Welcome to Module **9,** Bulk Processing Part 2. So here, we're going to talk about how you can create a map/reduce script. The processing of map/reduce script is a little bit different with any other script type in NetSuite. So you need to make sure that you pay attention to this one, especially with the map/reduce script being a very useful script to process large amounts of data.

Also, we want to understand how we can process or how we use distributed processing to execute map/reduce scripts. And then we're going to compare. What's the difference between a scheduled script and a map/reduce script? So considering that I can use both for bulk processing, I want to know when I'm going to use a scheduled script and when I'm going to use a map/reduce script. So that's what we're going to cover in this module.

So let's talk about a map/reduce script. So what is a map/reduce script? So this is a structure of a map/reduce script execution. And basically, if you want to process a large amount of data-- when I say large, I mean thousands upon thousands of data-- then you want to consider using a map/reduce script.

A map/reduce script allows you to perform something called distributed processing. And we're going to see later how that distributed processing works. But basically, that distributed processing means that you can actually run multiple sections of your script at the same time. So unlike a scheduled script where you have one execution, you can have multiple executions.

So in this case, you can see that I have the Map stage of my script running four times, OK? But again, this is just an illustration. You can run as many scripts as you want with a map/reduce script, or as many stages on your map/reduce script as you want.

There are several advantages of using a map/reduce script. Now, one advantage is distributed processing. And again, this is unique to a map/reduce script. Unlike other script types, where you only have one execution, the map/reduce script allows you to have multiple executions of the same queue, OK?

Now, aside from distributed processing, you also have automatic governance handling. Now, we haven't really talked about governance yet. But once you start developing your script, you would know that governance would be something that you have to really think about whenever you're developing your script, OK? But with a map/reduce script, governance is not an issue because governance will be handled automatically by the system.

Also, you have custom-based time yielding, which is only available with a map/reduce script. With a scheduled script, you have to wait for the script to stop running before you can actually move to the next script. But with a map/reduce script, what you can do is that, OK, maybe I want my script to execute only for 30 minutes. And then after 30 minutes, I want to let other scripts pass through the queue so that I'm not hogging all of that queue execution, OK?

So if you only have one script queue and you did not opt to purchase a SuiteCloud Plus upgrade, then what you can do is that you can just have a map/reduce script and say, OK, maybe I just want to run this script for 15 minutes so that other scripts would be able to run. So that's something that you can do on a map/reduce script.

So you have automatic governance handling. This is something that is really big, especially for SuiteScript developers. We have custom-based time yielding. We also have the ability to have reports. So we have built-in functionality we can use to handle reporting.

And then lastly, here, we have a built-in system for error handling. So we don't need to worry about catching certain errors. If you get to the summary section of your script, then you'd be able to tell if something went wrong in the other stages.

Now, with the map/reduce script, what you need to do is you need to really think about the script in stages. Unlike a scheduled script, where you have a script and it's just one gigantic block of execution. For example, here, you can see that this would be the structure of a map/reduce script, and this would be the structure of a scheduled script. Which means that if you push a scheduled script to the queue, it will run from the top of your script down to the last before it stops executing. So that's kind of what happens.

Now, the problem with a scheduled script is that if the scheduled script is processing for a very long time, there's really no way for you to stop the execution without affecting the actual script, OK? So it's not like you can, OK, I'm just going to chop that scheduled script off at this particular point and just assume that everything would be OK. But that's not really the case. Because if you do that, you might have some processing here that is necessary for the processing here to function.

So there's really no way around that. You have to wait for the script to finish before it executes. And also, since this is one block, one gigantic block, if you push your execution to one queue, it will stay in that queue until the script finishes. Now, comparing that with a map/reduce script-- so here, as you can see, the execution happens in multiple stages. And in one stage, you can have multiple executions of that particular code chunk.

So here, we can see that we have four stages on the map/reduce script. We have the getInputData stage. We have the Map stage. We have to Reduce stage. And we also have the Summary stage. So we have the four stages here.

Now, we're going to talk about what those four stages are in detail in a bit. But basically, what happens here is that the first execution of your script would be treated as one script, OK? So the getInputData stage, that will be one script. And it can be pushed to any of the queues that you nominate for this particular deployment.

Now, after you execute the getInputData stage, it moves down to another stage. And the Map stage would have individual execution per record or per data set. So for example, if you have, say, 1,000 records, you can have, potentially, 1,000 executions of your data set.

That means that instead of waiting for a script to finish in one queue, you could potentially push all of the execution in all of the queues. So if you have five queues, then all five queues are maximized. And if you have 10 queues, all 10 queues are used. If you have 15 queues, then all those queues would be used. And that would heavily, immensely speed up the execution of your script. So this processing is just really, really fast compared to, say, a scheduled script, especially if you have multiple queues, OK?

Now, aside from the Map stage, once the Map stage is done, you go down to the Reduce stage. And the Reduce stage is basically a grouping of the data that you did in the Map stage-- again, more on that in a bit. Then after the Reduce stage, this would be the last processing stage of your map/reduce script. So everything should end here.

You have the Summarize stage if you want to have reporting. Or if you have errors on your script, you can also have the error handling over here. So those are the different stages. And again, as you can see, the map/reduce script is more efficient than your scheduled script because a map/reduce script has the ability to break down the execution.

So if you want to yield your script, like if you want to have other scripts execute, that's fine. Because if you chop your execution right over here, that's OK. Because this would be a separate set of execution. In fact, if you chop your execution right over here, that's still going to be fine.

