jQuery in APEX

An Introduction for Database Developers (Tutorial Transcript)

SkillBuilders.com/APEX

Dan McGhan, Oracle ACE

June 14, 2011

Watch and listen to the tutorial at skillbuilders.com/apex-jquery-introduction-tutorial

Welcome, everybody. I'm glad you could join us today. Let's talk a little bit about what we'll be doing today. I'm going to introduce you to jQuery in APEX, but I'm not going to do it as I would to a new group of new programmers because you're not. You're database developers, and that gives you an edge over new developers.

So I'm going to leverage that fact and introduce you to jQuery in APEX through a series of analogies that relate critical concepts back to what you already know in the database. I think you'll be surprised at how similar these technologies can actually be.

The first analogy: A webpage is like a dataset. As database developers, we're used to data, all kinds of it, but we rarely think of HTML webpages as just another dataset. We see something like this and we flip out. Why? Because the data doesn't look quite right. It's foreign and out of our comfort zone.

When we see data that looks like this, we're happy. It takes us very little time to understand it. In fact, let's prove that and take a moment to get to know this data a little bit better. We start with a header row. Basically, it describes the data that we're looking at. The first column is ID and the second is Parent ID, so we know that there's some kind of hierarchical relationship amongst the rows in this particular dataset.

The next column is Element and it looks like every single row here has an element. You could say it's required. So this is really what we're looking at, elements that are hierarchically arranged, and then there's a number of other attributes which seem to be optional.

Okay, so that's the basic dataset that we're looking at. Easy to understand, right? Guess what? It's the same dataset we saw on the HTML page before. If I rearrange these rows using the element column as the main attribute and then use these parent-child relationships to lay out the elements hierarchically and finally, we take these other attributes and put them in line with the element, it would look like this. It's the same but different, just another dataset, and with some practice, you'll feel right at home with it. We can see here the same kind of relationships that we saw on the page before.

If I show you this one last time, we have HTML is the first row. Next, we have Head and you

can see that Head is beneath HTML. Title is linked up to Head. Going down a little bit, we have Form -- I'm sorry -- Body, and Body links back to HTML. When we look at it here, it's actually easier to see the relationship.

When people are new to web development and they hear the term DOM or document object model, they can be a bit intimidated, but you're already used to working with objects in the database, and it's really very similar. The database object that's most like the DOM is a table.

Once again with our dataset, if we take our dataset here and move it into the database, we get something like this, a table. It's through tables and SQL that we select rows, perform inserts, updates and deletes and even add new columns or remove columns.

The same is true when we take our HTML webpage and move it into a browser. The page comes to life. This is how our browsers see the webpage, and through the DOM and JavaScript, we can do the same things to our HTML webpage. We can select elements, insert, update and delete them, even add and remove attributes.

So, when you do database development, it's important that you use the right tools, and many of you are familiar with tools like Toad or SQL Developer. We use these tools for all kinds of important functionality. A lot of that is listed here, things like debugging code, manipulating objects. It's really important to use the right tools. Some of you may still be using SQL Plus as your main development IDE. I think you get the point.

In the web world, things are no different. We need to use the right tools for the job, like in Firebug to the IDEs we use in the database world. Firebug is a plug-in for the Firefox web browser. So that browser is a prerequisite, but with it, with the combination, we get all kinds of functionality to help us work with our webpages.

In fact, a lot of the functionality we see listed her is the same that we saw for the SQL IDEs, the ability to debug code, manipulate elements or, as they're often referred to, objects. If you're not doing web development with these tools, well, why would you do that to yourself?

Let's take a moment to explore some of Firebug and get an idea of what it's capable of. Here's the webpage we've seen a few times now. Behind the scenes, it is the content we saw in the HTML webpage and of course, as you can see, I'm in Firefox. Down below, you can tell I have Firebug installed. You see this little bug, and when I click the bug, it essentially turns Firebug on.

