So today we're going to talk about database design and database design. It's a kind of. It's both a theoretical talk and a very pragmatic talk. And we're going to bounce back and forth between the two. Or hopefully we will. I want you to notice a few like one of the things that we always have a. An issue with in terms of module 2. Is that we spend a few days away from Java, it starts receding and so one of the things that I II try to do with bring you back. You'll be brought back hard tomorrow, but if I try to bring you back to the ideas that we have in Java and how they relate. So database design. In a relational database, it's a lot like creating your class design, and when you're trying to solve problem. But there are some big differences. One of the things that you will see is. We will talk about now. Alright, and we talked about nouns and how you can tell with nouns that it's probably a class and things like that in Java. The difference is a database is just, it's just the data, it's the data and the relationships, so it doesn't have any methods. It doesn't have any verbs. The verbs would be implemented by your your middleware, which is your Java. Right and. As part of that, I also want to focus on where we are in this whole big picture. Kowert going trip. Back end of the database the the back end of a full stack is in a sense the the database. That is where all the data is stored. That is the the the persistent part of your application. Java, which we taught first, is really the middle. Java is the part that that allows you to interact with your back end database. End allows you to push data forward to your front end web design whatever you whatever your front ends OK and in our case that's going to be a web front end. It could be something. It could be a device or something like that, but in our case it's just a web print. So we are. They learned a lot about Java because that's sort of the biggest thing you guys need to know to get through this. And then we're talking about the databases, and then we're going to go back into how to make Java work with your database. And when we've got all that working, then we can start talking about later in the module. We can start talking about API's and how you interact with the world with your Java and with your back end and everything else OK? so the first thing we're going to do today is we're going to share my screen with cord OK. So we're gonna look at this. I don't know how many of you managed to get through the tutorial or take a look at it. But yeah. It Oriel went through a somewhat similar exercise, but the idea that we have here is that we are tasked with. This is often the way it will be presented to you. We are cast with something where we're given some sort of, you know. Bring voices or displays or or screens or something like that. And you are then told what the customer requirements are and you're supposed to then come up with a design. Now I say this. This is a little bit. Is a little bit untrue because the reality is, as junior developers, nobody is going to ask you to divide and design A database. You just need to understand the process that is going through. At some point you may get to the point of designing databases, but it's not likely to be in your first job. But I want you to understand the process so that you understand why things are made the way they are made. And with that in mind, all the other thing today is that. It is important for you to know the level of SQL that we have taught up until today. Today we're going to learn data definition language and you don't need to know it, so you need to be know about it. You need to recognize it. But almost nobody except database administrators, memorizes and stuff. If you ever are asked to do this, you should go back to an existing one. And say OK, what is in that existing one? How do I you know? How do I do the same thing over again? There's there's just you're not going to be in a position of creating your own database. You might OK, you might create a database for a side project or something like that. Again, go back to the United States SQL database that we keep using and things like that and and see how it's done there and we'll figure it out. But don't don't waste a lot of time trying to memorize any of this. This is not what's important. What is important is understanding. How we look at something like this. This is a art gallery and it says the guilarte. Can everybody see my screen? Just wanna make sure. I feel art gallery wishes 20 to maintain data on their customers, artists and pinks and have several paintings. But each artists in the gallery at one time paintings may be bought and sold several times. In other words, the gallery masela painting them bye aback at a later date and scale from our customers. So what do we know? Already about our like if we were to try to start the design process. What are the nouns involved? Or an artist. Art and artist OK customer. Yeah. Date I guess, and then price date and price OK. So you're gonna need. That's alright phone number phone number. The address contact information could be. OK. Let's let's say it is phone number and address is 2 separate things but OK. So let's let's write a few of these down, and then this is the process you go through. So what are the entities we have art? Which we're going to say here. You know what? Let's, let's do this as a list. 'cause we'll shift it around so, but. Clip art. We have artist. We have customer people said. We have address. We have phone number. We have a price. We have date of transaction. Probably under art. Well, we have the title right here. It's a larger stay there. OK, let's say. If the art the title or is it? Is it something more? It's got an art in an artist. There's going to be something between those. Let's start with this. This is a OK. So the very first thing we do is kind of keep track of what are the pieces of information we need here, right? And that seems to be all the pieces that I see. That's all that I see. Anybody see anything else that I'm missing? OK, so then you try to say how am I? How do I bring this up in a way that I figure out which are the core entities? Which are the things I need to react to? And which things are dependent, like which are the main entities and which things are simply dependent on those? Right? So for example, what is something that is? Like if I say if I. Pick one of these. Basically the customer for example. Right? Rid of the books and then. Basically the customer. Which of these things are related to the customer there? Things that are more attributes than separate entities. Particularly, which things that might I in this particular context need to deal with independently. Other contact information. Well, OK, so this is the contact information a thing. Or is it related? Related, it's related OK. So for the moment, let's say that the contact information is related, so we will make that. Give myself room. Make. We're going to say the phone number. The address. Shift is down so. Phone number address of sort of related information information. Allow myself to straight through, probably not. OK, what else do we have as an as a as a main entity or a related entity? Manage sheet looks like it would be the artist, the artist and the related entities would be the art titles that are made by that artist. The Pride and the data transactions. So oops, yeah alphabetically here so we. OK, so the