Because again, that would be a separate execution. It's not going to affect any of your code, OK? So it's very flexible. It's fast. And it's just really convenient to use the map/reduce script.

Now, how does the map/reduce script work? So let's try to look at the different stages one after the other. Let's start with the getInputData stage. So getInputData stage, as the name implies, this is where you would be getting the data that you will be using to process your script, OK?

So for example, here, this data that you have could be coming from a search. So that's a very common. So if you have an array-- so maybe you have a large set of data that was passed to you through an array. That's something that you can also process.

If you have an object-- so maybe you have a large object, like a JSON object or something like that. You can also use that as input data. Or you can actually have a reference. So this is the neat thing here. If you already have an existing saved search, you don't even need to load that search on your getInputData stage.

So you don't need to use search.load. All you need to do is just point that data and say, OK, I have a search at this particular account. I could just say search, or type search. And then ID is the internal ID of the search that you want to get. That's it.

So just create a payload object. And then return that payload object. And the system would know what to do. So that makes it very convenient. Because you don't really need to think about anything else. Just pass the data, and the system would automatically handle that data for you, OK? So that's the first stage. And with the getInputData stage, this is a single execution. So it will always just be pushed to one queue because you just need to prepare your data once.

Now, once you have your data set, what's going to happen is that the system would execute that data set or parse that data set for you. That means that you do not have to parse the information on your getInputData stage. All you need to do is just tell the system that, OK, this is where I'm going to get my data, OK? So you're not going to run your data. You're not going to choose between a page range of result set. We just have to create the search object. That's it.

Now, once you have that-- for example, if you're using a search object-- now, the information will be passed here to your Map stage. And what will happen on the Map stage is that it will be processed per result or per line on your array or your search result. So for example, if you created a search on your getInputData stage and then you have, say, 1,000 results on your search, then each individual result will be passed here, OK? So again, you don't need to parse the information. The information would automatically be parsed for you.

Now, on the Map stage, you can use this stage to finalize your process. Maybe you just need to total something or you need to compute or transform certain sets of data. You can have this stage and then just ignore the Reduce stage. So if you have a Map stage, the Reduce stage becomes optional. And also, if you have a Reduce stage, the Map stage also becomes optional. So you only need one or the other.

You can use both if you need to. But you are only required to have one, OK? So if you are going to use both, the Map stage and also the Reduce stage, know that the map stage is used to group key-and-value pairs, OK?

So what happens here is that if you have the data set, what you can do is that you can get the data that you want. And if you want to group them per value, whatever value that would be, just return that to the execution. So write that back to your script. And then the script would process how the grouping happens.

Now, to give you an example of how that works, let's say I have a search, and my search has four results. So I have a result where my key would be the name of the customer. And then the value would be, let's say, a total.

OK. So what's going to happen here is that I have two keys. Maybe I have two transactions for ABC Marketing, one totaling to 450 and another one totaling to 850. I have another one for B&B Design. And then I have another one for Gentry Inc. So these are my results, OK?

What will happen if I push my execution to the system? Let's say, OK, I want to group this based on a key-and-value pair. I will write the key and the value on my request and then tell the system that, OK, I have a key, which is ABC Marketing, and a value of 450. If I do this again-- for example, if I write another key to the system and the key already exists in the system, what will happen is that the system would automatically say that, OK, I'm going to group all of those values and put that in an array.

So that means that I'm not even grouping the keys. I'm letting the system group the key-and-value pair for me. All I'm doing is that I'm returning the value to the system and just have the system handle that for me. So as you can see here, since the key already exists, then the system will just put that together and say, OK, now for this particular key, you have two values. You have one for 450 and another one for 850.

Now, in this case, since these two keys are individual keys and you don't have those keys in the system yet, then that will give you a different entry, so one entry for being B&B Design, then one entry for Gentry Inc. So that's something that you can do on the Map stage. Again, you don't need to group your data. You can group your data if you want to. But you don't necessarily need to do that.

Now, again, the mindset here for a map/reduce stage is slightly different because there are a lot of steps that you are skipping. If you're dealing with a normal script where the developer would have to think of everything-- so if I'm going to do the same thing here, what I would do is that I would create the search. I would run the search. After running the search, I would parse that so I can get the name of the customer and also the total. And then I would have to create a loop within that statement so that I can group the key-and-value pair.

But those five steps are things that you would do if you only have one singular script. On a map/reduce script, you only have two steps. One, you create the search. You don't even need to run it. And two, you parse the data and then return that to the system. The system would sort it out for you, OK? So we just skipped a lot of steps here. So again, when you do this, the programming style would be a little bit different. But once you get the hang of it, this is a very powerful script to use.

Now, moving on to our execution-- so next, we have the Reduce stage. And this Reduce stage-- as I mentioned, this would be the last stage of our processing. So when we get to the Reduce stage, you would have a smaller data set. Because here, in this case, remember, in the Map stage, we grouped everything together. So by the time it gets to the Reduce stage, it's already grouped, OK?

So the data that you're getting in the Reduce stage would be a set of key-value pairs. So here, getInputData. You use a search, for example. On a Map stage, you use an individual result of that search. And then on the Reduce stage, you have a key-and-value pair. So you would always have a data set. And then you have to process that data.

So on the Reduce stage, you can do whatever you want to do. Just finalize your data. And then after that, the last stage would be the Summarize stage. And the Summarize stage would give you information about the execution. Like, how many yields did the system do? Or maybe if you have multiple queues, how many queues were used, and things like that?