The first thing you'll notice with Firebug is that we have a couple of tools over here on the left. Next to those, we have a number of tabs. These tabs can be disabled by default. When you first start working with the product, you'll need to enable them and the way you do that, you just click on the tab and you'll see these arrows here and you'll see the enable and disable. So, if they are disabled and you go to them, just click on that and you can enable and disable.

The first tab we'll take a look at is the console. In most IDEs, like SQL Dev, people are used to using dbms_output.put_line. It allows us to see output from that buffer. Well, in JavaScript,

we have something similar. We have what's known as the console. Not all browsers support console yet. Most of the newer ones do. Firefox does, and Firebug allows us to see it.

This is the equivalent of dbms_output.put_line. We'll do another "Hello World" example. You can come down here and hit run, and you can see the code that's executed and its output. Just ignore that return value.

So a lot of the same stuff you're already used to, and this particular technology can be used for warnings, debug info and things like that. If you don't see -- after you install Firebug into Firefox, if you don't see this window, it's closed by default. Just look in the lower right-hand corner and click that arrow to open it up. Otherwise, it's available at the bottom as a single line.

The next tab I want to show you is the HTML tab. Do you remember the old days of the web -- let me shut this down for a sec. What would we do if we wanted to see the HTML behind this page? We would right click somewhere in the content and we'd go over to "View page source." For this particular page, because it's so simple, it actually still works. But most webpages today are ten times this size easy, typically many more times that size. So it's very ineffective to continue that technique. Firebug has the right tools for today's webpages.

Now we need to use this HTML browser, and you can think of this kind of like the various object browsers in your IDEs. Only here, we're working with the webpage. Notice it shows us the content, and as I hover over various elements, they're highlighted above. Now we can scroll down and get more granular. Likewise, I can do the reverse. Using the "selector" tool, I can hover over the content above and it will find it below.

Now, once you find what you're looking for, if you click it, it locks it in, and once an element is selected here, you then have access to these tabs on the right which basically describe that particular object or element.

Now, I'm not using any style attributes here but if I were, you could see them. You can get layout information such as padding, border and margin, figure out why things are pushed in or not, and then we have direct access to the DOM for this particular element. So very powerful, and if you're looking for something specific, say, "jQuery is fun," you can type it in here and it will find it for you.

The next tab is CSS. I'm not using any external CSS style sheets, but if I were, I could see them here. I could choose from the various style sheets, view them and see why things look the way they do. Likewise, we have a script tab. I have to refresh the page to see it, but I am using a script file in this one. This is how it works: You can select that file and then see the contents. This one is compressed, so it doesn't do us much good, but it often works better for CSS.

The next tab is the DOM. It's similar to the DOM tab we saw before when I selected a particular element, but this is in a more general sense, the entire DOM tree. Then finally, we have NET, and NET can be used to help gauge performance-related issues with regard to the page loading essentially, and you can see that bringing in external scripts adds to the overall page weight and therefore requires more time to load, and you can get a breakdown of that.

Again, this is a really simple page. So if you do this with the pages you'll be working with on a regular basis, you're probably going to see a lot more content here. Images make a big difference, and you can figure out perhaps what's taking a page so long to load. Maybe images are not caching as they should be. So that's Firebug in a nutshell.
In the database world, a lot of what we do starts with and depends on queries. With jQuery, it's no different. Pretty much everything starts with a query, but in jQuery, we call them "selectors."

In databases, a query gives us a result set. Typically, it will contain one or more rows of data. jQuery selectors, on the other hand, return one or more jQuery objects and not rows of data. We get jQuery objects. We'll learn more about these objects as we go, but first let's take a look at some of these selectors and starting here with example one, if you can think back to our HTML table which, as you saw, I did, in fact, load that into database. We can run a select count star from HTML table and of course we'd expect to see the number of rows that were in that table, all of them. We can do the same thing with jQuery, only we're selecting them from the page.