artist and we think the related information is is what? And that it's OK to add new things now that we know like we have an entity. So the art or the art title. And then the the going price for it. Uhm, then I think we we could. Potentially I don't know if we want to include data transaction here or use data transaction like a like a link table or something like that. Let's put it here for now, because it shows, well, some of the issues, but OK. Alright, so now we have the price and the date of transaction. And we. The artist. And the art. We have the customer. So how we accounted for everything we counted for all this stuff with the artist, the customer except for their relationships between. We've, we've accounted for that things right? And think about this from the point of view of the art gallery. Or the point of view of the art gallery. What is the relationship between, say, a piece of art? And the. The purchase date and stuff. Like one piece of art may be purchased multiple times for different prices. OK. So if we're going to try to store the information in this. How are we going to water? We gonna start we're gonna start the art titles which is first of all, we're going to be multiple values here. So do we have a row we have the artist's ID and then we have an art title. What goes in an actual row? Looks to me like we have for an art for a single art title. We have A1 to many relationship inside the table. Having a one to many relationship inside the table is not ideal. Let's put it that way. OK. Once. Should we then have the art pieces themselves be up at an entirely separate table from the artist database? OK, so we could do that. And by the way, just to be clear, this is this iterative process. Is the way you're supposed to you're you're not supposed to get it all at first. The idea is to start putting down the pieces and start adding things to them, and then as you run into dilemmas. You you resolve the dilemmas and that is that is all part of what we call normalization. We are normalizing the data. And normalizing is the process of taking our data and making sure that we have a. That it is not repeated. We don't ever want our data repeated. OK, and that it is related to related well. To the table that we have in to the end, particularly to the primary key. And that it is. Both related well to it and it is not that the that we have set up a table where a primary key is never going to change. Ashley goal. Hi. If you guys looked in the reading and in the tutorial it well mostly in the reading they talk about. First normal form, second normal form and 3rd normal form. I will heads up just for all of you. Nobody really cares whether you understands the nuances of third normal form. But it is a common question on interview. People want to know that you are familiar with the concept of third normal form. I have been caught, you know, aware of the concept of third normal form since the 1980s. I am unable to articulate it. You know particularly well to this day and it doesn't make any difference. I understand the concept. I understand we're going to war and they're. They're like I because I read about it yesterday, and because I did, you know, LinkedIn. Course few days before that on database design, I now could tell you a little bit more than I have been in a long time about third normal form, and I won't be able to by Wednesday, alright, because it's just not going to ever stick in my head. The important part though is you need to recognize the term third normal form. You need to understand that concept and the concept is that we're not repeating data and that things are well formed in tables that that address the needs that we have. For our particular you know, organization. And that whatever is in the table is directly related to the primary key. That's the most important part. It should only be directly related to sort of. When I say to the primary key, I mean to the primary the primary key as the embodiment of what that table is about. So when we look here. When we say OK. We're going to separate out our art and our artists. It's because. The art. Alright, he and we have different ways of doing this. It art is. I has a. 1/2 OK, what does it have? We have an artist and artists may make many paintings. And or maybe make many works of art. And the art may be sold, bought and sold multiple times. But does the you know our concept here we're going to assume there's one artist. If there's just one artist. Then there is a one to many between the artist. And the works of art. So. Just a moment I had make sure I've got what I need. So the suggestion was that maybe our art. Should we separated out? And it will move things around, but what what do we need to what should be under a piece of art? Artist. The artist. K. The price. Turn off. Where does date of transaction go? Well, that's an interesting one. Where does data transaction go? Because it is. Is that relevant to? The artist. Or is it relevant to the? His action. I would say it would be the transaction 'cause you're talking about selling the actual physical art. Right? So from the point of view of our of our art gallery. Maybe what we really need to have is a a transaction. Or you know invoice or something like that. Let's call it a transaction for now. And then think about. Is really is the the price? It's not static for a piece of Earth. Alright, it's at the current price. Like what is the? What is the thing that that gives us from the art gallery's point of view may be valid. I'm I'm not arguing it is and I'm just trying to think how we how do we relate that? It would be like the value of the actual piece of art and then the revenue for the art gallery, right? OK, like that. Price of art would represent on sale the revenue that the gallery will take in. OK. So probably from that from the gallery's point of view. Now we do need to know the date of transaction. And we need to know what the price at the time of sale at that at that point. This price is a little different than that price. This is now the. Current and this is now the. Market. I'm just gonna say at. At transaction date of transaction. Like like let's not worry too much about the. I think the difference between what the. What they make versus what somebody else makes. Let's let's keep it simple, but. But this is more of a. This now is more of an inventory issue. The frame, the current price which says I got these things available. At this price, this transaction now is the price that I actually sold it for. Especially, you know some of these like yeah, you're particulare price somebody comes in and they offer you something else. You might accept it even if it wasn't the original breaks, because you might say you know what? There's a hot customer here who you know wants to buy it. Or this is a really good customer who. Likes to come here alone. I'll give them that special price you. It's the price they actually pay. You know, three of them are all standing there haggling over it, and maybe higher than the current price. Does this allow us now? This is a concept between behind. Does this allow us to now handle the kinds of things that a an art gallery is likely to be able to do? Are we able? First of all, if we have a table with each of these things, and remember, I guess the. Art titles. We took our and we put it into art, right? So do we have our titles still here and price? And still in here now like they all of this like