Or you can also use this for error handling. Because error handling would not be as easy if you're dealing with multiple records running at the same time. So OK, what if this particular execution failed? I want to know why it failed. So you want to know how to process that.

So any failure from any of the stages would be passed onto the Summarize stage. So you would put all of your error handling code over here, OK? Again, as you can see, the mindset is a little bit different. But it's not really that hard to use.

So moving on here to our discussion, let's talk about distributed processing. And then we're going to look at how this is used in action. So with distributed processing, what happens is that-- let's have an example with a scheduled script first. So for example, here, I have three queues. Let's say I have five queues, but I'm just looking at three queues here.

And then on the first queue, I have three scheduled scripts that are currently in queue for Queue 1. For Queue 2, I don't have any scripts queued and. Then for Queue 3, I have two scripts queued. Now, if I create a scheduled script, remember, on a scheduled script, you have to decide where you're going to deploy that scheduled script ahead of time, OK?

So that means that I don't know what's going to happen when I execute the script. Would the queue be empty? Or would it be full? Now, in this case, the queue is kind of full because you three scripts ahead of the script that you have scheduled. And you don't really have a choice.

Now, in this particular execution, we can also see that Queue 2 is empty. It would have been really great if we could have pushed that execution to Queue 2. But again, that's not something that you can easily do with a scheduled script because you have to choose which queue you're going to push your scheduled script to.

And with a scheduled script, remember, even if you have multiple queues, you really don't have that flexibility to know what's going to happen during execution. Now, comparing that with a map/reduce script, what happens with the map/reduce script is that instead of selecting one queue for execution, you are going to select multiple queues, OK?

So let's say I have five queues. I can say that, OK, use all five queues. Or maybe if you don't want to use all five queues, maybe you just need to use two queues. Let's say Queue 1 and Queue 2 would be used for scheduled scripts. And then Queue 3, 4, and 5 would be used for a map/reduce script. That's something that you can do.

Now, since you would be selecting multiple queues, what's going to happen is that the system would automatically push pointers to the queues that you have selected. Let's see here. So we have a getInputData stage. So to getInputData stage was assigned the first, second, and the third queue. So you can see here that I have my getInputData, and then getInputData in Queue 2 and then Queue 3.

Now, since my getInputData stage was pushed to Queue 2, and then I have an opportunity to run my getInputData stage at Queue 2 because Q2 is empty, the system would automatically just cancel the pointers that were placed into the first and the third queue. So as you can see here, the balance of your queues are handled automatically by the system. And you don't need to worry about, OK, do I dedicate Queue 1 for this? Or do I use Queue 2 for this, or Queue 3, or whatnot?

Just select the queues that you need. If it's empty, then it's going to trigger that. If it's not empty, it's going to execute the pointer where it's set to execute first. So for example, if you have like five scripts in Queue 2 and you only have three, then you most likely would be executing your getInputData stage on Queue 3.

And also, since a map/reduce script is executed per stage, not per script, that means that this queuing system is applicable to all of the different stages. So for example, in this case, I have an execution for my Map stage where I have two records that I need to process. If it goes to the second queue and that gets executed there, then it's going to cancel the pointers for Queue 2.

And then let's say Queue 3 opens up. So if Queue 3 opens up, then it'll process the second Map stage in Queue 2 and then cancel the other pointers. So as you can see, it is very convenient. It's kind of like queuing up for something, and then you have friends that you can queue up for you. And then whoever gets to the front of the line first, that's the person who gets to purchase tickets or something. So it's just really very convenient to have this kind of system. And again, this is unique to a map/reduce script. And it's not available to your scheduled script.

In this walkthrough, we're going to just do something simple. We're going to look at all of the invoices, then total that per customer. So we're going to gather all of the invoices on the system, group that by customer. And then per customer, I'm going to loop through that and then just log the totals.

So what we want to learn here is that we want to learn how we can use the map/reduce script. And again, you have to pay attention to this walkthrough because the system of creating a map/reduce script is a little bit different from other script types because of the nature. If you remember from our discussion, map/reduce scripts are broken down into smaller stages. So that's how we are going to process our data.

So instead of creating one gigantic script, we're going to use smaller scripts. And some processes will be skipped because the processes would be automatically handled by the system. And we only need to concentrate on one specific stage. So that makes our processing faster. But again, the syntax and also the way of thinking is a little bit different.

So let's go ahead and go to the walkthrough. Let's go ahead and build our script. So first off, let's close this because we don't need that anymore. And let's create our map/reduce script. So for this map/reduce script, I'm going to call this sdr\_mr for map/reduce, \_invoice\_totals.

Now, for this one, I need my Search module because I'm going to search for all invoice records. And let's click on Finish. And that's it. So we have our search. Let me just change the version number. The first thing I need to put in my script would be the data that I will be processing for this script.

So for this one, I just need to create a search. Now, for the search that I'll be creating-- so here, I'll just call this my invoice search, or maybe my invSearch. And I'm going to use the same method that we've used before, the Create method in our Search module. We could define the type of search that we need, so type equals to search.type. And then we're looking for a transaction search. So that's what we need.

Now, after my transaction search, I need to specify my filters. Now, for the filters, this one should be a fairly easy because we only need one criteria. Well, in fact, actually, we need two. Because this is a transaction search.