Do you see the dollar sign? I like to sort of read that like select or an ASP for select. What goes in between the parentheses is essentially your where clause. In this case, I'm saying select star or everything. It's a lot like the count star above and rather than use count, we have to access a length attribute.

Going back to our webpage, if you took a look here and manually counted all of these elements, we'd get a count of 33. That's how many rows are in that table. Using Firebug and the console, I can put in some jQuery code that shows us this. Here's our select statement. We're selecting "all attributes" -- or "all elements" rather -- and displaying the link. I'll run this. We get a count of 34, so relatively straightforward.

Next example: This time, we're adding a where clause to our select statement. So, rather than seeing all of the rows, I only want to see the rows where the element is an input. Now, for those of you that are familiar with HTML, when you look at that webpage, you may think you see two inputs, one and two. The fact of the matter is that there are three here.

I'm going to right click on this button. I'm going to go down to "inspect element," again, a tool from Firebug. When I hover over this one, look above and see what's highlighted. This is a button element. Now, the one to its side is actually another input element. It's a type of submit, but it is an input element and, as you can see, they look identical. Okay, so there are, in fact, three inputs here.

So, if we ran that query from SQL, we'd get a total of three rows returned and below that, you can see the equivalent in jQuery. We're saying select where element is input. So, when you're targeting a specific type of element, this is how it looks. You actually just put the name of the element between parentheses. Then, of course we're just calling dot-length* again. Bring in that jQuery code. So here we're selecting just the inputs. Looking for the length, we run this and we get back three. So far so good.

In this example, I've made a slight adjustment to the SQL. If you remember that dataset, ID and

Parent ID were actually numeric. In the web world, it's a bit different. If there is an element in an HTML webpage that has an ID attribute, it is like a primary key in the database. It should be unique across the entire webpage, but they're not numeric, at least not usually. They're just static text.

What I've done, I've moved the SQL around a bit to make this more clear but the idea is the same. We're targeting a specific row by its primary key. In this case, we want to see the row where the ID is wwwvflowform*, and that looks a bit confusing. The reason for that particular ID is because I pulled the content or I should say used APEX as the basis for this particular page, and that's what the main form is called.

In the jQuery below, we see how we can target the attribute of ID. It's with that hash before the ID value. So we're not targeting an element by its type now. We're actually going for an element's attribute. This particular attribute is ID.

Go back to the page. I'm going to start here, right click and go to – well, I can just to HTML. We'll scroll to the top. We see the form here. It sort of envelopes everything else. Here's that ID. We bring in the jQuery code to select it. We run it. Now we're down to one. So, as you can see in the first two examples, we got back multiple rows or multiple jQuery objects. In the last example, we got a single jQuery object.

Here's another example: This time, rather than doing a count as we've been doing, it's a more practical example. In SQL, we might select an attribute like, in this case, action. We're selecting the action column from the table, again targeting that specific row. In jQuery, I'm using the same selector we saw before. This time, I'm using the attribute method to get the value of action.

Let me clear these out. In looking again at the page, here's that form. Here's the attribute action. It's flow.accept, wwwvflow.accept. Go back to the console, bring in that jQuery code. We're selecting that using attribute to access action. We run it and of course we see the value. So the same types of select operations that we have access to in SQL we also have available to us in jQuery.

Let's take a closer look at attribute. Attribute, in addition to retrieving values, can also be used to update them. It's a lot like an update statement or the SET clause in an update statement. Here, we're doing an update in SQL to the HTML table, setting the size attribute for a particular row, and we're targeting one row because of course we're referencing ID.

In jQuery, we already know the selector for IDs, and we're targeting the job title and using the attribute, this time passing in two parameters. When you pass in a single parameter, it returns the value of that particular attribute. In this case, two parameters, we're setting the value.

Here is the job title. We can see the input here. Here is the ID and here's the size attribute. It's currently set to ten. The one above it, this one, is set to 30. So, if we set them both to 30, they should be the same size. Let's go to the console, bring in the jQuery code, our select attributed size to 30, and look above when I click run. Watch that job title field. There we have it, same

size.

