but just so you know the point of the the query for object is to make your life easier not to make it more difficult right it is it is a way of getting a single value whereas the query for rosette will get a whole Let's go look at some of our. OK, if I look at my. This is today's lecture, but it'll look really familiar because it's just yesterday's lecture was more stuff in it. OK, so if I look at my. The city info or something like this, right? This this infamous map rhoticity. All that sense is I need to be able to do this kind of thing. Alright, if I wanted to in where I do the? i'm sorry i'm trying to look for where you it I'm trying to look for where you create create. If I wanted to do this with a query for Rose. Then I would just call query for Rosette here, skip that long doc class and get back results, and then when I use the results I wouldn't call macro, I would just call this one line that says new ID equals row set dot get long of city ID. 'cause, that's the only thing coming back. So they ask for it as a series of rows. It will give me a series of rows if I ask for an object, it thinks I'm just getting one thing and it will pull that thing out for me. Alright, so it's just a shortcut. That said, I'm going to put these in in here, and if there's anything else I was going to put down the thing about the dates. Few people got stuck on the the dates right, and as you've seen, if we want to do with the date. Wrong place. There we go. OK. OK Does this one have it? Insert even have today so I I don't know. I I'll I'll get it from from the other one. But the date thing that is that Rick about the date thing gives anybody got stuck on. Is that? If you have a null date which is allowed in some of these examples, if you have a null date, you can't just do the get date and the two. Local time run together two local day run together because the Getdate is going to return in null value. So what you have to do is. Find my there it is OK. The simple answer and I will copy a better version of it is that you said you go date date equals row set. That getdate. You're seeing. Let's say my fromdate. And then you say if date not equal to null. Right now I can do my whatever my thing is I can do my. Project maybe project? Data set date. Page. Start or dead dot set from daytime. I'm doing this from memory so I'm. Trying to remember wishing data. To local. Date. So in other words, I'm doing the same thing as chaining them together, except that I'm taking into account the fact that if there's a null date, I better not be setting that date. OK, that's. Alright, in any case. For people who got stuck and like this includes, you know, angels saying you got stuck for a long time or whatever your problem you said was that you were missing something in the macro. The macro is in many ways the most macro to whatever your object is. It's the most important thing to get right from the beginning. The thing that you have to pay attention to and that pro. Like I would actually probably write it as the first thing that I wrote when I was doing a DA. And so when you do this, you are going to have you want to know where is my model. And three things you want to sync up here. You want these right here to be all the all the columns that are in your database tables. And they should be in the same order so that you can keep track of it. Make sure you got all the things that would show, so if you do a select star from you know city, you should see these columns. The name should be exactly the same, except that you turn into Java name so they don't have the snake case. In your. Ah. City in your map. In your map rhoticity yeah. You need to make sure that you also have exactly the same. Once in the same order, there's actually no required to have the same order. It's this is for your sanity, not not because it's a requirement of that you're going to do each one of these things in the same order. Make sure that the same type and use the database that the Postgres. Column name as the thing you get. You're setting it into the value in your mouth. I would recommend that the very first thing you do as you go and create this whole thing is create that map row, make sure your city and your map rhoticity and your PG admin city columns all line up. You will save yourself tons of time. alright you guys have mostly understand they'll be been working from the top down you start looking at the beginning when you just start going down and the tests aren't going to very well that Why that is true, that's an unavoidable fact given what we're doing here, OK? So. He. Follow that pattern, you will save a lot of time. What else did people? Where did people get stuck? So, you know, stuck on a selected the. Cities without or whatever what projects? Members without projects or something like that, alright. Where is that? Where else do people get stuck just so I can help? When I was working through with other people, I feel like in general. The process was kind of lost on people, so like setting up the map row and then what it does. And then making sure that we set up the proper string state statement that we pass into on my SQL first. What the JDBC template does 'cause at this point like even for me it's a magic black box that I know interacts with with some SQL. Yep. So just like a little overview on the process, for me at least like I have a general grasp about how it works now. But. I think what was losing me was after we did the JDBC template part, the ifs in the Wilds and stuff I was getting lost on. What do I do there right? The rosette thing is a is a. Alright, if you think about a cursor on the screen, it is where you are on the screen. OK, thank you. Move it around. I can show you where I you know where my cursor is right in when you're dealing with a row with a whole set of roads. When we were to our cursor, we're talking about which row we're currently dealing with. So when you go out and you call your query and imagine you're in, you know I'll switch over. I'm gonna be PGA admin for a minute here. Hacker by know what it is that I'm doing, let's go to my art gallery care. So let's go to my cities. Do I have my cities here? So many things. OK, now you states. So we're going to look at here, I'm going to. Network query and I say. Select. Star from city, OK. Query. K. Alright, I get back a results. When you look at the way the results set is implemented in Java. It has access to all of this, but it starts out. Above the table it starts out with nothing. Is not on any line on any road. Alright. When you call. Next. It tries to move to the next row. And it makes that the thing I'm on a curse. The cursors on that road, that is the road that I am. The only row I have available at this time. And if you think of it like a four loop. Particulare for each. I'm only dealing with that one row. That's all I have. So when I do that I am going to have whatever the results were. And they are identifiable by their column names. Error, also identifiable and as I think I mentioned you guys by index. You could say get the value at zero at the value at one, get the value at two and so forth. It's not very useful because then you have to also check for the data type and what you know the name is and things like that. The point being that when you in terms of the ifs versus Wiles. When I look at my job code. And I go this is I'm going to call macro to city. So I'm going to do something whereas a. Get cities, get cities by state. I don't care what the query is, the query is going to do something and it's going to return potentially multiple cities, right? So. When I call my query for Rosa, I am executing this SQL statement. Well, I am first wrapping in any of these things to replace the question marks. I then come up with a full SQL statement. And then I am calling it here. This is going to come back if I copy it over into PGF and and reproduce it. It's going to come back with some number of cities or no cities perhaps. And in this case, I think every state has a city, but let's assume that you could come back with none. When I do while. Every time I do a results dot next, including the first time it goes to the next row. Because it starts above the Rose, the very first time it's gonna go to the first row. So when I do that while it's gonna do results dot next and it is going to go from the from here above the roads to the first row. When I am done with that row and I do call next again, which happens in a while loop because remember you you you keep checking the condition. It's gonna Net results that next is going to return false. When it gets past all of them, try to get past all of them. OK. So in other words, you know. This is the exact exact logic that I'm going to do here abbreviation. Peoples TX. So if I call my Myjava statement that does this. OK, I'm gonna get however many things with 41 rose. My while loop every time I do that I check my wild dot next. Every time I check where is it? Alright, in my wild results.net this actually runs the next command method which shifts me down a row. And if there is a row there, it returns true and if there's not a row there. It returns false, so if there is another road to deal with, then I go inside my wallet. My while loop is only dealing with one row at a time. Whatever I do with it in this case, I'm doing the map rhoticity. I'm just copying the entire row into a Java object. Monkeys by Peach boom boom boom boom boom, boom. I get it. I created my Java object and then I do whatever the thing I do. What I'm doing in this case is I'm adding it to a list of cities, but Mac wrote it to city, creates a new object, copies all the pieces in, and then returns it, and so then my add can add it to my list so that whole purpose of that while loop is to process multiple roads. So when would I use it if? The two times I would use and if I would use and if. If there's either, I expect one row. Or if I need to know whether there are any rooms. Right. If I just want to know, are there any results I can say if results dot next and then do go about my merry way and it tells me whether there were any results? I don't it happens there maybe 10 of them. I don't care. I just know there were any. So the two kinds usif are when you want to get the first row 'cause you only. Expect one. we do