The first filter is that we need to look for a type. Now, instead of using the Create Filter method on our search object, we're going to use what we call search expressions. Search expressions would be a way for you to quickly create a search but also offer flexibility so that you can configure your search depending on how you want to build it.

So for a search expression, what that looks like is this. For example, if I want to create a search for my type, instead of just specifying a Create Filter object, I have an array. So here, you can see I have an array. And then I have my Type property for my array.

And then for the next element for my first [? array, ?] I have my field name. I'm looking for a type specifically. And then I have my search operator here, so search.operator, same operator object that we use previously in our search. So I have for any of because this is a search. So ANYOF for a list field.

And then we're looking for our customer invoices. So here, I'll just see CustInvc. Now, for this particular value, you can find this information on the Help Center. So there are a list of different transaction IDs that you can use for your search. So here, you can use this for a search expression. But you can also use this for a regular Create Filter method.

So anyway, as you can see here, instead of using the Create filter, I can just specify this array. And it's a lot shorter because all I need to do is just specify my filter, and then my type, and then also my value. Also, what I can do here is that I can just add in a condition.

Normally, if you are going to use the Create Filter method, it's always going to be an And. And here, I have an option to use either an And or and Or. So you have more flexibility if you're going to use search expressions. It's not only faster, but it's more flexible.

But anyway, for us, I'm going to use an And. Then another criteria that I want to add would be my mainline search. If you're going to search for a transaction, one thing that you would notice is that all the different lines on your transaction would be listed on your search. For example, if you have an expense report-- so if you're looking for expense reports, and that one expense report has four different lines of expenses, all of those lines would be included on your search.

Now, if you don't want the level of detail to appear on your search and you only want the totals, then what you can do is that you can set the mainline criteria to True. So by displaying only mainline values, you would just get the total or one line of your-- or the main line of your transaction.

So anyway, here, going back to search.operator, again, the same operator, [INAUDIBLE]. So this would be an Is. And then we want to set that to True. So as you can see, it's really easy to use in search expressions.

I can also use that for my search columns. By the way, I'm just using this to make it faster. You don't need to use search expressions for to getInputData. You can use a regular search, like a Create filter or a Create column. You can use a reference. You can also use an array.

So let's say you have an array of objects that you want to process in your map/reduce scripts. That's something that you can use. Again, it's very flexible. To know the different things that you can include on your search-- so you can just go to the Help Center to get more info.

Now, I know for this one, I only need to return two values. I need to return the entity ID, or the name of my customer-- so entity. And then also, I need to return the totals, the total value, just [INAUDIBLE] total. Again, as you can see, it's just really easy to use search expressions. And that's something I recommend that you do.

Now, anyway, we have already created our search. After creating our search, the next thing that we're going to do is that instead of running our search-- if you remember from the previous module, where we created our scheduled script and, also, we used the search that we have created, we had to run our search and to get the Page Range object.

Now here, in a map/reduce script, you don't even need to do that. The only thing that you need to do is just return the search to the system. That's it-- return, and then your search object. Now here, you can see that there are a lot of things that you would normally do on your script that would be automatic in a map/reduce script. And that makes it really easy for you to process information on a map/reduce script. Because you just need to concentrate on one particular stage.

Now, in this stage, the getInputData stage, you just need to concentrate on getting the data that you need. To process the data, the system would be handling that for you. Like what we did here, we just created that search.

We didn't run the search. We didn't process the search. We didn't parse the search. We didn't have to look through the search. Just create the search, pass that search as your return, and that's it. So you can see it's very, very convenient.

Anyway, so now that we have handled our getInputData stage, we're going to move down to our Map stage. In our Map stage, remember that this would not give you the search results, OK? So it's not going to give you a set of results that you're going to process. But it would give you a singular element on your search result. So instead of getting 1,000 records, you're going to get one. That's it.

Remember that the Map stage would be executed in separate sections. So one Map execution would be one result. So that means that if you have 1,000 results, you would have 1,000 different Map executions. So that means that you don't have to create the For loop to parse the result. You don't have to use the Get value to extract that from your saved search. Again, all you need to do, you just think of one singular element that you're going to get.

Now, here, what I'm going to get would be an entity value and also a total value. That's it, just these two information. Now, if I have a lot of information that I have on my invoices, then again, it's going to give me one pair of each. If I have more, of course, I'm going to get more information. But I only have these two in my result.

Now, that said, since data has already been processed by the system, do know that the search result would be returned by the system as a JSON string. So you have to convert that JSON string into an object so you would be able to parse the information. So here, I'll call the search result. And note that I'm using searchResult, not searchResults because, again, I'm just using one data.

So I know I've been repeating that information. But this is something that you have to think about. Because this is different from what you might be used to if you're processing other script types where you just create one singular script.

Now, for this one, anyway, JSON.parse, just the normal JSON parse call. And then if you want to get the value that's returned by your getInputData stage, you're going to get that from your Context object under the value property, so context.value. That's it.

Now, what this would return, context.value-- if I go to-- so here, I'm going to this website, JSONEditorOnline.org. And by the way JSONEditorOnline.org is not affiliated to NetSuite in any way. And this is not something that we endorse. But this is something I personally use for reformatting my JSON string.

You can [INAUDIBLE] I find convenient. So anyway, this is the string that you will be getting from your Map stage. As you can see, I've reformatted it here on the right side so we can see the different elements of our JSON string.

So you have three different elements. You have your record type. You have your ID. And also, you have your values. Remember, this is different from the value, context.value object. So this is a values inside your context.value.