what? What is it that that remains in artist? Just the title of their art, right? No, do we have? What do we have? Is it the artist name? Is the artist name? As far as I know I mean. OK, and so let's now because we're going to think database wise. So the primary key here. I mean, it's possible. I have two artists, both named George O'keeffe. So if I do, I probably better have an artist ID. Try and artist name. Now we need to be able to find what our is by that person. But do we need it in the artist table? And that's where you start getting into the way normalization works. I shouldn't need it in the artist table. I should be able to get to it, and the way that I get to it is that this. Is really not going to be an artist name, but it's going to be an artist ID. Because then I can query the art table. If I wanna get water the queries that I want, what are the things I might want? What art is there? By this earnest well, I I what I do is I find the artist name. The primary key is artist ID. I go to the art table. Look through all the the art. OK and OK, so this is now the art artist ID. The art itself must have a title. And you just start iteratively breaking this down and and the processes that's very important is at each step you start saying. What's the question I might ask of the database? What are the things I might do with the database if I want to go out and say what are? Is there available by this artist? I can now query the art table. Well, I first of all find out the artist name from the artist table, find out the artist ID, then go to the artist thing and and I can say OK. Now what happens if I want to say what art have I sold already? By this artist. I would say that in the art table you should have like a Boolean, UM, for having it been sold or not, because you can sell pieces that have already been sold, but it guess it could be a Boolean like if it's, you know in the gallery itself, in the available available. Available. OK, that's gonna tell you what's actually like. I might track art that is no longer in my. They almost certainly need to track art that is no longer in my. Inventory. Because I need to know in the transaction. Now let's OK. So what is the primary key of art? There should probably be like a a like an arts Q or an art ID. How it needs to be an art ID? Because art is often, I mean heck, how many things are titled, untitled? You know it's it's. OK, so we may need an art ID. So then I look at a transaction and I say OK. The transaction probably needs to know what does it need an artist ID. No, but they are ready. Because we know if we have the art ID we can go back here and we can find the artist ID. And normalization is all about not repeating that information by put the artist ID down here. And this is very important. I want everyone to get this. First of all, let's say there's a transaction ID. Just because we'll worry about what that is. If I put artist ID. Then I'm duplicating information. And I'm also making my design more difficult, and the reason I'm duplicating is because I already know that this art. Is related to this artist. By cutting the artist ID down here I am making it available in two places which you know. Occasionally you will decide to go in that direction. By the way, for performance reasons. I'm not saying that you would never do this, but the goal is to not do this. The goal is to say OK if I have an art ID. I think you know the artist idea is not. An important piece of information anymore because it's already contained, so it is duplicated, so I do not want to have that. Now. Who doesn't understand why I wouldn't want the artist ID here? It's fine, I don't try be clear. Anybody who goes, but it would be a whole lot easier to just know who the artist was here, OK? Series on the artist name. You know the artist name, because you can go from the art ID to the art to the artist if you need it. So we don't have any information which is not necessary in the in any one table. Instead, we use the the relationships between the tables. OK. Now where how are we gonna keep track? Of. Which customer bought this? Customer already. Right and where we put that under customer and transaction. Right, so we're gonna start with here we're gonna have a customer ID. And since we have a customer ID, is that going to be the name? Probably not, because again could have two John Smiths. So down here, we're going to have a customer ID. So now with this we could say alright. If I wanted if I wanted to create this right here. And I wanted to go through and and create this invoice. To have the pieces of information that I want. I'm going to give you a customer name. Or I'm gonna give you a customer. Let's say I'm gonna give you a customer and I'm gonna say what is water all the purchases they have made? Sort of, where is the query come out in? So I have the customer so I know the customer ID. What are they gonna look at? Remove look at the transaction and link link it with. The transaction ID and all that other data, right? So in other words, I my basic query is I'm going to be able to say something along the lines of. I have my in order to get the top part of this in order to get the the the name and address and whatever else. Then I would you know. Select customer. We're assuming I know the customer ID. So select customer name. Phone number. Address. From customer where? Customer ID equals whatever our value is. Question. So my wife? Later because we will deal with those soon. OK, that would give us the top part of this. And then you would go down and you would say OK. So in other words, select. Let's give it say select star from. Transaction. Where Christopher? Why do people? Order bye. Big transaction. How printing out are we doing it? Yeah, 917, we're doing it in ascending order. So the next step in this is always to say, OK, how do we do the things that we need to do? The next thing would be to say, OK, what happens? What are the the the actions that we might take is really start getting to sort of the Java thinking. What are the actions we might take and what are the queries we might need or the the adds, updates, and deletes that we might need. As we get to this next stage. OK, the next thing we do is OK, what if we want to sell a piece of art? To a customer. Well, why do we need to change what? What are the? What are the basic transactions? Don't you have to spell it out? But what are the basic? Pieces of data that we need to change in our tables to reflect as we sell a a piece of art to a customer. We tend to update that it's no longer available in the price, potentially. OK, so we have to and then it's an interesting thing is the price. It's the last price it was sold at then. I mean, I'm not really sure what we do with the price at that point. It probably it's like a house. Then you might have a list price, but as soon as you sell it you would change it to whatever price it sold for, because that's the most recent price you have for it. So yeah, you might want to go and update the art table to say it is no longer available and. To set the price to whatever you actually sold it for, which might be the same but it but just to be sure, whatever price you sold. What else would you have to change? Pass code would you have to change the date artist for the customer tables? Yes. Presumably wouldn't have to change the artist, because that didn't change. Customer table, it might be a new customer so you might have to