check if results dot next. Does this mean that there is only one? I have no idea. It may be multiple ones. I only assume one, so I'm checking if and doing the one time and I never bother to check for anymore. Right. That's the only difference between the Earth and the wild. It's whether you expect or care about more than one. You use a while. And if you don't, then you're going to use an S. Regardless of which one I do, if it fails if there are no roads at all, I'm gonna drop right past it. Whether it's a while or and if for whatever the heck it is, I'm going to drop down to here. And then get city. That means I'm going to get in all cities because I never will have added a city. And get cities by state. It will return the list 'cause the list was already created, but it will be an empty list. Got it. Not answer your question, kita. Yes. OK. Anything else on that? Something just like, wanna make sure I understood you when you use the while, it's if you expect more than one. Result is that what you said? Use wild right when you expect when you need to process more than well more than one a while is from zero to NNE number NOK. So it is when you expect that there could be multiple rows. OK alright, I just want to make sure. I got that. Thank you. Any other? Hang ups that people have on the homework. Or on the radios work so that we can go on to the testing 'cause I wanna make sure you understand those things. Could you go over left joining one more time? That. Bendy fur left join to you too. Unless you do. You need it for the homework. In this case, I don't. I don't remember. There's one problem that you used before. Yeah, you do? Yeah. It's just a cover for nulls because you need to check for nulls. So if you don't left join it eats up the slots that would be null. I think it's on employees for the homework. OK. I'm gonna hold off on that because I think it's gonna get us too far afield. Pretty much a left join. Just a big picture. Wise. A left join says that I'm. I'm taking one table. Alright and all the entries in that table are going to be represented. If I have a left join. Even if they don't have something in the on the other table that I'm joined. If I have. If you don't have a left join, I'm only gonna have the entries from the first table that actually had something in the second table. A left joint says no. I want all the entries in that that left table, even if they don't have any matching edge. And a right joint is exactly the opposite, which is to say I want all this stuff in the right table. Even if there was nothing in the first in the left table that had it. In either case, it's gonna put nulls in there. And one of the things to always watch out for is that means you need to. Be able to have nullable values. If you say that it can't be a null. That it won't be able to generate that result set because you can't generate a result set that has no values where you've said they can't be held values. Anything else on that? And what I would say is go back to the also go back to the sequel when I'm joins and look at the class notes on that day 'cause it has the, you know, the images of the two and it helps a little bit. Add one more. Go ahead. Sorry. No, it's fine. I have. English alright, I just had a question on I like because from my understanding in the reading I thought that it would automatically. Rap, I always forget the right term for the percent signs, but from what I was reading, I thought that it would automatically wrapping presents and in single quotes, but that wasn't the case when I was trying to do the problem that required. I like and I'm just trying to figure out what the point of it is if we have to manually wrapping presents anyway. When you say rap the percents, what do you mean 'cause? I'm just not like, sure. Like, I think that percent? Percent is that what you mean, yeah. Where we have to try that and it wasn't working. So we had to manually concoct a string that was surrounded by percents and we left. I like in there, but I was trying to figure out what it was actually doing the entire time. So I'm just confused on like terminology is where you're getting I like isn't doing anything. I like to those doodley squat. I like is part of SQL like it's no part of the wrapping or not wrapping or anything else I like. It's just a command that says whatever I have on my right. It's gonna be a string. And I'm going to do a compare on on that value for whatever is on the left. OK, so what's the difference between I like and like then is there like is a case insensitive search versus a case sensitive search like OK, so only deals with the case. OK, that makes sense actually. Exactly like like, but it's just a case insensitive so that I don't have to worry that I'm matching the kings. OK, I thought it was doing way more than it was supposed to. Then that makes a lot of sense. OK, that clears that up. Anything else? Alright, well I'm sorry, I did have one quick question. So in this more kind of a job question too. So in in the homework there was this one tricky column or at least it was I thought it was tricky and there was it was a long value that was. It was a long value that was of long object type so long as a primitive in Java and then capital L long is the object wrapper and it was one of the homework questions had a column where it had a nullable column that was of type, you know whatever with some integer type, right? So when I guess when I the question I have is the default value for a null long will map to zero in Java, so it'll it'll it'll it'll do that autoboxing or whatever for you, do the unboxing and it will unbox the null. Long object into a zero. OK, well, I guess the punchline here is OK, that's what I thought. Do you need to do any null checks for any wrapper primitive types ever in SQL? You know what I mean 'cause it was a nullable column in SQL, not for the well. Not if you treat them as as in as the. You've got to force it to do the unboxed. If you leave it as a. As a rapper. You would need to do the null check. OK, so for instance, when you if you if you treat it as a number then it's going to unbox it first, so it will handle the null. If you don't treat it as a number, you treat it as an object, and so you're going to reference the object. Then you would have a problem. So for instance, and I'll give you an example of that. If what I'm doing is calling one of the. If I can just do it in here, I'll get rid of it again afterwards. Yeah, specifically, the whole. I'm sorry. Yeah, but if I say long. My numb, OK. And equals null. Right. And then I try to do my NUM dot. Compared to. It will crash because it's a null value. But if I'm treating it as a number, if I say if you know my NUM my NUM is greater than two, it won't crash because it will unbox the null and say I'm treating it as a 0. It'll fail 'cause it's not created too, but it'll you know. That sort of answer your question or not. Or you want to push the issue in the? Well, no, I mean we can maybe test at another time, but I just wasn't sure if I I had I. I didn't know if the tests were testing for the tests may not test for what I was curious about. I guess I didn't know because there was some long value specifically in the in one of these models classes and I think if I forget what it was in the homework, but we'll get back to that one and then the quick question or hopefully quicker is when you're querying for object is that only used? For a when you use aggregate functions 'cause I never I never had saw a time to use that in the homework. Or if you were supposed to, I did it other way. Functions are are good case, but but if I let's say that I did a query and again think of it entirely in terms of the queries 'cause this is not. Let's say that I said. Select. Population. From. City, where city ID equals 20. OK if I run that right there. Population spelled wrong, yeah. You lation. Half the time I want these things to be smarter and half the times I want them to be Dumber. OK, I'm only getting one value back. I could call query for object with this thing right here. And and just and Sam, I want a long and it will give me along. And so if you. Theoretically, like the homework question asked you to use query for object to find. Employee ID's and put that into a list and compare it to some sort of list of all the employees and do something that wouldn't. That was one way to do the homework question. That was one way to do it. Yeah, gotcha. OK, thank you very. Alright. Anything else before we get back to the actual topic of the day? Alright, good. Just so you guys know and there will be more chance for some of this in that we you will have a review day tomorrow, it will not be with me for now. It appears it will be with David from the C sharp clouds. Who is, you know, trying to remember all his his Java while he you know, so that he's ready. And Colt is also going to be there, you know, in in a supporting role. So that the two of them can help you with any of this. Any of these things that you want to learn. Excellent place to ask any other SQL questions 'cause. Fortunately, sequels awfully similar between the SQL for SQL Server and SQL for Postgres are very similar so but but anything you want to talk about. It's always nice to get a different person with different ways to answer the questions and whatever else, so so be prepared and think about things that you want him to talk about. When I post today about the homework, I and the teams and all that kind of crap, I will mention that if you have questions, you should respond to that thread. And then David can see them. So that in case he needs to say, Oh my gosh, I don't remember anything about that, he can go and and brush up on it before he then teaches kids. David knows everything about all this. I I've no question about it. But I would do the same thing if I were handling review for David, which I did like a couple times last cohort. Then I would be brushing up on all my C sharp stuff. So that's. You know the way you do it right. All right, the topic of today is DAO test. Deal. But dear testing is. A type of integration testing. So we need to step back for a minute and talk about what integration testing is. In the. What we did earlier was unit testing, unit testing says I want to look at something like my get cities by state. Or in some methods somewhere in a class. This is not a great example because Randy L, but if I wanna look at one method and I want to that, I can pretty much set up. Run my test and finish. Now when I'm doing that. These are very very simple objects. Excuse me. So there's not much I can test in these, but. That's what you normally have is is something that is not dependent on anything else. It is the IT is a class that I can do a sort of generate, set my values into if I need to sort pair for my test. Run my thing. She doesn't do the thing I expect. That's what a unit test is. Right. The integration testing that we're going to talk about today will look very similar. And the reason is we're still using J unit. We're still using the same testing framework and a lot of the pieces will still be the same. But the difference between them? Integration testing and and unit testing is that integration testing is when you have I might have. Pure down there somewhere. Alright, integration testing says I might have multiple pieces of my system working together. Or I might have outside things that need to get set up and so it is a. It is a. It's a more complicated scenario and it is. It takes more setup. And often close down. In this case they the reason we need integration testing is because we are not only dealing with our own little piece of Java code. We are also dealing with a database out there. And so if you think about what has to happen in these cases, is it, it needs to go out and essentially generate a database. That is configured properly and that is filled with the data we need. And run our test and we need to do it. For every single test. Like when we've been doing stuff so far, we have that United States database out there, we insert new cities row in cities are still there. But what are the? What are the big rules you guys remember about testing big rules are it means to be repeatable? And I need to be able to do it in any order. Alright, so I can't have the fact that Irena tested insert cities. Impact the one or that I deleted cities impacts the later test that expects those cities to to be there or not be there. I have to start from scratch. Right. If you think about the test for the homework you've been doing. Right. They're actually pretty good examples in some ways. Because you know, for these last ones. What you'll see is. Earlier homework has pretty much every single thing that you fixed. You know what actually passed some individual test every every method you worked on. There was a specific test or more than one test for that matter. So it was very easy to work down the row of methods that need to be worked on and have your tests passed one by one as you kind of made progress. Obviously the CIO stuff was nothing like that. Oh my gosh, all kinds of weird things with pants are not passed depending on which things you had set and hadn't done. That's the way integration testing often works. It's a whole bunch