So if you will be writing this as a Javascript object, that would be context.value.values if you want to extract that from the value. Now, if you want to get the record type of your search, you have your recordType property. You also have your id that points to your search.

And then inside your values, that would be your search columns, whatever columns you define. Now, we have two column values that we defined. We have the Entity field, and then we also have your total.

Notice here that my Entity field has two values. If I expand that, I can see here that I have a value, and I have a text. Now, value and text would be similar to your getValue and getText. Value would be pointing to your internal ID, and text would be pointing to whatever is displayed on that field. So in this case, this would be the name of your customer.

Now, for your total, this is a simple free-form text. Actually, it's not a free-form text. It's a currency field. So you can see that this is just one value. So you don't have that value in text. Remember that value and text would only be applicable if you have a dropdown. And dropdowns would be for list and record field, or a list or record fields, OK?

So again, remember the structure. So you have your object. So this would be context.value. Then inside that object, you have three properties-- your recordType, your id, and your values property. Then inside your values property, you have all of the fields. So you can get a total, the name of the customer or department.

So for department, since that's a record field, you would have two values for the value and text. But for email, phone number, you only have this one property. So again, remember that format.

Now, going back here, I have my searchResult. And then for my searchResult, what I want to do is that I want to group this based on the customer. So it could potentially have multiple transactions. Because one customer can potentially have 100 different invoices.

And what I want to do is that I want to group all of those invoices per customer so that I can loop through the invoices per customer when I'm totaling it. Normally, if you would be doing this, say, on a scheduled script, what you're going to do is that on your search, you're going to create the search. You're going to run the search. You're going to loop through the different results. You're going to add in a For loop, and then you are going to extract your customer information and also your total information.

Then after that, you're going to process that by looping through your results again. So you have a loop within a loop. And then that loop would be combining your different invoice records for that customer. Now, with the map/reduce script, the only thing that I need to do is I need to tell the system that, OK, I will be grouping this based on a key-and-value pair.

And that's what the Map stage is for. You can use that the group key-and-value pairs. You can also use this to process your information if you don't want to group-- like, OK, I'm going to stop at the Map stage, and I don't need to go to the Reduce stage.

But in our case, I'm going to use the Map stage and also the Reduce stage. So for the Map stage, I'm going to use that to group my key-and-value pairs. My key would be my customer information. Then my value or my values-- because I can potentially have multiple invoices-- that would be my total.

Anyway, so here, if I want to get the customer name-- so var customerName-- or maybe I'll just call this Customer. It'll be faster that way. Say I have Customer here. So that'll be searchResult. That will be my context.value, searchResult and then values. Remember, that's where we're extracting the information.

And for my values, I have my entity. And I can use either Text or I can use Value. If I want to group this by internal ID, I'm going to use Value. But for this one, I think I want to see the name. So I'm going to use Text. Then for the total-- so here, searchResult.values.total. So I don't need to use value or text because I only have one element here. And that's it.

So if I want to group this by key-and-value pair, all I need to do is just use context and then write. That's the method that I need to use, context.write, and then say key. So the key that I'm going to use would be my customer. And the value that I'm going to set it to-- value, that would be total. That's it.

So as you can see, I am saving a lot of operations. I'm not parsing the data. I'm not looping through the data. I'm not grouping the data. Because everything would be done by the system for me. And again, the system is just-- or the way of creating a map/reduce script is a little bit different because we're skipping a lot of steps.

And skipping steps is good. Because that means that we need to just concentrate on one specific thing. And on our Map stage, at least for this particular script, I'm concentrating on grouping my key-and-value pair. I don't need to think about anything else.

I just need to think about, what is my key? And what is my value? Now I have my key and value. I let the system handle that. And that's it. So as you can see, we've skipped a lot of things. And it's just very convenient to use the Map/Reduce stage.

The thought process is a little bit different. That's the thing that you need to be aware of. But once you get the hang of using the Map/Reduce stage or the map/reduce script, it's just very, very easy. So anyway, once I have my key-and-value pair here, I'm going to move down to my Reduce stage. And then on my Reduce stage, what's going to happen here is that I will be given the key and values that were grouped in the Map [? stage. ?]

So it would have one key for execution of my Reduce stage. And then my values could be one value. Or it could be multiple values depending on how many records were grouped in my Map stage. OK. So again, in this stage, if I'm going to count the number of values, I'm going to add another loop inside my statement if here I'm going to use a scheduled script.

Now, just imagine you have a loop for your parsing of your information. You have a loop for grouping your key-and-value pairs. And then you have another loop for totaling your values, just a loop within a loop within a loop. Just imagine how slow that would be if I'm doing this on something like a scheduled script.

But anyway, on my Reduce stage, again, all I need to do here is just do my totals. So I have var total. And then after that, I'll add in my For loop. In my For loop, by the way, what I'm doing is that I'm just using the FOREACH format. So the FOREACH format would be var i. And this would be my index or my property list, and then the in keyword.

And then here, I have my variable or my array list. And that array list would be context.values. So this is a little bit different from your Map stage where you have value. So that's one singular value. But here, you can have potential multiple values for your key-value pairs. So you have one key and then multiple values. So that's why I'm using values.

And then here, if I want the total, I'm just going to use that total value. Nothing fancy here. I'm just going to get the information from my Values object. So for this one, I want to make sure that this is a valid number. So I'm going to use parseFloat. Then I'm just going to extract that for my values, so context.values. So that's my array. And then I put in my index number.