add a customer before you did it. So you might end the transaction. You're gonna need to add a transaction. You know the customer, even if we just created it. We know the art ID because the thing we're selling we know the date the price is going to be whatever we record. If you say it's always sold at list price, I'm ready to take the current price and assume that there's some other process for changing prices. And it'll auto generate a transaction ID because presumably. Now that transaction ID just to be more real-world. Maybe an invoice number? OK, so this is. This is the process that you go through. As you try to to say, can I add count for things? What if an artist brings in a new piece of art and says you know I have this new thing? What do you have to add and you just go through in your head. Each of the things and the art gallery might have to deal with and say do I accommodate? Those things. There's a later process where you go now, unless you're really, really successful art gallery. You don't have to worry about, you know the performance on this one because you're not likely to be selling so much heart so quickly that you're going to have to deal with it. But if you were dealing with, you know, selling things. Add wayfair.com or something like that. Then performance becomes an issue and you have to figure out where things are stored, partly in terms of where. How do you get to the vast stuff and things like that? We're not going to worry a lot about that in here, but I will tell you that the process that you go through. Normalization? Is a a theoretical design, you know, in a perfect world. And then there is sometimes when people will say they want to denormalize data. The reason they do that is because there is a situation where even though they are moving away from normalization, even though there may be moving away from the ideal of one piece of information only being. One place. It may turn out that they don't, they just they cannot afford to hit the art idea. Go to the art and the art. Go to the artist to get the artist every time and they need to put the artist name in here or the artist name keeps changing because you got one of these freaky new world artists who likes to call themselves a new thing every week. Whatever the reasons are, you might have a reason why you would denormalize data, but that is a conscious decision that you make. So let's go, take a look. I intentionally did not look at it because I like to challenge myself on these things. Let's go look at. I think this is the art gallery. Let's look at hold on second. I gotta find out where to put it. There we go. Look at the design that they made. And we're going to talk about the. We're going to talk about how this this works. The language that you use for dropping and creating things. And actually, you know, before we do, let me just show you, by the way. If you go to one that you are more used to, for instance like let's go to the United States database, something you should know if I go to most of the simpler right now. This is their there. The two important ones at the database level. Are dropped which deletes the entire tease. And create which creates the entire database. We mostly will not deal at the database level. The reason why we go ahead and tell you to create the database manually or whatever is mostly with Lui are going to deal with the design inside a database does already recreated, but just so you know there is a create integration that droppings. The level you will mostly deal with is at the level of tables. So let's go to say, the city table. Alright. And use drop table and

|  |
| --- |
| Again, you don't have to memorize these exactly, but it used to be idea. Well this auto generates if I I can create things like create columns or rows. It's going to auto generate this is the DDL that would be needed. The data definition language that would be needed to create the table that I may have created manually through. PGA It always puts the drop table here, but it comments it out and the reason it does that is because, like you may have an enormous database, we don't want somebody accidentally running this and dropping the table when we aren't intending to. Add all the data to it. So be very careful drop table. It's just a simple little drop, people whatever. And boom, everything you've got is gone. So again all the cautions in the world. Be careful. Both drop table and create table have these additional things that are if. I gotta write this somewhere where I can do it. Drop table. If exists. Alright, if you're going to create a dated definition for your database. And like in your case, in the case where you're trying to recreate the databases from scratch. I talked to you before about the idea of item potent. Farmview maybe I can't remember. I talked about it in class, but the idea of something this will become important and later think if something is item. It means that if you do it multiple times you will wind up in the same state. So for instance, if I insert a row into my united, you know when I when I rang this row I was inserting the city? You know Cicely, Alaska or whatever it was and I was inserting into the the data into the whatever table the city table. But I ran it twice. There were two Sicily last. So the insert into is not item because you don't mind if you run it more than once. That it is going to. You're going to wind up with an end result that is different. It doesn't mean that the thing the very first time you run it. You won't change things. It means that if you ran it two times, you would end up in the same place as if you would run it one time. Right, so the idea behind a making your DVL item potent. Is that? If I don't put in this, if I leave this commented out when I try to create the table. It will create it. If it doesn't exist, but if it does exist, it'll give you an error and will never get down to the point where you can make alterations on it, or at certain things. So to make something item potent and make it happen, able to do it over and over again. This is the reason why I gather the knighted States and I could over and over again. Do things was because they put in something like the drop table if exists so that it will get rid of it all and start from scratch every time. Since you get rid of it all and get started from scratch, you always wind up with the same result no matter how many times you buy. And for example like this, it's very handy to make sure that I'm going to always, you know, I'm gonna always get back to a set case. You may have noticed that. When you run your tests over and over again. On the homework for last few days. Mostly for for Friday. When you run it. You run the test, it's it's always starting from the same place. Like otherwise, it would be causing errors the second time you were trying to add things or when you were trying to delete things or whatever else. They actually run as part of the test. They run an entire DDL through. As as part of their APA, four kind of test thing. So you always start from me the exact same point. And that's that's where when you see these, when you look at the United States, whenever you always see they do the drop tables first. Four or if it if you don't, by the way, if you don't say drop table if exists, it will just give you the error that says it doesn't exist. Unfortunately, drop table is not. It it doesn't. Stop if you run through it. So if you don't put the exists then it just. It'll give you the error but it it'll. It'll just a warning, OK. So now let's talk about. |