of integrated things that have to happen. So for example. If my generic database that I am setting up. If I want to test. A something that that gets a list of cities. Then my map Rhoticity has to be working and my get city probably has to be working and my you know my get cities by state has to be working. They all have to be working at the same time because they're all being used as you start setting up these these tests you have to be able to use multiple ones in order to test the right thing. And I am not likely to want to set up. Lots and lots of different databases for every individual test I want to set up one database and then if I need to test for instance, that what happens when I delete all the the cities from the state. I need to then delete the cities from the state and that's going to. My delete has to be working, so it's like more and more different individual pieces have to be working in order for the test to pass, so integration testing is more complicated. It is slower and I should say slower. That usually means it's, you know not. It's more milliseconds, not just as few milliseconds. I mean it should still be fast where it's gonna come in is if I've got 1000 tests to run and their unit tests. They made the whole thing may run like when you guys run the tests on your homework and there are. 35 different you know, tests. Right. Slowest thing in that process. Is 4. intelligence to make the little checks and show them up on your screen. If you ran it without bad, the whole thing would be done in under a second. So that's the only thing which makes it take longer than that. Once it's, it's kind of gotten initialized and starts running his tests. The only thing which takes longer than that is is the printing it out. It's making the messages show up OK. Integration testing is going to be slower than that, so if you're running a whole lot of tests, you can actually have. Delays you have to, you know, you don't have to worry about performance, but I like for this course. But I'm just saying that's a thing about integration test. It puts more demands on the system. It also means that I've got a database out there that may be creating and deleting, creating and deleting and creating, deleting. You know things well. I mean, I I'm sorry. I've got a Postgres out there that may be doing that all the time. It may be creating these databases and deleting them and doing all kinds of things. And then if I'm using that Postgres for something else, it may slow it down. So those are just things to know about integration tests. They do not matter much to you for this purpose, but you should know that in general. So let us take a look. At what all this how you actually do this? I'm gonna show you one part of it, and then as as instructors have been chattering about this this morning, all right, this is the ignore. The man behind the curtain. Moment. That I'm going to show you that it's there. You're gonna know that it's there. You're gonna trust this there and then you're gonna ignore it. Right. That is, there is this thing, OK? Just like we had before, we have our City Park in state, we have the and with the whole DL model, we've got the plain ordinary Java objects pose as they call them, which are the the object for each entity. We have the interfaces which say which methods are going to be available for that thing, and then we have the city go and so forth, which are the actual they implement that interface and do the actual sort of work on the query. You know for roseton all kind of stuff. We are pretty much going to have a set of tests for each one of those. You know that that's the ordinary way we do it. And then there are going to be a couple of extra things in here which you don't worry about a whole lot. There's testing database config. Right, which does a whole lot of OK. What is it that I am? What is my database name? This has got my my username, password whatever. It's got all the stuff and it sets it up and you basically don't need to worry about it. OK? It will be there when we give you an assignment to create something. You want to create your own test. You're basically going to do that. If you got to the point where you had a side project and needed to be doing DAO testing and well, congratulations 'cause most people just ignore this part when they like they may do the detail but they they skip the DNA testing. It's way too much you know? But but if you do, all you really do is copy in this testing database config. And then go look and change these. You know the the values of what you're test databases. And the other thing we have in this world is more common for. In general, it is common for integration testing. Is your going to have the base DAO class? And the pace delete VL class is going to do the it's going to do whatever the connection type stuff is to the database. It's got to do the connection is going to do the cleanup. Alright. It's got more of these really fun annotations, and if you are super into that sort of thing you can look at them, but we're going to ignore this is the man behind the curtain we don't share. It's got autowired. Run with and we will talk about a few of these at some later point, but you don't, you don't need to know, you just know I'm gonna have to have this if I'm doing adieo testing. Copy This from this lecture. Paste it into yours and then you don't have to worry about it. You don't care why it works, it just works. What you care about are going to be the JDBC city VA oh test park, go test and state. What you will see when you do one of these is that it is going to extend basically air test. It's gonna have some set up stuff. Right at the top. This is going to be objects that you can deal with. Right. And it will have a couple other variables which we really stuff to make things like this work. Now, what would you do if you were dealing with a DEO test of something else and you were asked to set it up? It's like we've done with the homework for the set up before. Anywhere where it talks about a city, you probably are going to be doing whatever project you're thinking is. If your thing is a pet shop. Then it's. You're gonna call it, you know, private static, final pet shop. Pet shop one. You know, you're just gonna map out the variables this way. You're gonna just use it here and and and do not worry about the rest of that. Fine. Here I am going to do a setup which will again look fairly similar. So it's really the test that get down into the actual test when you when you were doing DNA or testing. It is best not to get hung up on those details until you're far more advanced. You just say I'm going to copy one that exists, or if we already set it up for you, which we did for, you know, park whatever, then I mean. Copy what exists and it change things to refer to my current object. The hard part. Easy part whichever way you want to look at it, I think it's easy. I love this stuff, but is what are the tests you need to run. And that I wanna step back again from all the testing that say. Part of what we're trying to do at tech elevator in putting you through this whole course. Is helping you to think like developers. And developers have have to be able to do certain kinds of things. They need to think in certain kinds of ways to make stuff work. How one of the? I have a son who, you know, has anxiety issues or whatever. Actually, everyone in my family has anxiety issues, but I have a son who has anxiety issues and I talked to him once about the fact that when he gets he doesn't travel by plane much because when he travels by plane. He gets obsessed with all the things that might go wrong. And in some ways, that makes him a good candidate for being a programmer, because he can think of all the things that might go wrong and then try to anticipate them. Make tests. When I get on a plane. Alright, I'm not sure that this is a healthy thing to admit to. I think of all the things that I can make go wrong and I I try to figure out how I could take out the the, the, the pilot or how I could manage to get open that door and it's like do I want to do any of those things? No. Have I ever taken down a plane? No. But. That is what my mind does again as a developer I try to think about what what could I make go wrong. So for testing you have to think like me on a plane. You think? What can I make go wrong? Right. That's heart of your job. You also have to think, and this is because I watched you guys with unit testing. This is the less obvious thing and what I do not do on a plane is figure out how things can go right. Like I should probably do that. I'm gonna try that tomorrow. I'm gonna try thinking about what I should. I don't have to do anything. I just sit back and read my book. If you know, if my business going right. So I don't usually care about it. But as a developer, you do have to think about making things go right? So tests should not just be. Give you another plan. Example. Just 'cause. We're on that and I'm I'm traveling tomorrow. OK, so. If you were to go out, you want someone who's going to test your plane, right? You want them to test what is going to happen when it is facing extremely high winds. And you want them to test what is going to happen when the there's ice on the wings. And what happens when one engine fails? So whatever all the things that could go wrong. You also want to test what that plane does when it's just in Los Angeles. In a bright, sunny day, day after day and and just flies, OK, because that's what is actually going to do most of the time. And if you test all the disaster scenarios and don't test the the like, we're just doing our business and we're happy. Then maybe you miss something like it overheats because you've only tested in extreme weather conditions otherwise and it just sits there in the sun every day and it, you know, everything starts breaking down. You may not have ignored that because it's just normal to you. So when you think about tests, you are testing what should happen normally. And what should happen when conditions are abnormal? So let's take a look at our. Little early, but let's go ahead and do it. Let's let's take a break until 10:05 and then we'll come back and we will start looking through the individuals tests and how we make them work right.