You can use a FOR EACH loop like this one. So this is a FOR EACH equivalent for JavaScript. Or you can just use a regular For loop. It's up to you. So I just find it convenient to use a FOR EACH loop in this particular case because I need to loop through all of the different elements. But if you're more comfortable in using the standard FOR loop format, then go ahead and use that.

So anyway, that's that. The next thing that we're going to do here is we're going to perform a debug, so log.debug. And then I'm going to say this is my totals, at least for this particular customer. And then for my customer, I'll add in my Customer key field here. And I'm going to get my customer information from my key, so context.key.

Again, the information that you're going to get in the Reduce stage is a key-value pair, OK? So key would be my customer. And then values would be the values list. In this case, I really don't need the values because I've already looped through the values. I only need to get the total.

So I have my key here. And then I add in another line break. And then just have the totals right over here. So Totals, and then I have Total-- maybe just Total, not Totals. That works. OK. There we go.

So that's it. So as you can see, with the script that we're doing, where we are totaling all of the invoices per customer, how many lines have we used? One particular line for creating a search, then a few lines for grouping the key-and-value pairs?

In fact, for this one, you can do this in, like, two lines. I didn't even have to use a separate variable for my customers. I can just put that here as my key. And then my value would be this. So here, two lines-- so essential, I had three lines.

Then for this one, I have, essentially, four lines or five. So this is three, so there's five different lines. And just imagine the number of lines that I saved. And if you're doing this on a scheduled script, you have to think about everything.

So again, we're just processing that per stage. I know I just keep repeating this because the mindset is a little bit different. So for the getInputData stage, you just need to get your data. For the Map stage, you just need to use your key-value pairs or map your value to your key.

And then for Reduce stage, you're going to process the information finally, where you can do whatever you want. Now, in this case, I'm just doing a total. So that's what I have. OK. So that's it. So that's how you use a map/reduce script.

And then lastly, on our Summarize stage, there are a lot of things that we can do here. We can perform-- get some information for reporting-- for example, the number of queues we use. So here, I can just say log.audit. So maybe this is something that I want to keep displaying on my logs. So I'll say this would be the number of queues. I want to log that.

And then here, I can get that on my Summary object. So this is essentially the context object named differently. So that's my summary.concurrency. So that's the value or the property that I can use to get the concurrency.

Now, for error handling, this would depend on the object. So if I go to the Help Center, and if I open our definition for my Map/Reduce stage-- let me just open this. Let me go to my Map/Reduce stage definition. The objects that we need here would be these objects. So you have three different objects.

So you have your inputSummary. So your inputSummary stage is an example of what you can do to get information about an error that can potentially have happened on your input stage. So for this one, since the getInputData stage only happens once, or it's going to be triggered once, you could potentially just have one error message.

Now, for the Map and Reduce stage, since that would be triggered multiple times depending on the number of data that you're processing, then you have to make sure that you loop through the results. For example, here, I have my Summary. And then I have my mapSummary object.

Now, inside my mapSummary object, I have my keys if I want to get that information. So if you want to know what keys is about, it's here, OK? And then I also have my errors. And this is what we're interested in. Because we want to know the different errors that has happened if there are any errors in our script.

So you have errors here. And then I have my Iterator. So that's what I'm going to use to loop through my function using the Each method. Now, this Each method would be the same Each method that you're going to use if you are going to get a result set object on your search.

So if you created a search or loaded a search, and instead of using pageRange, you decided to use a result set, then you're going to use the Each method. So here, in my Each method, I have two values. And again, this is just going to loop through all of the error messages. And then now, in this case, we're just logging that on the page. If you want to do something else with the error, it's really up to you.

Now, the key-- by the way, one thing that you have to remember here is that you need to return a Boolean value. And typically, you just want to return True. By returning the True value, that means that you want the process the next result.

If you don't return a Boolean value, or if you return False, what's going to happen is that it's going to stop the execution at the first run. So it's not going to loop through all of the results. And you don't want that to happen. If you want to get the result or the error messages, you want to display all of the error messages, not just the first one.

Same thing for my reduceSummary-- again, this is very similar. The structure is that you have your reduceSummary. You have your errors. You have your Iterator object. And then you also have your Each function. So let's go ahead and process that error message. And I do recommend that you do this on all of your map/reduce scripts.

So for this one, let's log the error message. Again, this is for my getInputSummary stage. So here, I'll just say this is my input error. Then my input error would be from my summary.InputSummary. Then I have error. So that's something you might have.

One other thing that I have here is my summary. And then remember, I have my mapSummary object. And then inside my mapSummary object, I have my Errors object, where I would be getting my iterators.

So if you're not familiar with the different objects that you can get, of course, you can always just go back to the Help Center. So as you can see here, I have my mapSummary or reduceSummary, errors, than iterator. Then you have your Each function.

Then you have two values, your code, and also your message. So here, it's displayed as key and error. But essentially, that is your code, whatever error code you have, and then the message for that particular error code.

So anyway, I have my Iterators object here. And then I'll just use Each. Then for my Each, I have my Return function. So for my Return function, I have two values. I have code, and I also have message. Then for my code and message, I'm going to process that by just logging the error, log.error.

Then here, I'll just say this would be my Map Error. And then for the details of my log, I'll just say code-- maybe I'll just put in both. I'll just say-- or maybe what I can do is that I can do something like this. I think that [INAUDIBLE]. There we go.