Create table is where we try to take the thing we just said here. And put it into a language that the. PG Admin and Postgres will understand really Postgres, but it will understand. So we have to give it a name as I think I mentioned before, everything in here is really part of public. That is the schema. So when I open this schema and you see public right there, everything is really public dot city. You don't have to specify it because they're all inside it. There are there are other things you could have at the schema level. Don't worry too much about that. But in your ddlj will say public does. And then we're going to give those individual things that we're used to. If we look at the columns. Here. OK, we have. City look at the columns. These are the things that's creating city. It is saying it's an integer, so we give it the type of thing. Or character varying 50. If you went to the city name right here and right clicked so the properties. The definition is that his character varying and has a length of 50. That length, by the way, is a maximum length. It doesn't say it's. You just said you can't get longer than that, and then it's got some other stuff that we don't care as much about. We don't deal with the colate much that has to do with view with indexes and how things are stored internally. If you don't put them, you don't figure it out on their cell. What you do want is the not null. Or this integer not null. And. Finally, that default and this is the fancy thing. That allows us to say this value is going to be the next value. When I create a new web, when I later on created a new thing, that means if I don't specify the city ID. It's going to be the next value in the city city ID sequence rate class the meeting memorizes. That's just it's the way you do that thing. Similarly, I went and looked in here. I'm just I'm telling you, because you're going to look at some of these, and I'm not gonna mean a lot to you. That right there is what controls this. You just look at somebody else who did it. And make it look like that thing. Or in Postgres you can also just call it, give it a type of serial. And it would do the same thing. It stinks and I'm not positive, but I think we may let me just check the. By the way, yeah, OK so. In our artist. Yes, we have customers. If there's one thing that drives me absolutely, start creating that by the way, about our curriculum team. I should never say bad things. Rather, curriculum team because they make my life so much better in almost every way. Is that we they go out and they do exactly what they tell you not to do. OK, give me one to see what they do wrong with these table names. Table name the plural. Yes, they're plural, and they're supposed to be singular. Now I can't even tell you why they should be singular. Alright, I know that's the norm, and I know that if we're going to tell you it's the norm that we should go do that thing that we tell you is the norm. So do as I say and not as I do make them singular. OK, this is the customer table. It's not the customers. Table. This totally violates what I tell you in Java, which is that a collection should be plural, but it does go along with what I say in Java, where I say you should heavy customer class, not a customer's class. Right? You just gotta learn the rules and clearly our curriculum team didn't learn the rules on SNL, GO and chive them with that, but just so you know, anyway, there create table the customer ID. Right, it is marked as serial. Serial is the same thing as what you're going to see in PG admin as all this like everything here the whole thing. It will tell them it's not null. It will tell them the default is the next value. It will tell them it's an integer. Is cereal is just a special type that is in Postgres? And I think they have another name for in SQL Server. They've got something similar, but it's not called cereal, but cereal just says this number is going to start from. I don't know 01 at least start from 1. And every time it creates something, it's gonna do. It's gonna just do that thing one more or one more after the last one. That is where when I say it's next Val. Next cell if if I actually explicitly set it. The next value is gonna be the one after that. So if I had three customers, there's one and two and three. And then I add one as 1000. The next one is going to be 1001. Part of the reason I mentioned this to you is 'cause. What if you guys have ever gotten checksum with anybody gets checks anymore, but when you order checks? They often ask you what do you want the first number to? And this is sort of the equivalent of that if you want to make sure that your checks don't start at one, because then people will think that you've just opened the account and they won't accept them or something. And you started, you know. 12:45 and they, they think you been writing checks for years? And since everyone does this, I think I don't know. The whole thing seems silly to me. In any case, if you want your first customer, if you don't, if you want every customer to be at least 4 digits long. Which I when I'm running a small business, I have to tell you that one of the things I made sure was that my customer numbers and my invoice numbers and things like that all start at somewhere in the four digits because it makes it sound like, oh, I'm not going business and have been for years and the reality is that over 25 years of being in business. You know? I still only had a few thousand customers like someone big, but they're only a few 1000 customers so I never would have gotten into the four digits if I hadn't like no done something like this. So customer ID is cereal if you want it to start in particular number, you just the first one you insert should have that number and you should specify. Alright anyway, name bear care 64 says I've got a string like thing that cannot be more than 64 characters. Alright fine alright? Let's take a look at what they did and how close it matches up to what we've described. First of all, they dropped the tables and commented it out. Why but OK? They are customer kind of like ours. They are artists and the artists. They have a serial and a first name and last name. The only value. To having a first name and last name. Is if you have different places where you want to display their name as a full name or as sorted by last name. Honestly, if you don't have that user full name because why do you care what you you never gonna go out there and say I wanna know OK only if you're going through tech elevator will you ever care that you get all the Georges from your from your. Movie, database or whatever. OK, that is purely an exercise in in, you know we we are trying to make you do your stuff but the reality is, if all you care about is their name, do you have any reason for the first name? Last name? It is very common to do. I object a little bit because it is such a. Social construct of the United States that it does not match up to Asian cultures doesn't match up to Hispanic, you know, to Latino cultures. I just II object to it by theory, but the reality is, you know if you could have just full menu, you wouldn't have that problem. But it does allow you to sort by last name. We attack elevator sort by the first name. She don't need that. Object artist that is artist and. Sorry, we had