Doing this stuff right, it's gonna drop the table. It's going to create all these different tables with the right constraints and all the stuff. And then at the very bottom, it's gonna insert a bunch of stuff into it. Right. I say this because I want you to look at these briefly and you're going to see you're going to say. 3130 two 6364 all this stuff is going to get created. If I go back into my. Java now. This. If I go back into my Java. This is going to get loaded and run before every cast. So when I look back in my bestie, oh, it's not there. It's. But I look at my JDBC city. CAR. Before now we have it before. In essence, that first thing says the JDBC city. Their data source is going to set up the database spot. It's gonna be already and what you might have noticed in there was you had a begin transaction. You never had to commit. So we are dealing with that temporary data in database in memory. Until we do a commit transaction, it never really gets saved. And then what happens in our basic peeo is the after which is shared by all of them. Does I get connection dot rollback which just like in SQL just says well just give it alright. Forget forget everything I just did throw it away. So this is the general way that it sets up the database so you know it's there and then what you will see inside the the JDBC city Diego test. I know. My data alright, I'm not figuring it out, I know it. So I'm not going to go out and retrieve it or anything else. What I'm going to do is from a purpose of my test, I know what is in the database. And what I'm going to do is create some constants which are the cities. These three of the four cities that are out there, it's going to say I'm going to create the objects for them the way they should be created. Because I remember, I don't even know. That I could go out 'cause. I made my code may not do that. So I'm going to say these are the cities that I know exist in my database. I'm going to use those. And then when I do something like this OK where I I can go out. I now have a fixed database. I know exactly what's in each city has. And I I know what's going to happen when I create New City III predetermined all that. So now I can run a test and I can go do something where I start testing my GDC City DAO methods. Right. So when you see get city return City correct city for ID. How do I do that? Well, I do it by. Getting the city with a particular city ID. I now know what it should look like. Right. It should look just like this one. So then I call this thing and this is I again. It's sort of like doing your macro, you know to city. One of the first things you're going to create is this. A certain cities match. Or assert whatever The thing is matches. You're gonna create that when you do DAO testing for your object. And that asserted cities match again is down at the bottom. It's very similar to what we had in our map row to city. In that it's gonna just check each of the things it's gonna say my expected cvid and compare it to my actual cvid. Now what are expected natural in this case? They are when I called it up here. I am. I am for whatever my kind of test is. I mean, you get the thing that I think I should be getting. And then you could compare it to one of my constants values. So the expected is going to be the constant value that I know that I should be getting. And then the whatever I've actually done in my test is going to be the thing that I is my actual. That's what I got. That's what I tried when I when I did this thing, if I tried it, if I had a method which, like we created before, which is add a, you know, whatever it was. I had 1000. To the population. Right. I would just in order to do that, I would have, I would create a new constant which was like cityone adjusted. And they had the population plus 1000. So I have something to compare it with, so this is where you set up these constants to be able to have something to compare it with. That's what you expect. That's what you want, and then you just use them down in here. So I can say I actually tried two of them. I tried go out and get city one go out and get city two. Do they match? What I know should be there. So that's the basic way. Interaction is you're going to have constants which are the thing I just created this object to match what I think should be there, and then you do whatever, you know, manipulation and compare it with the thing that you actually do. This is much easier than compare all the individual things in your test. So you just this is the normal thing you'll have. Return null is easy, you just do an assert null. I I go out and I get a city in 99 there shouldn't be a city 99 so I insert know so that's relatively simple one to do. And then you start getting something like this. What do you do when you are? Return all cities for a state. Hey, Brian. Yeah, for these, could we annotate the IT looks like for the first two the arranges all done up top. But could we? Do the arrange, act, assert. It helps me break it up personally for the test. Like what you're doing? Yeah, these are sort of mixed alright, but yes actually all do the. Remember the three terms, but yes, do it in that same order. And. But just remember sometimes what you want to do. Because integration testing does have so many moving parts, sometimes what you want to do is you will want a an intermediate assertion. Because if you get that far. And it fails. You don't want to keep going necessarily. Alright? But you don't want tests to do is is end. It's pretty good at preventing it. Is, is it's crashed the system. You want them to run? You know? I mean they can. They can have no pointer exception where they handle that just fine. They are doing all this in the try catch it'll throw the. The things again, if you look at your base city go, I think it says here it's in the other one where it says it you know it'll handle it if it throws an exception, but it's just best to be you know processing things as you go. But absolutely do the setup part. You can do the. What are the three terms you all should notice? I should know this, but I don't have a rain jacket. Add everyone. Everyone knows. Terrific all of you. Except for me. I don't do that thing. And in fact, it's. I would put the comments in there so that you have, you know, you know which part you're dealing with at which time. Just be aware that unlike the other one, it sometimes that what you're doing is going to take a series of steps. Hey Ben, I am stupid question sure. Where you're using the city like in the test, you have it where cities equals sut dot get, cities by state, etc etc. What is sut? Add you T is the IT is a magic thing in the J unit which basically says the object which I am dealing with right now. OK, so if I am calling this on a city it is going to magically be able to do something like call a method. On. Let's see if you can actually. It is right? Yeah, I see private JDBC, city, Dao Sud, right. And so if you went to jail city VL alright. This is the object that it's got. So, but it is a I don't remember somebody every once in awhile tells me what SUT stands for, but it doesn't really matter. It's just the term that people use. I mean it stands for the object that I am dealing with right now when I am testing OK OK, source object test or not object whatever I don't you sort unit test something or other. It doesn't much matter but, but this is the normal way to do it. You put something of that type up here, it gets set right here. And then you get to used in all other things. So you could call it something different. There's nothing magic about that, it's just it is align object that I am dealing with at the at this moment for this. So it's the entirety of the JD, CB, City dial test or City Dow. It is that it is an object of this type. We just don't usually use this type for anything else. OK, I think I get it. UK I think the problem is that what what if you called it JDBC you know? City. Dow it would just it would get confusing to see which sentence side of things you were doing. You're just giving it a name because you know my test unit. Bing. There isn't a good name called suck 'cause. That's how would call it. I mean, I I you know, I have actually seen it as different things. I I think I do remember somebody. I remember somebody just called that thing. Because then it's the thing that I'm dealing with. But you know I don't. Alright, so you're gonna do your stuff. And you're gonna do. You're you're gonna set up whatever it is that you you want to get. And range and then you you'll do it. You're gonna see what your your value is with your search. So let's go look at a couple more. I'm here. Null was easy. I'm. I'm just what happens if I have? If I don't have anything. This is getting a list of cities because that's what gets cities by state does. And I know the state names of a or whatever they could have used real names. I think part of the reason they don't in this for this kind of test is to try to keep you away from every thinking about the natural world. This is a test. It's a test data. OK, it's called mock data. That's the general term. It's mark data. Mark data is often set up so that it's very easy for you to tell what's going on. So in this case my first city has, you know, is in State AA with a population of 11, and then area of 111. Guess what? It's very easy for me to look at that object. You know, and one the city one a 11111. I can tell that it's meant to be that city. I don't have to go around and say it is is city one supposed to be in Utah or in in New Mexico or something like that. So it's going to use mock data often does some pattern like this to make it very easy for you to follow what's going on. Right. So this gets a list and then it's going to just say like it does the. Let's check that first of all in our list at the sizes, right, we know we should have two things. You know why we should have two things. You should have two things because. Wait, because they're in the database. I forgot they're just too. Later in the day. So there are two of that state good. You know, with that. And then I know which ones they are there one and four. In the database, so I'm going or one and two and it uses city four. I don't know what it says. It compares the constant with the one that I expect. And it compares it with the thing that I actually get. I guess if I went out and looked up here, I'm gonna look at my test data because now I'm getting confused and I want to make sure that I am seeing the right thing. So these are the things I added. I added aabbcc and then I added a new one. Into the. In a A just sort have two different ones in AA. So I know I should have two cities in that state. So I can my and they should both eat well. I showed two cities, so I go out and I look at the two cities and see if the two that I get. Where am I lost track? Alright, so I I guess you get an array with align with. I'm sorry it was missing sub cities don't get zero is the first item that I'm going to get an CDs dot get one is the second item that I'm gonna get in my list and they should match the one and four 'cause. That was the ones that were set up with a right. I knew it would be I should get one and then I should match that suit. This is you're just trying to say, what can I reasonably expect to get out of this? But remember, get cities by state might call other or any of these might call others of the methods, so you may be testing multiple methods at the same time. You're almost certainly testing the macro. To Sydney. So. That's why when your tests run, may they will fail all over the place in weird places because they are dependent on multiple things happening. If I am testing return all city get cities by state and I go back and look at my cities here and I do my get cities by state. It's going to test that this selection works properly and it's going to test it. Matroda City worked properly. And then I'm returning the right thing. But Matt Rhoticity has to work before this is going to pass, because otherwise I'm I'm not getting the right results. They are all interdependent. Similar to move four, it's generally if I am going to try to test anything. I'm gonna say, what's the way when it works, which is I'm going to specify a city, a state that actually exists. That's the, you know, the plane is in Los Angeles and the sun, right? It's doing what it's supposed to do. Then I'm going to have a test that says does it fail appropriately? If it's, you know. The plane thinks it's in Fargo, and in fact it's in Bangor. So that's the case where I do. Uh, so I did. I checked AMB to make sure that they were both right with two States and then I tried. What happens when the state isn't there? Because that's the end. I should get a null list. They don't get null list. I've done something wrong? so so when I call it in the test. I create the city using this test thing. Get whatever idea it is. Shove it back into the test city and then compare the two. OK, OK. Now I will point out to you again that this this came up as a question about why does in create city? Why do we go ahead and get the city again? Right. This again, this has been one of these debates that the instructors have had. When I do create city here. I have the ID. So the easiest possible thing