So that's interesting. The page is not static. It's dynamic, and that's made available to the DOM, just like a table is in the database.

Of course, as we work with data in SQL, we often have to insert new data. We do that via the "insert statement" and we can see an example of inserting a new row into our HTML table here. One thing I want you to notice is the ID and Parent ID. What we're essentially doing here, if you were to look into that table where you see Parent ID, the value here is five, okay, and if you look at that particular row, it's an H1 element. So what we're doing is adding an H3 element essentially beneath it and setting its text value to "Why not give it a try?"

Below that, you can see the jQuery equivalent. We start with the H1 element and on this particular page, there's only one H1 element, so I know I'm just going to get that one. And "after" works a lot like "insert." Frankly, with jQuery, there are other methods we could use for this depending on exactly what we needed, but "after" is a good example here. So we're saying after the H1, add and you can see the H3 along with its text value.

Here's the H1. I'll inspect that. We can see it here. So what I'm trying to do is to add an H3 element directly below it. Let's take a look. jQuery code. We hit "run," look above, and there we have it. So we can add elements to the DOM that didn't exist when the page first loaded. You can even now use Firebug to see them.

Of course, if we can do inserts, we can do deletes, right? In the SQL above, I'm deleting the job title row from the table. jQuery below, equally as simple. We already know how to select an element by its ID and then we simply call the remove method. Note that we do need the parenthesis on the end. Again, here's the job title. We bring in the jQuery code, select that element and hit "remove." Look above, I hit "run" and it's gone, completely removed from the DOM.

What I'm going to do now is sort of the equivalent of a rollback operation in, say, SQL Developer. I've made some changes to the DOM at this point. We've added an element. We've removed one. If I refresh the page, we're going to go back to that source HTML file and bring things back to the way they were. So I'll just hit "Refresh," and then we have it.

That's a lot of code, huh? Perhaps I should take a moment first to explain what it is that we're trying to do here. So I want to hide a row. If we look at the content here, we'll see that there's an HTML table and beneath that table is an HTML body and within the body we have one, two, three rows, and what I want to do is remove this row in the middle. The problem is that -- actually, I don't want to remove it. I just want to hide it.

The problem is that I don't have a good way to select it. Notice that that particular table row does not have an ID or any other attribute I could use to get at it. So the only way for me to get it really is to use relationships that it has with other elements on the page or, in the case of our dataset, other elements in the table.

Admittedly, the example is a bit contrived, but follow along. So we start down here. I'm selecting the ID from the table where the element is table row, so I'm getting those three, and I'm using the relationship with connect by to basically traverse the tree and I'm scraping off the bottom there the bottom two rows.

So we get two rows from the first set here and then we scrape off of that, targeting the -- getting rid of, I should say, the tops and now we're left with the bottom -- I'm sorry -- the middle. That gives us the ID we're looking for and at that time, we set its style to "display none."

If you're familiar with connect by in SQL, you don't need this example to understand what it does. It basically allows us to traverse the hierarchical relationships in our databases. It's far more powerful than the tools that people have in other database systems, very easy to use.

Here's the equivalent in jQuery. In my opinion, it's even easier. What we see here is an example of what's known as chaining. We start with the table and there's only one table in that page, so we know we're going to get one jQuery object. It returns a jQuery object. Then with that, we can all the children method. The children, of course, gives us the table body, but it returns it as a jQuery object. So then we can call the children again which of course gives us the table rows and returns those rows as a jQuery object.

Then we can call EQ, and EQ is a way to filter a result set of jQuery objects, but it's a zero-based index. So, if you want the first row, you say "zero." If you want the second, you say "one," and that's the one I want, and once we have that, we can use the CSS method to display attribute "equal to none."

Here's our page again. This is the row I want to hide. Go to the console, bring in the jQuery code and hit run, look above, and it's gone. The fact of the matter is in this particular case, it's not actually gone. We've only hidden it.