our customer and it has a serial. It has a name. It has an address and a phone number. OK so how does that match up to what we kinda thought our customer customer ID customer name, phone number at except for the fact that they did at first name last name I think. It works now. So interesting for the customer, they did go ahead and use the name as a one name somehow for the artists they do really care. Maybe we do all care about all the the Georges. They're artists. How does that match up to our list? Great artists artist ID and artist name. So except for the first name, last name, thing, they did exactly the same thing. The constraint they have is a primary key. When it does this like this, that just tells you which of these things make up the primary key. Now remember, a primary key may be a single column or multiple columns. So this is telling you how you make up that primary key. This similarly said the customer ID is the primary key. Then they came up with art. The art code ID. So that's our ID or title. Artist ID and he said the primary key is the art code ID right? Which is this? That matches up to what we did in ours, Nope, so not there. OK matches that we did here. They do not have sorry bouncing all around all the place. OK they don't have price which is an interesting choice and I don't know whether this art gallery doesn't particularly care about how you do your your your pricing. Maybe they just say you know, whatever the buyer will offer is if they do. But they don't have a current price here. I think it's a valid choice on your guys part. Our part to say that there could be a current price. They have the primary key as the article ID and the constraint for the foreign key. Is the. Artist ID which references the artist table. And their artist ID. So that is the thing that says. That joins, you know, in our in our entity table. Entity relationship table table is the thing that says this. She matches to this key. Can you please tell me what is the value that I get out of doing this? Like I grew friendly artist. Anyway, what? What is the foreign key give you? To actually give me two things, it gives you some performance just 'cause they they predetermination things. But it gives you an enforced relationship, which means that because I have it here, I cannot specify an artist ID that doesn't exist and you cannot delete the artist. If I have art out there by that artist, so it enforces this relationship. In such a way that for referential integrity I must have. Both of the things like if I reference a I may not have any art by that artist. That's fine. But every piece of art I have has a listed artist. And I cannot remove the artist until I removed all their works of art. Turns out I can't even do it then because I have that part may also have been sold that I have to keep track of that. So I mean it. It enforces that sort of relationship. And I have a quick question about that set up. It's a it's unclear to me if in this implementation can you have more than one. Or can you have more than one artist make a piece of art? Or is this locked in like one piece of art is made by one person, one is locked in that one named artist. Very good point because if. They did not have that requirement. If it could be that you know two different artists worked on a particular piece of art, which is perfectly valid, but if you had that. So what would you mean to do here? 'cause you would have them a one to or a many to many relationship. So you would need something else. What would you need here in order to do that? And associative table you would need an associative table that would allow that to resolve that many to many relationship. He chose not to make it quite that complicated. I don't know there's a valid choice, you know, but it was. It was their choice. So right now this is a one to many, and I mean one artist for any piece of art. But that artist may have made multiple pieces of art. Everybody understand that it's the. It's a when you have. Multiple artists may have made a piece of art that you have to have an associated table. You can't somehow list them all in here. Alright, and then they have customer purchases, which seems to be equivalent to our transaction table. Alright. It's got the customer ID, which takes you back to here. It's got the art code ID which takes you to the art. The purchase date and the price. So. The primary key. Is that? Customer ID the art code ID and the purchase date, combined together as a single key. To my mind and very questionable choice, but it's perfectly valid as a theoretical choice. What they're saying is I cannot have sold a piece of art. To more than a customer like, I cannot have sold it to multiple people at the same time. I cannot have sold. You know, this transaction is is the three of those combined. The way you do that is shared. The primary key and you do list it like that, just common delimited. All of those are the primary key. Together, not independent. When we talk about third normal normal form just so you understand that thing that they care about. It is that. You cannot have any values in this table. That are dependent on only parts of the primary heat. They must be related to the entire thing. So if you had something that was just related to the customer and the art. But there was independent of the purchase date. It would violate third normal form. That's a kind of complicated thing to describe, and then what I have found is that people actually make tables that make sense more quickly than they. Yeah, they're normal form, so I don't. I don't emphasize it too much because realistically, most of you won't have to like. Just make tables that make sense and and mostly that will. You will be OK on that, but the idea is, yeah, if you. Give me an example 'cause I'm not sure that's really super clear. If I had a customer address in here. Right? And that is related only to the customer. It's not related to the art or the purchase date. Right so or if I had the artist name that is really only to the artist code. There's no more form says nothing can be related. None of the other elements in the table can be related to anything except the key and the whole key. And nothing but Becky. That is really the way they keep saying. Alright, so this we got that that's the primary key. The foreign key is the customer ID goes to the customer table. And the other one is the art code goes to the art code. So those are our relationships. If I were to go. Hi there. If I were to go and. Create this particular thing. Let's say. Sorry create I don't start calorie. I'm trying to remember whether I gave you guys this DDL. I mean not have, in which case I will drop it. Yeah. Watch me do all the magic. Anyone who really wants to follow along. Give this themselves. Create your art gallery. Double click on it. Train and then. Query tool. Going to your lecture. Happy where mine is. And run it and then all of a sudden here your art gallery will be here and you can sort of take a look at what it looks like. So if I go down to my schemas. I got my tables. That I have my art. And my artists and my customer purchased my costumes. I'm gonna go we're gonna go ahead and take a break now we'll come back at 10:25. and but if any of you wanted to do this it is now available for you to pull down try that out