for me to do would be to just set the city ID in the object I already have and return it. And instead I go out to my database and get the city over again using the ID. Why do I do that? Why don't I just set the city ID and return the object I already have that has everything already set up? Well and and yeah, this is not such an obvious question. The instructors were reading about it and. While I like the idea of the simplicity of not going back to the database, what if you have other? Things in your database that have a default value, for instance, and you're passing in a null. Or that have a serial number other than the primary key. Sometimes let's imagine that I am setting my my primary key as a serial number and I have a visit count. Or a modified count thing on my sequence number on my pabel it's going to get updated every time I update my my that city, right? If I save the object I've got, I've got whatever the original value is, but I need to know what's actually in the database, so I go out and get the city again, even though I just created it because I don't know for sure that there aren't other various other values in there that I that I need. So that's why you get safe from the data. That's again. That was a little bit of a side, OK. So does it have the expected values? That was easy when we update it. We did the same kind of thing. What happens? We set the population. How do we do this? We call the update and we we compare we get the city, we see if updated the way we expected it to update. You're not gonna memorize all this. What you're going to do is you're going to look at it when you go to create another one. You're gonna say this is kind of how I managed it in this other tests. How do I manage it to do the same thing? The important concept is you need to make sure. That everything in your. Java object in your city object. That everything gets adjusted and mapped to the database object correctly. Or if you have a list of them that all of them get mapped correctly. Alright. You are you have. You are expected to do this, so when it's retreat, does it have the values? When I update alright if I delete it, let's make sure it's not there anymore. Right, so I delete uh city. The idea of four. I then try to get it. It better be null. If I go out and get my cities by a one of them, was that deleted city? It should no longer have two city it's it's it's fairly. It's fairly straight, it's hard to create the methods, but it's fairly straightforward in the logic. So. And again, I'm saying this probably 'cause I saw this a lot in people doing the homework yesterday. Go homework. It's very easy to get your brain to get stuck in the syntax to get stuck in the the the. And how do I code this? Or remember to step back and think about the how. What is the logic here? What is the what all what am I saying? I'm saying there are four cities out there and I'm deleting one of them and it happens to be in state AA. So I'm gonna need to test. And again I would, I highly recommend is when you're doing this. Say that thing that you're gonna try to do. Like right the comments, you know right here go alright. Test whether the city is still not there. The city is no longer there. Right. Test weather you know. Test weather. It is return. In list of states in list by state or something just right for yourself. The little test statements that you're going to try to do so that you breaking down the problem into the things you need to do. Alright, let's switch over and look at the I think the park go is one where they. They kind of set it up for us. Right. They went ahead and created. OK, then I go back and look at our test data. Like, be sure we know what we are looking at. We know what the. Perks have, right? So we are going to create these three parts, 1/2 and three. It'll created in these. Different dates, OK. They've got. Whatever area of 102 hundred, 301 of them has camping, the other ones don't, because is a serial. We know the first one is going to be item, you know ID 1/2 and three. This is where we get a little more complicated then we say. Let's add to the park state. Which states the rent. So that we know. We can say that part three is in two different states. And we have all the data, so we know this is what we're trying to map out. It's fine when you are trying to do your tests to go back and forth between your test data and your test and say alright, what am I gonna? How am I going to do this? What am I gonna do? That thing that I'm doing? So I'm going to be testing this data. And I'm going to be testing it with. Right here. So. I'm gonna say yeah, I've got my, my three tests that are the original data. I'm not doing the park state thing. 'cause you remember, we don't represent that in here. But we do need to test it. Was that it is. Available and set properly. What are we gonna put? In our before. Do I know? No, I have no idea. So for the moment, I could do nothing. I could say, well, this does enough. Or I could say let's assume we're probably going to have something a little bit like we had in this one. No, don't try to be smart here. Whoops. I'm the right thing. Don't try to overthink it. Just go back and say well, if I needed to do this in my setup here, I'll probably need something like this in my setup. So let's go ahead. And we're gonna copy. We're gonna say. Alright. This is obviously not the data we need, so we're going to say test park. New park. Part ID is going to be zero because it's going to when the create happens, it's going to happen. So we should set it to 0. Park name. Well, let's. You really invented here and call it Park 4. Data stab Lish is gonna be. 20 or, sorry, it's gotta be. Local date dot purse. I'm not trying to be fancy here. I'm just gonna do whatever the heck was done before. And what date are we going to do? Will you? Today's date 'cause? 24th. Whoops. Right. Area is probably gonna be 400. 'cause it's the 4th one. As camping, let's make another one that has camping. Again, I just kind of said, well, what was I doing before and you know now this is failing because I need to make it a. Class variable so that I can do it. I go check my city just to make sure. Then I'm doing it the same way and it's going to be private city test city. So this is over here. It's going to be, you know, private. Test park. I just say the private part is test part. Setting. OK. Alright. Somebody walked me through how you would do this right here. Before we do, though, again, I'm not going to try to be super smart here. I'm going to go back and look at the exact same kind of thing I have in here and say how do I do it? Right. I know I have my constant perks. What am I gonna do over here? Sure. Gonna have it park the object park. And then park equals shut, that gets get park. Sorry, I'm sorry park. Park. For. Not yet park. Park ID would just be one. And then assert parks. \*\*\*\*\*\*. Perks. Edge. Perenne. Would it be? Park 1., park. Right. And statement. Yeah. Alright. So I mean, that seems legitimate, right? Go back and check if we do the same thing. Yep. Did the same thing go back here to check this? If I want to check a second one, I'll check the the third one actually 'cause I you know just. I don't really care. I'm gonna say I don't need to say the park anymore. I'm just gonna say park. Whatever. 3. Patches Part 3. It's pretty straightforward. OK, I better go back and check and make sure that my insert parts match works. 'cause, it's the first thing I'm doing. So did it do it? It does seem to have all been set up for me. Alright, so that test should work. I wanna try running it. I now say run and see what happens 'cause. OK. So again, like you saw before. This one is a pretty straightforward one. It just passes down. The other ones don't pass. Let's go get another one. OK, so somebody else. We're gonna go to returns null here. We look at our city. One I'm trying to model for you how you were going to go ahead and do this kind of homework. It's not going to be rocket science, but it's going to be careful. It's going to be making sure that I'm following the steps appropriately. So I go back and I look here, how do I check something is null? And then I go back here. You could copy and paste if you do copy and paste and I will just tell you this is a gap because I am trying to what I would ask you to do is don't copy in and then change the values. Instead. Uh. Copy and below or inside or somewhere and then come in and out so that you have a model to look at. But one of the things that I want to say with all of you is. I think too often you're letting intellige fill in things for you. And that it gets harder when you need to see what, especially with intelligent, does a crappy job. Sometimes of it throws extra stuff after it's harder for you to learn to recognize what's going on if you're not typing it. So well, it is definitely in your future to let intelli J you know fill out things and do all that. I would advise you to spend a certain amount of time actually taping all the characters in so you're sure you've got the pattern now and you your muscle memory is there or whatever else. So somebody walk me through. So now I've got it. What do I do now? Park. You would do a park with a capital P park with the lowercase P equals sut dot get park and then you put in like. I don't know 666 and then we're going to Death Valley here no. And then to assert. A certain null for a. Yeah, not not for for park. And that should be it. There is no park ID for. Alright, so I can try running it. But I tell you, it may or may not work because. Like we don't know what other things will impact. In this case, we probably will work. This one is a pretty easy one. Let's go ahead and run it. It will probably give us the satisfaction of yet one more thing is passing. Yay, happy day. OK. So let's go to the next one. We will get more complicated, but let's go to the next one and we'll say, OK, so get parks by state returns all parks for the state. So let us look for a moment at our data here. We want all parks by state. Alright, so we have these three parks alright, park 1/2 and three. And they are going to have. Let's try one end. Three, maybe 'cause parks that should have two states, right? So those little ones, we're going to look at. Somebody walked me through the. Code well or if you want I can even make your life super easy. And I'm gonna say let's copy this stuff in. Again, I'm OK with doing this. One thing I would actually say is from now on when I do it. I'm not gonna comment this out. What I would generally ask you to do. Because otherwise it's going to get very busy in here is that as soon as you finish using your template. You delete it so that it's relatively obvious when you're what you're actually going against 'cause I have watched people do this and then copy off some other template because they were. They had done that earlier. You just mess up your life. So somebody walked me through how I'm going to do this same thing. I have a paid paid you walk me through one of my what am I doing here? You muted? I'm not completely sure. OK, it's gonna be just like down here. Right. Someone else want to take it? Biking. Or Jay, do you wanna take it? List some done bracket Park. Denver parks equals SUT dot. Get parks by state. Not sure what goes near actually back and look at it when you're not sure pop back and you look here. OK, so the state so let's. I say this only because it there is no shame in popping back and forth and and double check yourself. That's what you do, OK? Assert that assert equals. Shirt. Search equals. What do we? effect. We expect two, right? Yes. And. Nash this. Is it gonna be parked outside? So no, that doesn't seem like it is park start size. It's clear just following a sad sack with the same thing. Assert. Parks