OK. And then of course, return true. Now, again, it really is up to you in how you want to log your error. You can send an email if there's an error. Because that's something you can do. Or for that Map/Reduce stage, since you will be processing a lot of records in bulk, I really wouldn't recommend that you would send an email. Because if 1,000 records failed, then you can potentially have 1,000 different error messages. That would be really annoying. So just go ahead and log your information.

So anyway, since the Map stage is similar to your Reduce stage-- so I'm just going to change this to Reduce. I'll just copy that, change this to Reduce, and then change that Map Error to Reduce Error. And that's it.

OK. So that's how you would handle your data. And again, Summarize can be used for reporting or error handling. And make sure that you do put in some error-handling code on your Map/Reduce script. So that is pretty much it.

Let me upload this and see how this executes. And let me go back to my NetSuite account here. And let me build my map/reduce script. So building this would be pretty much the same as my scheduled script, so Create Record. And then for my Map stage, I'll just call this SuiteDreams MR. And then I'll just call this Invoice Totals and make that simple, \_sdr\_mr\_invoice\_totals.

And similar to what I did for my scheduled script, I'm not going to use my Quick Deployment subtab. But instead, I'm going to hover over my script record and click on Save and Deploy. So we can look at the different settings that we have available for the map/reduce.

So anyway, here on my map/reduce, for the ID, again, I'll just use the same ID, \_mr\_invoice\_totals. Options are pretty much the same. So you have Testing, Not Scheduled, and Scheduled. If I scroll down, you can see here I have Single Event, Daily Event. We've already seen this. So it's pretty much the same-- Weekly Event, Monthly or Yearly.

So for this one, I'll just set that to be a Single Event. Now, there are other options that would be unique to a map/reduce script. And the one option would be your queue. Now, you do have a Queue field on a scheduled script. But if you remember, that's a dropdown.

Now, for a map/reduce script, since this is running on-- or since this can be executed based on distributed processing, you can select multiple scripts or multiple queues for your script. That means that you can potentially use all 15-- if you have a Level 3 SuiteCloud Plus upgrade-- on your execution. And if you do use all 15 queues, that means that you can possibly-- because, for example, instead of having one execution for your script, you can potentially execute 15 queues at the same time. And that will be so much faster than just executing one. So you can see the benefit of using a map/reduce script over a scheduled script is just really immense. That's multiplying your execution to 15 times if you are going to use that.

Now, in this case-- let me just try to look for it. Before we try to execute our script, let me look for our invoices. So for my invoice, I almost clicked on that. But let me look at the number of invoices that we have here in our account.

So for my invoices, I have a total of about-- let's just wait for this to load a little bit. So I'm on the Debugger domain. So it's a little bit slower. So here, I have a total of 8,696, so just about almost 9,000 records. So that's the amount of records that we are going to process.

Remember, we are going to look for all of them, group them by customer, loop through the results, and then do the totals. That's what we're going to do, OK? So as you can see, we are processing quite a bit of information. But instead of using all 15 queues-- and I can do that by selecting Select All-- what I'm going to do is that I'm going to select just five.

So one, two, five, that's it. And we're going to see how fast processing would be if I'm using a map/reduce script to process just about 9,000 records using just five queues. And just imagine what we have for 15 queues, OK?

Now, also, one thing that we have here, another option that we have is the Queue All Stages at Once. So you have an option here that would allow you to perform your processing without any gaps. Because what can potentially happen if you are creating a map/reduce script is that since each script is executed per stage-- for example, for each stage that's executed [INAUDIBLE] separate script, you can have other scripts that can potentially go in between.

So for example, you have a getInputData stage. And then you have a scheduled script in between. And then you have your Map stage. Or maybe you have, say, your Map stage. And then you have several scheduled scripts in between, then another Map stage, and then a scheduled script, then a Map stage. And that's going to break it down.

Now, that's going to slow your execution down because it's waiting for other scripts. What if that scheduled script would execute for 30 minutes? So your script would be waiting for that other script to finish. So we don't want that to happen.

So if you don't want any other script to go in between your execution, what you can do is that you can enable this option, Queue All Stages at Once. That means that it would treat your script as one unit. It's not going to execute as one like a scheduled script, one after the other.

But it's going to push that into the queue all at once so that if there's any other scripts that were scheduled in between, it's not going to go inside the execution but after your Reduce stage, OK? So this would be very convenient. Again, just make sure that you enable this option. That would speed your processing up.

Now, if you only have one queue and you think that your Map stage would be processing so much data that it could potentially delay the execution of your other scripts, then you can just disable this. So by unchecking this option, you're still allowing other scripts that may be of higher priority to execute. You also have an option here to yield your script after a certain number of minutes-- again, also convenient if you are dealing with a singular queue so you don't hog all of the resources in your script.

So for example, if you have multiple scripts that are running on one queue, and if your Map/Reduce stage is running for a very long time, that will delay other scripts. So here, if your script has been running for, say, 15 minutes-- so maybe we can just shorten this down to 15 minutes-- I'm going to say that, OK, let the other queues pass. And then I'm going to go to the back. And then rerun once the other scripts have executed.

But again, if you don't have anything queued up for that singular queue, then it's going to stop the execution, try to go back. But since there's no other scripts to go back to, it'll just continue with the execution. So this is not really going to slow things down if you don't have any other scripts. But if you do have any other scripts, this would be a good way to allow them to pass.

Now, another option that we have here would be the buffer size. It's something that you also want to consider, especially if you are dealing with record creation. The buffer size tells us how many sets of data you will be processing at the time.