, let's say that I'm going to do something kind of complicated. And I need to change around. I need to swap something or change something in this here that is then going to violate that the integrity of the the park state table. I can delete the relationships in the associated table then. Alter this, then go back and recreate them. But another thing that we can do is we can alter the constraint just temporarily. So I could say alter table. And. Shrek that much. Syntax ratso. I made put in here. I might say let's alter. Table. Perk state. Right. Now I can safely do this because I know that. I can just recreate the whole thing at the end. Don't don't normally do this without, you know, a whole lot more care, I might say. Begin transaction. Action. Because I wanna make sure that at least. Does what it's supposed to do is. I'm gonna alter straight and I'm going to drop straight. I got the. Terminology right and I'm the constraint I'm going to. Draw both of these constraints. Perk. Hurt. Park State Park. Strange. FK. Stand. yeah. One sack. I go out and Google and say what the heck was it up? Drop, constraint, alternative, whatever. Or I think I just sent you one at a time. It's probably the problem. OK. So that committed and if I go back and I look at my constraints again, I'll probably have to. Refresh the thing. Yeah. I did that successfully. OK, so now there's only a primary key. Now I could remove the the state. And. Or I could do my complicated manipulation. Right, so sometimes you need to drop. You need to alter the constraints temporarily while you're trying to do certain things that may. If you ever remember if you remember in the homework assignments when you had an array and you were trying to swap an element from from, you know one place in the rate to another and you need a temporary variable to store them in between. There are no temporary variables in in in a database like this. So in order to do that kind of manipulation, you may have to go through a series of steps which at some point leave you vulnerable to preferential integrity. One of the approaches that you can take, and this happens more often than you would actually this, is the sort of thing you wind up having to do is you might have to drop the constraint temporarily while you go through your process and add the constraint back. If you're going to do that, and sometimes you even do it in the original DDL, sometimes it is too complicated to actually set all your constraints and Add all your data. From the beginning. So what you do is you wind up in Add all your tables but without all the foreign keys. And then you can add your data in any order without having to worry as much about the like the exact orders. What is the order that you're going to add? What is the order you're going to either add or delete data. Right, you can't add. Per state data until you've added the parks and the states. And at the end you can't delete parks, parks or states without having deleted the the associated tables. Well, one bully got five cables. That's one thing, and we've got 50 tables that have complex things that go through. Our friends think it can be very hard. To unroll the data in the right way, and one of the ways to unroll the data is to try is to to alter the table temporarily, and if you're going to do it, you should always do it inside a transaction and then add the alter the the. Let me go back and look at the original. It's gonna take me a moment. 'cause. I'm gonna have to find it again. Add. Stop doing that. Reason. Fine. Oh my so many things. How much murder trial? Sorry, it's gonna take me a moment to find this thing. If I go back into module one, I'm going to issue the databases I go into. Watch your final note going to lecture. Well, look at end visualizer, so that's what's start up. So I have an association between dot PSQL and visualizer. And it actually does come up and show you the thing. And I haven't created how to undo it yet. You know, I could, but I haven't figured it out. So come on, find out how to. Somewhere, you know, show me how to. So my cable perk has this. Fact they do it every. Adding a whole lot of stuff. See in the in the United States table. They went ahead and added all the data in. And then they added the foreign keys in last. Right. If you get into the lower a whole lot of data rather than have it constantly checking constantly, you know doing whatever they just put the four end key relationships in as altered things at the end. So I'm going to take steel, the ones that they did right here. Try hello. And then I'm going to go back into my. And here. Add them in here. This might be the kind of case where what you're going to do is you're going to alter them to drop the constraints. Do my stuff do my complex things. OK. And then I'm going to alter them and add them back in again, all within the same transaction. The live database will never be without the constraints, so somebody else can't go in and make your changes and do whatever else. But sometimes it is just too hard and I I say this is part of database design as opposed to database management because sometimes if you're trying to load something up. It's just too complicated to try to figure out the relationships to make sure that you're always doing everything at the right time. And so the easy way and I go look at the United States table, but that's include example of that. The easy way is to add your primary keys as part of your original DDL. Import all the data, do all the stuff, get everything ready and then try to alter table. If you screwed up, it'll simply fail, but you won't have touched any other database, so you know you'll be in good shape. Sorry, too many things open. Yes, the other one quick visualizer. Quick show of hands how many people? Even attempted the the tutorial. A few people. OK, so let's take a look at the tutorial a little bit and justice and play through it because you know it is a good exercise in doing this same kind of logic. Get to where I need to be. Can you hear me? Yes, I can. I don't see anything on my screen at all. You don't see anything on your screen. Like you don't just see. Everything is black. Let I'm come out and go back in, OK? Good luck. I wish things it hasn't here. We're not going to go through the whole. The whole tutorial. But just to be clear on on trying to see our relationships and how we get go through this process and how we create something in the end, right, you will often be given like we were with the other one, you will be given sometimes a series of different pieces of information. Like these two. OK, we've got a health history report. We've got our invoice. Right. You will be given a series of things which will often be given is a series of instructions that say we need to be able to do this or that. So let me see if I got my readme in right place. This steps you through the entire thing. The whole way through but. Well, I want to talk about it's just how do you start because you're going to be given this kind of a challenge and how do you start? The first thing like we did before is to say water. Just pieces of information. So when you sit here and you look at this and you say, what are the pieces of information here, you can list every single one of them. Don't try to make your tables first. Don't try to make your entities first. Like we did in their exercise, the beginning class. Just put down all the pieces of information. Alright, and make a list of just all the things. If you want, you can kind of group them as you know you'll get better and better