match. Pronouns. So now we're going to do what do we say with city one? And I think I mean part one and part three or both. When we looked at our data. Part One and part three are both in EA, so let's. OK, so we have. Score and. Park stuck it. Parks that God loves her. Right and. Copy that. Accenture, right, This is tricky. It's very easy to start. Take list three in here or whatever you want to be typing 41 because it's the next thing in the in the row. So those are now the tests that say it's apart. It's a list of two. The first of those items is gonna be part one. The first number 3. If we want, we can go ahead and check one of the other ones, and if we're going to do that, yeah, I'm gonna. Big fans. Say. Again. We need to get this. That's why I have to remember to re get this with the new value. So we get it again. Mistake you off to make this up with the list parking there and again it's already defined. He needs just use it like this. What should we do? Let's look at our data and again go back and forth. This is you look at our data and see what is rational. OK. Shall we look at let's maybe look at three? Because. It's both in both of them. Right. So that's a little funky. So we're going to look at. Look at. You're gonna get away. Sissy, right? I'm sorry. I'm bouncing back and forth. OK, so CC is the state that we're going to look at. Why do we use a CC? How many parks do we expect? And. It's gonna be one. Or is that right? But there are two of them in CC? Now, if you were testing. If if you were. Building all the tests on your own. Not just doing homework. Building all the tests. You might go back at some point and say. I need another one of these combinations so that I have, you know, another state that is in both of these things. It is fine to do that, create another state. Just be aware that all your previous tests. You need to make sure that you didn't change the logic. If I create another test and put it in a A, I now need to make sure that my tests reflected so it is a good idea to spend a little time making sure that your data will handle like thinking through the kinds of tests you're going to do logically, not how you're doing code, but logically, do I have enough? Teachers have information that I could run the tests that I want to do. This is what integration testing takes a lot of thought in. Getting both sides set up. OK, so we did this. We should expect one and we're only going to have one. So we're going to delete it. And what should be? I think it should be Part 3. Hey. So we have done this. Does it return the right thing? So we run our tests again. It is a very iterative process. Hey, Ben. Yeah. Why are we writing over the test that we just paid for EA and not creating like a separate? Long. Just because it's easier than to copy things and paste it really, I mean like we don't need parks. This this list of parks anymore. We've done our job and we already have the variable. If I do change this to more parks. Which is fine. I mean if I want to do that, you can have two different things. But if I do that, I now need to make sure that I do change these or they're still looking at the original parks. I like to overwrite it so that I don't accidentally. OK, what's going to happen if I do this? I'm going to run that same test and what am I going to see? Well, that didn't work, so I'm going to say my list list of park. And then I run my my test. Now I'm going to say this because this is actually a very common mistake to make. Where I I do it now. First of all my test which is get perks by state returns all parks will fail when I click on it. It's going to say expected one and actual two which doesn't frankly tell me a whole lot of information. Right. I might look at this and say, wait a minute, I don't expect one. I expect two. So why is it saying expected one? They like it. It makes your life more complicated because I made a mistake here. You want to, you're going to have to debug your tests, but you want to debug them as little as possible. So you try to do yourself favors. One of the favors is. Only use one variable the whole way through if you possibly can. If I use two, I have to be very careful now to say, oh, this needs to be more perks and this needs to be more perks. Well, wait a minute. I don't. Yeah, more perks. And now it should pass. So it is perfectly possible to do it. But just be aware that if you do do different ones that you need you are, then you can't just copy this stuff here, paste it in here and change it. You've got to be really careful with all that. So that's why I personally tend to reuse the variable so that I I don't make that kind of mistake. But good question. OK, So what do we do? Last thing I do is get rid of all this stuff because it's just going to confuse me. Alright, get park space saver turns empty list. Can we manage to do this without looking at the other template? I'm going to tackle that. You look right above. Here. It's gonna be something like what we have there. Andrew, you wanna try to tackle how we would create this one?

It all below you. What am I gonna do here to, to mimic this for the park where I'm creating a park. And making sure that it returns it the correct idea. Well, I guess exactly like it says Park Capital created Park. He calls Sut dot create park. Test park. Test Park again is that thing we set up back in hour before. The long new ID equals. Created park. Start Camp Park ID. Hand. And we're gonna start. Got a certain true. The new idea is. Insert. We're not going to assert that assert true. Search for new new ID. I think it's a. It's greater than two one. Then we're gonna. Now we even say, alright, we've done some of this, so let's let's go on to our next thing and now we're actually test our city or we're going to test this Friday back in support. Set park ID. It's the new ID. I'm gonna search the parks match. Herb smash the test part and the craving part, creative park. Right. So again. Looks pretty straightforward. Run our tests. I don't even know which like like go that test pass probably. They're going to pass a little more frequently here because I'm not simultaneously, but you will be simultaneously putting stuff into here. And writing your tests for it. That's what you were doing yesterday essentially. Although they created a test for you. What they want when you're creating these. You're gonna create. You're responsible for both sides. You're responsible for filling out the things in JDBC Park info or DL. And in the test for it. And you gotta get them both work. At least we know that they're actually working here. So this, that's the only reason this feels pretty straightforward, but remember, you're actually doing both sides of it when it fails is it failing? User test is wrong or because your implementation is wrong. Bear that in mind is your process. Also, I did want to just maybe ask to clarify for myself. Really, it seems like what is kind of when Andrew is doing the problem like we didn't have a null return for the empty list and stuff like that. Really all these tests and all the implementations do lead back to if we're in what the park go test. So it's really in the interface Park Deo, where they give us in those comments. What the return type should be? They kind of give us the specifics. So I guess what I'm trying to say just to clarify for myself is that in the event that a park is not found, that's an implementation choice, a design choice that you make like you're not always testing to see if the list is zero or if it's null. It's a choice that's made before the test is right. Right. And in fact, if you were, if you were assigned with the task of, if you were assigned only the tasks that you were not assigned the task of of implementing these JDBC park video, but you would essentially be doing is saying the things I need to test, I need to test. Whatever methods are in here, these are the things I need to make sure work. So when you go out and you create your test, I'm gonna like I what is logically my test? Forget per or forget parts by state. What is my logical test rolls? And this will tell you that all the stuff I need to worry about, what's going to happen if I if I don't have something in here? I don't have to test it. But if I do have something in here, there better be a test which reflects every one of these things. In here we are now giving you a whole bunch of stuff here. But maybe we're not testing enough things. Maybe we, you know what could be problems with remove part from state. No, and the issue with remove park from state is OK what does it do when the park was not in that state? You know, so we might go out there to test it. Like when we know there is, it is in that state, but we don't necessarily know. So we should probably test when it is not in this state and things like that well also that brings up another question. So this this, this example is bad or from when I'm just saying if we're looking at this park dot Java file in the homework, they actually gave us that. You know add parameter turn all. They gave us a little more instruction on what in the interface they gave us the instructions of how to implement each method. So that's why I thought maybe that's missing in here. So the comments are not there so. Which is a shame. 'cause I right? Like those are actually lovely. And and you will probably have them in your homework. I haven't even looked at Tony's homework. But. But you will probably have those again in your homework because. At usual, if you don't go back and look at the earlier project that tells you this stuff because it's it's all just sequential and sometimes the stuff to get copy forward. OK, there's another question. I'm a little confused on and you probably said it and I was a little distracted. Maybe when you said it. So when you're looking at this thing inside of VS code. Is that so? If you bring up these code, you have that it's like a mock database. Is this? So is this run in the test or do we need to basically copy what this is? This is right and that was not clear. This is literally run every single time you run a test, right? And could we, could you please show that again in your code here the place where it's running is this. That is in here. This is the thing and I will let me just make it clear where it happens. Alright, it happens. Right here as part of. So this is a a constructor. OK, so the constructor is data data source, right? So if I go back to my just so you understand where the things actually happen, I go back in here in my setup. Where I do my my before and I'm creating a new city DAO with the data source. The data source self got created. Wait a minute. I'm sorry. Probably the biggest risk is created in here is probably the top hurt. At some point it's going to create the data, the data source. So when it creates it. It's going to do that in the construction of the data source. It's going to do the load each time, and for now we can just accept this on faith that this will be delivered right. And so and then fundamentally, yes, the important thing is that whatever your test data is, because even if you accept that on faith. You may change your test data. And you may use it. But accept on faith that all that. Scaffolding crap that's in there. We'll load this every time. So this is the actual database that if you're going out and using it to help design the tests were coming back to this because it helps us make the little mock tests inside. Like, this is what's already it's already set up. Actually objects which are manually creating with the data you're you're doing the actual manual mapping between the two rather than doing your VL because what you're testing is that your DL is doing it properly. OK, then the last question I have is, have we done