So for example, if I set my buffer size to 64, that means that I will be processing using the fastest setting that I have, which means I will be processing 64 data sets per execution. But that is a good thing if you want to speed things up. So you just set that at the highest.

But the drawback of using the highest setting is that if something went wrong on the server-- for example, the server failed for any reason or there's something wrong with the execution, what will happen is that the server would restart the execution based on the last set. So for example, if you're using 64, and then your data broke at about, say, 35, it would be restarted by the system. So you're going to potentially have 35 duplicates if you are creating a record.

Now, in our case, it's not really a big deal because we're not really creating any records. So it's not going to mess up our database because we're just counting the totals, OK? So using 64 would be pretty safe. But if I am loading a record, and creating a record, or doing anything that would affect your database, then you want to think of a balance. What is a good number you can use?

For example, instead of using 64, maybe you can just use 16. So 16 would-- a very good number. Or if you're processing a lot of data on your script, like maybe you're creating 10 different records on your map/reduce script on one execution, you want to make sure you get that lower.

So maybe you can just put that 2. So just in case it breaks, it's still OK. Because it's not going to create that many duplicates. And in fact, if it's 2, the likelihood of that breaking in between, it might just already process everything before it breaks.

So again, this is pretty safe. So it's really up to you on how you want to manage this. But just do remember that just in case there's something wrong with the server or there's a breakage in the web application somewhere, then you can potentially have multiple duplicates if you are going to use a higher buffer size.

For our case, I'm going to use just a very high buffer size. Because again, we're not doing anything on the database. And then here, I'm just going to save this and then edit this so that I can run my script. So for this one, I'm going to click on the Save and Execute, similar to what I did in my scheduled script. And similar to what I have on my scheduled script, you can see that I have a scheduled script status.

And here, on your map/reduce script status, the page is, again, similar. But you have other options, such as the queues. So the system will tell you how many queues you're currently using, the details. So if you click on Details, it'll give you whatever is happening on the queue.

You can also have this option. Or you can go back to this page by going to Customization, Scripting, and then go to Map/Reduce Script Status. So again, that's Customization, Scripting, then Map/Reduce Script Status.

Now, let's try to look at the details. And let's see how our page is executing. And as you can see-- oh, it looks like we have a failure. So maybe I just misspelled one of the codes there. But as you can see, everything completed. So we have our values here. So you have your Map stage. That already completed. And you also have your Reduce stage here.

And then you have a Summary or a Shuffle stage. And that would be the system-generated stage where it would be grouping your key-and-value pairs and then passing that off to your Reduce stage. Then here, you have your summary.

But anyway, let's try to just take a look at where our Summarize stage failed. We go to my execution logs. Oh, OK. So yeah, it looks like for this one, I just had some problems with my Iterator object. So I just have to fix this.

But this is-- it didn't really fail, the execution of my script. But it failed on my error handling. So there's something wrong in my iterator. So I'll just try to look for that later. But anyway, as you can see here, the execution of my script was just right because it reached everything. It was able to total all of my customer invoices.

So you can see here, I have my information on this particular customer. And I have my total here. And then I have another total here, and another total. And then it went into my Summary stage. So here, it told me the number of queues that I used. Input Error, which is empty because I didn't really have an error.

Oh, and that's something that I forgot to add, just add in a condition so I know if I have an error. Now, for this one, since I don't have errors on my mapSummary because it executed properly-- so this would be empty. So yeah, that's why I wasn't able to get my Iterator object. So it's as simple as that.

But anyway, those are things that you can do. And that's how you would process information using your map/reduce script. Again, the map/reduce script is a very powerful script type. And I hope you do try to use that on your bulk processing.

Try to practice this one. Because again, the mindset is different from the regular type of script. But once you get the hang of using a map/reduce script, it's very easy. It's very convenient and also very powerful.

So moving on here, just a few things to remember as you're dealing with the Map stage. And I hope you're really excited using the map/reduce script. So first off, your getInputData stage. So here, you have the stage where you prepare your data. You can use the data from a saved search.

You can use an object reference which points to a saved search. You can have an array if you have a series of arrays. Or you can have an object just in case you have JSON object that you want to process. Once you move from the getInput stage to the Map stage, the Map stage is processing your data. And you can use that to finalize your data or group your data into key-value pairs.

Now, remember that you only need one or the other of the Map or Reduce stage. If you already have a Map stage, then the Reduce stage is optional. Or if you already have the Reduce stage, then the Map stage is optional. But it doesn't mean that you can only use one or the other. You can actually use both, which is what we did.

So after the Map stage, you have the Reduce stage, where you finalize your data. You do whatever you want with your key-value pairs. And then lastly, here, we have our Summarize stage, where you can add in additional information and also error handling.

Now, comparing map/reduce script and the scheduled script, because the nature of map/reduce-- where you are allowed to process multiple data or large amounts of data at the time, if you have multiple queues, then that allows you to process more data quicker. So if you have lots of data to process, consider using the map/reduce script.

Now, if you are going to use a scheduled script, try to use that with smaller data sets where you think that it's not going to really execute a lot of information. So maybe you just want to throw processing from your user event script so that your user would not be waiting. So you can just [? processes ?] that in a scheduled script. But if you're dealing with really bulk data, then you want to use a map/reduce script.

Now it's your turn. Please complete the following activity. This includes Module 9, Bulk Processing, Part Two.