at grouping. You say? Well, OK, BBS, things are maybe related to each other. You can put the close to each other. But let yourself not. Get caught up in in the major enemies at first, and the reason I say that is because. Database design is all about context. The same pieces of information may or may not be major entities in your. For your particular challenge, for your particular organization. So far we have. We have tended to look as if you wanted all the pieces you want to say. I need to know. He is struggling today. You want to know all the pieces of information you want to say. OK, I like we have said that we can say, OK, I dog, what breed is it? OK, we want to say what is its name? We want to say, you know, how old is it? If you start creating your design and add absolutely everything, you may add extraneous information. And extraneous information is a problem, both because you're waiting, weighing down your database, but also because somebody has to add it. And so you have to like it is possible to put in values and say, well, we'll allow them to be null. And then you know, that sort of solves our problem because. We don't have to have those, but you still have. Columns that are that may not be related to the tasks at hand. So you need to look at the sort of things that they show you like this. Alright. And you need to like if it's actually showing up in a report, you do know you need it, but if you look at this, for instance, OK with this this pet thing. Do you need to know for the purposes of the person who is doing these things? Giving you these requests? Do I need to know the breed? Do I need to know the the name of the of the pet? Maybe? Maybe not. I probably need to do because right here is going to the penny and because when I'm doing a health history report at a vet. The same person may have multiple docs. So I better have a need for it. I can't just have it tight. Now, it may be that the vet needs to keep track of. Preteens. Because different breeds have different kinds of problems. But it may not be that you in the context of the database that you were creating need that information, and for that we're going to talk a little bit about domains. Right. The idea of a domain is a set of information that is related to each other for a particular. Um herbis. So. if end like you can have overlapping domains. If we went back to our United States database. Alright. And we say. Are United States states? It may be that this is one domain. It's also, it could be argued that if I'm dealing with state parks and and that sort of thing, I'm not dealing with cities at all. And maybe they should be in different domain. OK. And. Over there, if you've got an insurance company. Their claims process. May be unrelated to their policies process. Or their, you know, customer relationship process may be on now. They're they're not that there aren't overlapping pieces of them. Part of what you were defining when you were doing a database. If you were saying what is the domain that I am working with right now. And so when you look back at this at this health report and whatever else. For the the person who is doing this and I again. The. Let's say it's billing department. Who is doing these two things? They need to be able to generate stuff to hand to the customer. But they may have a different domain, a different set of information that they need. Then the the vet themselves. Now in that office, maybe you keep it all together. Maybe you don't. I mean, just like, but but the larger organization, more amount of data, you have, the more you want to make sure that the things you're keeping are not all the things you might eat. But only the things you do need. And when you need to connect up to a different domain, you will, but it becomes a transaction between domains as opposed to saying, let's like if we're never going to query anything from a whole set of data because it's not really related to our tasks. Then we want to to keep them in separate domains. In a SQL database, that domain is usually going to be. I mean in in SQL. Postgres, I'm sorry, I'm using all the wrong terms in Postgres. The domain is usually that database, right? So United States in this case where the pet DB is the database that is your domain. I may as a programmer. Want to I may run both an art gallery and at a pet database and I may have customers in common and I somehow want to keep track of them, but I do not in any way want to to make them the same database. So. Let's look for a minute at our. That database. Customer like columns whatever. What we're going to get into for the rest of this week? Is we're going to get into, how do we take this data that we're used to? In a relational database that we've added in whatever else, how are we going to then map it? Two Java objects to a Java data store. Remember, Java is not going to. It is just about the specific transaction. So like I don't need because I have my database now I don't need all the data when you did your vending machines. You need to store all the data at one time. But but I have a database for that now. I don't need all that stuff. What I need is what is the data that I need right now for this transaction. So you've got both the domain in the in the database sense. And then in Java you got. What is the sort of? I may call us sub domain or what is the domain of the data that we need for our purposes right now. Similarly, I may add in my Java I may need to do things between one database and another database or between databases that are disconnected in other ways so. In terms of our client server module. Let's say we have a client server module. Java is going to start being a client. That interacts with the database server management system, Postgres, which is now the server. And I'm gonna make queries. I'm gonna add things to him and go back and forth and and and. Through my interaction that way. OK. That said. The rest of this week is going to be kind of complicated. And. This is all we really need to talk about with. Image design there's a lot that can be said about database time, but. Not much of it is going to be useful for where we're going. So what I want you to do is I want you to go out and I want you to spend some of the day we're gonna finish up early. And I want you to spend some of that extra time you have. Go back and look at the object model you had in your. In your vending machine. And talk about, think about where you could have stored that data in a database. And from a conceptual part not worrying about how you actually interact. What would you be able to store in the database and therefore not have that code anymore in your? In your Java. Right. Think about that conception later in the week. We're going to actually try to implement some of that, going to try to actually do some of that. But for right now, let's just I want you to look at it. Partly, I want you to look at your Java code so you remember what Java is. Look at your Java code. Look at your classes. Think about how they would relate to an actual if you created it. If you want it, you know it's not that hard to do to put a little bit of your your data. Into. You know I Postgres database. But try creating for yourself a a model in your head between between those two, and then we'll talk about where where it falls short and where it needs to get added to, OK. For that I will give you a power back of your day and. Go forth and enjoy. pregnant