a test? So it seems like these entities that we're testing, so these tables in this mock database. Have we done stuff yet in the tests where there are other like if we want to delete something where there are dependent foreign key restrictions, have we done anything like that yet? Using the tests how you you just do it like you normally would in the in the in you would just program it as if it. Stumbling over my words here, I know what you're trying to say, and the reality is we haven't. Reality is we don't have to worry about whether we have done it or not. OK. But if we look at the JDBC Diego State Diego test, all right where it's going to test the delete. Right. Haven't been tested live state. Every dog. OK, well, let's look at the park then. I don't think we've actually done it, but we're going to have something where we say deleted park and we're going to delete a park. And that is going to be. It's gonna have to deal with it with the dependent banks. OK, OK. Now the reason I point that out though is when we get to that, all we're going to do is call delete Park. We're not going to worry about how it does, yes, because we're testing the JDBC Park Dow that jacks the thing. Best team that did its job? We don't in this point care. But if it didn't do it properly. Then this would fail. In other words, we might try to delete a part and then we retrieve the part. What that's going to tell us, remember, you're not just doing this for the sheer sake of writing texts. You wanna know that your functionality works, so if for instance we did the delete park and we never took care of the dependencies. The SQL call is going to fail when we do our delete part. It's not going to delete that part. And then we're going to try to retrieve it and we're going to say, wait a minute, I can still retrieve it even though I tried to delete it. Right. So this is actually going to be for the first time at Test that's significantly more simple than the implementation exactly where the taste is really simple, but it. But it will fail if it. If The thing is there. So in fact, let's go down and do that. How is. How are we going to do this right here? Deleted part cannot be deleted. Or cap can't be retrieved. I've got to be fairly simple. We're going to we have to call our delete park, right. So if we look in here, I think there was a delete just to to model it. Deleted City campaign deleted. OK, so there's a fairly simple one. We do it like this. Then we try to retrieve it and make sure it's null. Right. I'm not gonna get into the other part. I'm just going to say let's do this then. So if we go into our , VA oh. Right. We're gonna be trying to do something like this. OK, we're going to go ahead and we're going to. Commented out so that we don't. We're not going to comment out the final. Boy, if you ever wanna see things really mess up, leave off a curly brace in a test because. COD is ugly, OK? So again, we don't care. City didn't have any dependencies, perhaps. Maybe it did. I actually does, I guess. But if we do a set. Dot. Right. So which park should we delete? Write. We look at our data because we want to make sure we're testing the right sort of things. Which part do we want to delete? We want one that has dependencies. Because that's sort of what we're we're particularly curious about. Let's just let's delete three because it should get rid of two of the dependencies here I don't know. Right. Now, again, remember that the way we are writing this, I can delete Part 3. And it will not affect any other tests. And as part of the way we prove to ourselves that every single time before it runs a test, it's going to go ahead and load up that entire test database and create it. It's part of the reason why we don't create huge mock data databases because it would just slow down this process. We want the minimum we need to be able to prove our point casually. You're going to use against a a mock database where you don't do it exactly this or you have one set of tests that uses a different set of mock data. That's huge. So that you can deal with situations like do I make sure that I get all five million of my, you know, whatever thing it is, but most of the time you're not going to do that 'cause. It would be so incredibly slow. So I'm gonna do this. I I deleted it. It should be gone. And then I'm gonna test whether or not the city is there or the park is there. So it's the park park. Right. Retreat unless they retrieved part. Perk. People start that get hurt. And try to do the same one. Then I want to say assert. Shirt now. 'cause I am only. During the thing alright as retrieved, retrieve her. If I have not handled my delete properly, I have not taken care of. Like if you look back 'cause, we'll see that it probably has, but if I look at my. ABC Parkdale, where it deletes. You guys have been through this same thing. They delete it in two parts, so we delete. From Park State and delete from park if you had not done this. OK. Right. Our our test is going to fail because we are this is gonna fail. Because we didn't delete the dependencies first. So let's see how that actually looks when you do that, because remember you may be writing both parts. You may be writing the DEO and writing the test. So when you do something like this and you run this. OK, well it would help if I. OK, so part. It's called deleted. Part cannot be retrieved, failed and when we click on it. When you say. This is right. I want to encourage when test fails, don't just say it fail. Click on it and see what exactly the failure is. You you're responsible for figuring this stuff out, and part of part of your job is figuring out why. Don't not, trusting that it just failed but saying. But what about it? Fail? Well, it says my statement was called and there was an error return which said. Update here delete on table part violates foreign key constraint. FK Park State Park on Cable Park state. Alright, I've got a dependency problem. So if I were running this product this thing. The key is still there. I have a problem right because it's still there, so I then would go. I need to know my my test is not wrong. My test is great. It caught exactly what it was supposed to test. It failed, but that wasn't in a good way 'cause it told us what's wrong. Alright, so then we go back to our. Here and we undo. Those two things we put that in and now when we run our tests. Park remove from state now it's not in list of parks we didn't implement that one. So where is it? Completed part can't be retrieved that passed. It passed because we did delete the dependencies, so that is where yes my my actual test is relatively simple, but because we don't care how complicated these are, some of your details, so they're going to be even more complicated in this, they're going to have all kinds of steps. OK. In one other question too. So in our tests. It seems like, at least for this read for now, it seems like we don't have any tests. We have a test for the individual entities. We don't have any associative tables in the tests. Is that probably always going to be the case? See, tables are kind of like two table like it's a sequel specific thing to get something to work in SQL to have an associative table but cannot test associated tables on their own. You test whether they are, whether they return the values they should. So when we did for instance our. We did get perks by state. That is actually using associated tables. We don't care that it's using them. We just care that somehow in the database manages this process so we don't ever directly test the park state. Then we indirectly test it by whether or not the things that were required to do work. Is that? Yeah, yeah, certainly. Keep going. Sorry, I mean. Imagine for a moment because this is where you know, imagine that we had a whole bunch of tests like this. And somebody decides that they want to replace. The SQL back end that we're using. With a go another Mongo DB database thing. OK, totally different DEO implementation. Right. All we would have to change are the the general database config. And B I think the basic test would probably have to get next, but we might have not when that night have a rollback. But these tests are basically going to be. Set the way they are. The test these tests should not have much awareness of. The actual database backup. They're testing the functionality. Right. Into clear up conceptually here the. Part of our Java code that would be aware that there are associative tables in the in the database would pretty much be not in the test but in the JDBC parked out. That file is the only it is, it is that's the only that's the only place in our Java code that would even know that there's a special right and that's. In terms of different? Different parts of our Java code have different jobs. And back to the whole general thing, and this is a very important concept to all. Keep in mind the model is supposed to be just what our entities, the way we represent them with plain ordinary Java objects. The interfaces say the rules of what kinds of functionality we have. As I've said before, we have these cut. These are the things we can do. It is only when we get into the JDBC park video that I am now starting to interact with an actual, you know, Postgres database. So. And part of the reason I said is, OK, what if we didn't have Postgres? What if we had SQL Server 'cause you can connect to SQL Server or MySQL. Those would only be changes in these three files in the JDBC city, VA, OH, and BBC Park. Go and state DL. We wouldn't even change the other person, the configuration, the test. Because it would only be here that we are starting to deal with the guts of what is in SQL and these would have changed much either they would change change a little bit. So. In here is where we get into things like. Yes, we have to know that we have an associated table. In our test mostly we just say if you delete a part it should work. So we've got separate responsibilities there and and the JDBC template which is part of the Spring Framework. Move framework is part of the screen framework. That is the thing that handles. It doesn't matter if it's Mongo DB or anything, it just does that automatically for us it does. JDBC is specifically connecting to. Well, yeah, actually it's connecting to anything that JDBC connects with, which I don't know whether it has a JDBC, you know, a mongo DB. Driver. When we looked at our stack of things, though, there's something underneath this which has drivers for certain particular databases. It certainly has one for my SQL and SQL Server at any other in Oracle and any of the other relational databases it may have them as well for Mongo DB and things like that. Just know that they don't have. But that's sufficiently different. This sequel may or may not drive. Good. So. Looking back at the park, PA oh, so we have. I I'm not going to go through every one of these, I don't think, but let's see if there are any others that we have to to deal with as. OK, so you get things like Park added two state is in list of parks by state. Right, and wish to that one and then we'll you know. And then perk remove from the state is not in list of parks by the state. That's really the same test that we had up. Where we do the delete by park. I'm sorry that the delete and we could just check is it still there? Let's not worry about that. Let's do the the added two Part 2 park added two state. OK, so if it says Park added to state which. Interface thing are we going to be dealing with, we're going to be dealing with. We have add part to state. That's the one we need and it's gonna use those two things. OK, so. I'm gonna go and say I need to test that so my. I am going to call Scott dot. And park to state. And I'm going to add a perk