Before any of you call me out, I know that this particular trigger would not compile. Again, it's a contrived example, but I think it illustrates the point. What we're doing with this trigger is basically looking to see when the value of a select elements changes. Now, if you remember that webpage, there's only one select element. So, if it changes, then what we're saying is if the new value is active, then update the table and set disabled to known; in other words, remove the disabled attribute for the job title row. Here we have the opposite. If it's not active, then set disabled equal to disable for that job title. It's pretty simple.

Here's the jQuery equivalent. Essentially, what we're doing is creating what's known as an event binder here. We're starting with selecting status which of course is our driver, and we're saying if it changes, then call this function. The function does not have a name, but its body is simple enough. If this value is active, then find the job title attribute and remove -- I'm sorry --find the job title element and remove the attribute disabled. * find job title and set disabled to disable. So, if it's not active, we'll disable job title. If it's active, we'll make it available.

Refresh this. Okay, so here's job title and notice if I toggle to inactive now, I can still access both elements in active, no change. Let's put a trigger on the page. Here's the jQuery code, just

like we saw. We're starting with status and on change, we're calling this function. I'll run this, and nothing happened because all we've done is set up the trigger. The trigger is now waiting, so when I change status from active to inactive, now I cannot access this element. I can access this one, not this one. We change it back to active. Ah, now I can get in there.

One last change to this page. We have a reset-form button here. Right now, it doesn't do anything. If I want it to do something, I could use JavaScript, jQuery. In this case, I'm targeting the button. And remember that this one's a button, this one's action input. So when I target button, I'm only going to get one. We'll call this function if it's clicked. Call this function, and I'm getting the select and setting its value to nothing and then finding the input. If you remember, there are three, but I don't want all three. I only want the inputs where the ID does not equal submit form. In other words, I want this input and this input, and I'm doing the same thing, setting the value to null or an empty string.

So let's see how this works if I type some stuff in here and in here and have this on inactive. Right now, I hit reset form; nothing. We add this event binder. We click the button. It clears it all out.

As you can see, there are a lot of concepts in the web world that are very similar to what you're already doing in the database world. If you'd like to learn more, I have a couple of next steps listed here, starting with dynamic actions. For those of you that are already using APEX, hopefully you've heard of dynamic actions at this point. Basically, what they're doing is taking the kind of code you saw me putting in the webpage and making it declarative.

I'll show you a simple example. I'm in a 4.0 APEX instance here and I'm showing you the sample application. This is an application used for learning APEX. If we go to the product tab, drill down on a product, we'll see a form page, and what I want to do here is some of the stuff we were doing in the previous demo page. I want to basically hide some elements or maybe disable them if something else on the page changes. In this case, if the product available is changed to no, maybe I want to disable the list price. It sounds simple enough.

Let's see how we can do this with dynamic actions. Go to "edit the page." The driver in this case is the list price. So I'll right click that and I'll go to "Create dynamic action." APEX can handle a number of dynamic actions. The standard are the easy ones to start with and because I started by right clicking the item -- some of these things are set correctly already -- if the list price is equal to, let's say, no; in other words – did I go with list price? I meant product available. Sorry about that.

If the product available is equal to no, then I want to do something. So, rather declarative, as you can see. And if it's no, what do I want to do? Let's say if it's not available, I want to disable something, and there's a checkbox here that will create the opposite false action; in other words, enable it if it's not equal to no. Then we have to specify what it is that we want disabled or enabled, and in this case, we're working with the list price item. We'll create that, run the page. It's currently enabled. We switch this to no and it's disabled.

In fact, if you look at this disabled compared with the disabled that I had in my example, this

one's a bit more clear. In addition to actually disabling the element, the style has been changed as well. The background color has been changed to give us a better indication that it's disabled. So it's doing a little bit more for us.

