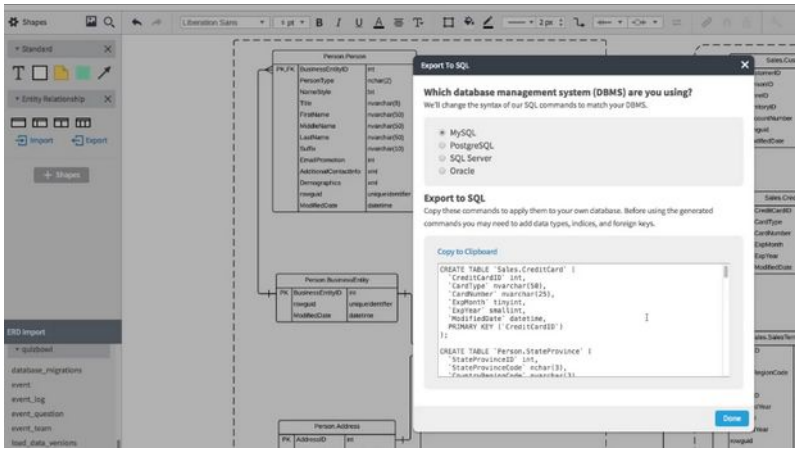
It's ***much easier to visualize*** it through an ERD and that's a super fast process with *Lucidchart*'s ERD import tool. Just run a query of your database and *Lucidchart* automatically imports the tables that you can then drag out as entity shapes. And the relationships between entities automatically connect as well.



So you quickly create a visual representation of your database and then it's so much easier to spot database errors, you can see where you're getting duplicate data and it's way easier to onboard someone who's new to your database. They can look at an ERD and see how the whole thing works.



9. On the flip side, let's say you don't have an existing database, you're ***starting from scratch*** and want to build one... Well ERD is a great tool for concepting: you've got an idea for how your database is going to work and you flesh it all out in a diagram.



And the awesome thing is is that when you're done concepting, the diagram itself can be translated in the code that forms the actual database. You don't have to manually recreate your concept in database form. The entities automatically transform into tables, the attributes to columns in those tables, and your relationships get translated into coded connections.

10. Hopefully this gives you a bit more context as to ***why we use databases*** and how they relate to entity relationship diagrams. To learn more about *entity relationship diagrams* and our other tutorials, like *primary keys*, *foreign keys*, *attributes*, and *cardinality*, click here. Click here to start creating your own ER diagrams today!