ID. Well, which park should we add to which state? Let's add look at our back at our thing. Let's add. Part 2. To help me clearly. Everything touches on whatever state it is in his AA, so if we say we wanna add Park 2. Which is number 2 and we want to add it to a, so there will be three. Three different parks are touching that state. We go into our. Come here. We can say so. We're adding Part 2. And we're going to. A A. And now? What are we gonna assert? We're gonna try to see. We're gonna do the same thing as our. Our get parks by state. Alright. Well, where's? Yeah, sensually gonna do this same test right here. We're going to say, does it have the things that it needs to do? So let's copy it from there. And we're going to say we now need to say. List perks. Get parts from A and they're now we know should be three instead of two. So let's say 3. We should say park. And then. Now this is where we have to be a little careful. What order are they going to show up in? Well, they're gonna show up in this order, I think. How would we know what order they would show up at? What is Jeff Parks by state? What are the rules? This is a little weird and was talking about with. You have things that might have been told to us, but let us just check implementation wise how we actually do it and see if it works. When we go to get parks by state. Hey, we're doing a select from this joint to this where this order by Park ID. So they are going to show up in the order of the park. Not by park name or something like that, which we have to worry about, whether we have a we we wouldn't have a problem here because we are you doing the same way but so we we know that they should be ordered by Park ID. So we know we should be able to do this all three of these should be in. Am I run this? Hans, we have part added. Two state and listed parks by state. So what's the test that we don't have here that we possibly could have here? Because remember, we're not constrained to just the tests that are already here. If we want to test park outages, add park to state. What's another possible test that we might use? And park to city. No 'cause, there is no associated there I, if we were really just going to use add park to state just just for that one alone. What are some ways we could test it and see how it behaves? I can think of two more right away and this is I'm trying to get you thinking about how you think about the logic on this. What is going to happen if we try to add a park that doesn't exist to a state that does exist? And what's going to happen if we try to add a perk that does exist to a state that doesn't exist? Will it return on possible things right? I'm sorry. You were saying turn a null factor, no value in one of those comps. It should return something. What does you know? We don't really have a definition for it, but OK, let's say we we take those two situations and again we go back to our. Implementation for a minute, which is we add. Park to state. Bring into and insert into and we use these two things and we're going to do an update. Probably it's gonna fail in here with a SQL error if we use a park date, ID or state abbreviation that don't exist, right? So both of those situations are going to add are going to return. I'm going to change nothing. Let's put it that way. They're gonna change nothing. So. What we should see is that we get the same results as we would get otherwise. So I could say and I'm going to combine the two because they're they're really in a sense the same test. Partly trying to remind you, you get to add tests to you don't just add the one you get to say what else should there be? No changes. Nothing. And I D4. State. Is invalid. So I could say, OK, I'm gonna add, I'm gonna say. If I try to add 99. To stay A and then they get a A. Then what should my number be? Remember, we didn't do that other the other add now. Should we do? I should be like this practice should be exactly the same thing as that that other test. Right. And then we could say well, let's. Do this say? We're going to do. Park again. Right. And we're now going to say we're going to add Part 2. OK. Right. Yeah, 22 state abbreviation. \*\* or YY, OK. And now when we get now, we're still checking a. Does that make any sense? I'm not sure it does. So what happens when we do? Why? Why? Alright, we should expect. 0. So that is what happens if we have invalid data and when we run our tests, we can now say do we have invalid data? So. Think about what happens when the parameters fail and this park outage state changes nothing and what does it say? It gives errors, it gives SQL errors. I'm OK with that. But this is a case where you're not that pass. It's gonna fail. No, I don't know whether there's a way to suppress the SQL errors. I think there maybe, but I don't know it off the top of my head. I will look at that up and let you know, but the test was designed to fail. Therefore it's passing, even though it's not because it's failing in this party in the in the known way. There may be a better way that. saying this will throw an error i'm serious no should row error. If assets who? I probably say something. I could probably come up with an easier way of saying that, but like yeah, I have no problem making me as long as you want so. But I'm trying to say there are cases where you want to say, well, what happens in this case. OK. Go back and look at my stuff for a minute. This out of the way. Look through my notes and make sure there is nothing else that I need to. Urgently tell you all before we. Send you off to tread and do more of this. Yeah, I will look it up 'cause. There should be a way to make that pass. By perhaps doing by actually catching the maybe if I tried it and actually catch the SQL exception. Because remember, in here it is. Catching the sequel somewhere or other is. It was there it is, or throws the sqlexception. Let's get a quick try. Remind you about trycatch things. What happens if I say? I want to do this and I'm gonna say end. Say try. Then I will put it. OK oops. Say now what would I do here? We have to name the exception in the catch. Thank you for the variable name. I think you're right. But what I what would I do here? OK. I could do a search. Assert true. A little risky, but I I could. Essentially I could test that it gave the right error. Around each one of them. And that would essentially say I would pass my test if I got the the SQL error which is what I should get. This is just an experiment. I have no idea what it or work, but what is it having a problem with? Probably needs to shape context. We catch for Java dot SQL. I don't want to. I want to catch it. Alright, so it doesn't work. Never mind that. I was just trying to see if there was an easier way to do it. You're gonna. This is one where you actually want it to fail. You want it to not pass a not to pass the test. I'm OK with that so long as you've somehow indicated. Alright. So now what are you going to do today? You're going to go off. Let's take a quick look at what your homework is for today so that we have a. You can all grown loudly. But let's let's see what it says you should do, because that might help me to set you up for it properly. Tested. OK, so you're learning objectives, how to write tests and how to find bugs. There you go. I like that. By the way, I will point out that. In your, the project must not have any build errors. One thing which is it, it's come up a couple of times and people doing their there. Database designs. I should be able to load your database design. Without it raising any errors. And that is like having a project can't have any build errors. I can't test things that don't load. Now I'm being like, I don't really care and I'm going ahead and and and fixing the errors so that I can do it and just make your comment on it. But just be be aware that. Anything that's gonna be evaluated. One of the last things you should do is try to run it and make sure that it actually compiles, that it, you know, runs that it loads whatever else that is just is something you should all get used to. Whether it says it in the requirement or not, you should always know that you can have things that don't work. But it should compile because otherwise it's very. Otherwise the tester, the evaluator the greater in this case boot camp OS So what does this say? We need to do. We need to start with some code which let's see which one we're going to work with today. We're going to work with. The timesheet. Yeah. OK. So you've got a timesheet model. And you're going to write some tests. The cats are down here. In the. OK, here are your tests. You're gonna be doing. You're gonna be filling out these tests right here. Does it want you to write any new tests? Alright, this is a very important one for you to keep in mind. 'cause you may not read all the way to here. When fully implemented, five of the test passed and four continued to fail. After you do this. So you are expecting some of them to fail. And then you should fix the bugs so that they will all pass. Alright. If you are doing the test right. Five will pass and four will fail or whatever it is that it says here. And then you have to go back and fix. This logic. The interface does have clear expectations about what should be happening. So you do know what should be happening? You will have to fix bugs that are in here based on your tests. So you are writing tests. Running them. And then finding out which ones don't run and going back and fixing the DNA the database. Is this homework or Paris? This is hard. Alright. Good day window. I will now that enough from that. Let me show you what the pair is project is OK. It shows under review day. OK. You've got a schema. OK, you're going to create the the data source. Stuff for it. Gotta load the test data. Rollback. It's very familiar stuff, by the way. Now you're going to create both the methods and the tests. So your job is to is to do the combination that you. No, that you're dealing with stuff that is very, very similar to what we've just been doing in class today. So looking at that lecture final for today should give you huge clues on what you're doing here. But your your what you're trying to do is stepping you back one step further and say let's get the whole thing put into place. That's what the parish project is. Just don't try to do it without looking at what the lecture from today 'cause. You'll need the lecture from today. OK. After class I am going to post the pairs things and I will give the permission. So you'll start seeing invites in whatever the pairs don't even touch the pairs project. Until you have started on and worked a little bit with the homework for tonight, even though the due date is the same for both of them. You will be better off giving going through the exercise for tonight before you start on the the parish project. I will be around. The rest of the day I have an hour long meeting so but it overlaps with your your employer showcase. So like you wouldn't be talking to me anyway. Don't hesitate to reach out. I would start. I would go into the things actually let me amend that. I would probably go into both. I would look into the exercises for tonight and make sure you know what's going on. And then I would look into the parish project and make sure you know what's going on so that if you have questions for me, you can ask them today. I will be relatively unavailable. On the other hand, Colton and David should both be in Tori can all help you. After this they got the same project as just you know in C sharp, but they should all be able to help you. So reach out to them. But but today I will be available tomorrow. I won't so look through things enough that you know what to do. OK. Alright. We're going to call it quits for today. and a good with