So you may be saying to yourselves at this point, "Wait a second. Why am I even here? If dynamic actions can do all of this for me, why do I need to learn this stuff?" Well, the best analogy for that I can give you is it's a bit like saying, "I'm not going to bother learning SQL because I can use a graphical query building tool to build my SQL for me," and that's true. You can do that, but you're really going to limit what you're capable of, the complexity of the type of queries you can create by not learning SQL.

If you look under the hood, the people that created dynamic actions were well aware of this fact. If we go into the dynamic action, you'll see that jQuery or the ability to work with jQuery was built right in. APEX now includes jQuery and jQuery UI built into the product, and the dynamic action framework has been built to support that.

Another next step I recommend is to go to jQuery.com to start learning more about this tool. Here's the site, jQuery.com. If you look toward the bottom, you'll see some books that are good for learning. I recommend this one written by one of the authors. In addition to the books, if you prefer learning, you know, just through the site, you'll want to go to this documentation part.

We worked a lot with selectors today, so here's the API reference and here's the page on selectors. It's not very well organized. It's actually really hard to understand and read. So I want you to remember this little trick. Look in the upper left-hand corner for the dynamic API browser. Give that a click instead. It takes a little bit to load. If you can remember this URL, API.jQuery.com/browser, you can go directly there. Isn't this a bit more organized?

Let's look at selectors. Okay, it drills us down a bit. Let's learn the basics. Okay. * ID, element, and so on. You could learn about class if you wanted. This is far better than using the regular API documentation.

Before I get to the questions, I just wanted to let you know about a couple of things. We have some upcoming training, a class on HTML CSS in JavaScript starting February 28th. We're going to customize that class for APEX. Then of course we have our regular intro and advanced APEX classes after that. In addition to training, we offer a number of other APEX-based or Oracle-based services, from micro to long-term consulting and full application life-cycle development as well as plug-in development. We have some contact information related here.

Also, if you're using jQuery, I just want to show you on our page, you can just go to APEX -- I'm sorry -- SkillBuilders.com/APEX to get to this page. We have a new plug-ins tab. If you're using plug-ins in APEX, make sure to check out this tab and download some of these plug-ins and we'll have some new plug-ins and updates to existing plug-ins coming out in the very near future.

I'd like to open it up to any questions we may have at this time.

Question: “Is jQuery tied exclusively to Oracle?”

Dan: No. jQuery is an independent JavaScript library used by a number of major companies. I believe a lot of those are listed right on jQuery’s home page. Google, Dell, Bank of America, NBC, so a lot of big companies using it. It has nothing to do with Oracle. The APEX development team was just smart enough to integrate it into APEX.

Question: “What is to micro to long-term consulting full life-cycle development?”

Dan: So basically, a lot of the time, organizations require minimum time investments with regard to consulting. With our micro offering, it can be as little as two hours. So that’s the shortest amount of time we offer and then of course long term, depending on how long it is, we can offer better rates. As an example, I just finished a job up in Nashville. I was on site there for five months. So micro online, you know, X months; onsite, it’s completely up to you.

In terms of full cycle, our full life-cycle development, we work with organizations to develop products for them in APEX from scratch and then continue the maintenance of those products as well. So, even if you’re not an APEX developer but you have a need for an application, we can work with you to build that application from scratch and then help you maintain it going forward.

Question: “What is the difference between jQuery UI and jQuery?”

Dan: You saw me using jQuery today and in terms of visual things that you see in the page, it can do some things, like we hid something and we can then display it. We can even fade it out and fade it in but past that, there’s not a whole much you can do. The UI component here takes things a bit further. Now, these are separate libraries, so if you don't want the additional page weight that would be required to run this additional stuff, you don't need to add it.

In APEX, it’s already been added for us, so you have it at your disposal. The kinds of things it does can be found under the demos and documentation, things like drag-able and droppable elements, accordions, date-pickers, all kinds of things that are very popular in the web, 2.0 stuff we see today. I’ll just show one, a date-picker. So, when we click in here, we get a really nice looking client based date-picker that functions really well.

All right, everybody. Thank you very much.