JEREMY KEPNER: All right, so we're doing lecture 06 in the course today. That's in the-- just remind people it's in the docs directory here. And I'm going to be doing a lot, talking about a particular application, which I think is actually representative of a variety of applications that we do a lot of things, similar statistical properties of a sequence cross correlation.

OK. So diving right in. Just going to give an introduction to this particular problem of genetic sequence analysis from a computational perspective and how the D4M technology can really make it pretty easy to do the kinds of things that people like to do. And then just talk about the pipeline that we implemented to implement this system because, as I've said before, when you're dealing with this type of large data, the D4M technology is one piece. It's often the piece you use to prototype your algorithms. But to be a part of a system, you usually have to stitch together a variety of technologies, databases obviously being an important part of that.

So this is a great chart. This is the relative cost per DNA sequence over time here, over the last 20 years. So we're getting a little cut off here at the bottom of the screen. So I think I'm-- hmm. So you know, this just shows Moore's law, so we all know that that technology has been increasing at an incredible rate. And as we've seen, the cost of DNA sequencing is going down dramatically.

So the first DNA sequence, people nominally say to sequence the first human genome was around a billion dollars. And they're expecting it to be $100 within the next few years. So having your DNA sequenced will become probably a fairly routine medical activity in the next decade.

And so the data generated, you know, typically a human genome will have billions of DNA sequences in it. And that's a lot of data.

What's actually perhaps even more interesting than sequencing your DNA is sequencing the DNA of all the other things that are in you, which is sometimes called the metagenome. So take a swab and not just get the DNA-- your DNA, but also of all the other things that are a

part of you, which can be ten times larger than your DNA. So depending on that, so they now have developed these high volume sequencers. Here's an example of one that I believe can do 600.

AUDIENCE: [INAUDIBLE]

JEREMY KEPNER: OK. No problem. That 600 billion base pairs a day, so like 600 gigabytes of data a day. And this is all data that you want to cross correlate. I mean, it's your-- it's a-- so that's what this-- this is a table top, a table top apparatus here that sells for a few hundred thousand dollars. And they are even getting into portable sequencers that you can plug in with a USB connection into a laptop or something like that. It-- to do more in the field types of things.

So why would you want to do this? I think abstractly to understand all that would be good. Computation plays a huge role because this data is collected. And it's just sort of abstract snippets of DNA, you know? Just even assembling them into a-- just your DNA into a whole process can take a fair amount of computation. And right now, that is actually something that takes a fair amount of time.

And so to give you an example, here's a great use case. This shows, if you recall, in the summer of 2011 there was a virulent E. coli outbreak in Germany. And not to single out the Germans. We've certainly had the same things occur in the United States. And these occur all across the world.

And so, you know, this shows kind of the time course in May. You know, the first cases starting appear. And then those lead to the first deaths. And then it spikes. And then that's just kind of when you hit this peak is when they really identify the outbreak. And then they finally figured out what the-- what it is that's causing people and begin to remediate it.

But, you know, until you kind of really have this portion, people are still getting exposed to the thing, usually before they actually nail it down. There's lots of rumors flying around. All other parts of the food chain are disrupted.

You know, any single time a particular product is implicated, that's hundreds of millions of dollars of lost business as people just basically-- you know, they say it's spinach. Then everyone stops buying spinach for a while. And, oh, it wasn't spinach. Sorry. It was something else.

And so that's-- so there's a dual-- you know, so they started by implicating this, the

cucumbers, but that wasn't quite right. They've then sequenced the stuff. And then they correctly identified it was the sprouts. At least I believe that was the time course of events here. So-- and this is sort of the integrated number of deaths.

And so, you know, the story here is obviously the thing we want to do most is when a person gets sick here or here, wouldn't it be great to sequence them immediately, get that information, know exactly what's causing the problem, and then be able to start testing the food supply channel so that you can make a real impact on the mortality? And then likewise, not have the economic-- you know, obviously the loss of life is the preeminent issue here. But there's also the economic impact, which certainly the people who are in those businesses would want to address.

So as you can see, there was a really sort of a rather long delay here between sort of when the outbreak started and the DNA sequence released. And this was actually a big step forward in the sense that DNA sequencing really did play-- ended up playing a role in this process as opposed to previously where it may not have.

And in the-- and people see that now and they would love to move this earlier. So, you know, and obviously in our business and across, it's not just this type of example. But there's other types of examples where rapid sequencing and identification would be very important. And there are certainly investments being made to try and make that more possible.

So an example of what the processing timeline looks now, I mean, you're basically starting with the human infection. And it could be a natural disease. Obviously in the DOD, they're very concerned about bioweapons as well. So there's the collection of the sample, the preparation, the analysis, and sort of the overall time to actionable data.

And really, it's not to say processing plays everything on this, but if you, as part of a whole system, you could imagine if you could do on-site collection, automatic preparation, and then very quick analysis, you could imagine shorting this cycle down to one day, which would be something that would really make a huge impact.

Some of the other useful sequences, useful-- I'm sorry, roles for DNA sequence matching is quickly comparing two data-- two sets of DNA. Identification, that is, who is it? Analysis of mixtures, you know, what type of things could you determine if someone was related to somebody else? Ancestry analysis, which can be used in disease outbreaks, criminal

investigation, and personal medicine. You know, the set of things is pretty large here.

So I'm not going to explain to you kind of fundamentally what is the algorithm that we use for doing this matching, but we're going to explain it in terms of the mathematics that we've described before, which is these associative arrays, which actually make it very, very easy to describe what's going on here. If I was to describe to you the traditional approaches for how we describe DNA sequence matching, it would actually be-- that would be a whole lecture in itself.

So let me get into that algorithm. And so basically this is it. On one slide is how we do DNA sequencing matching. So we have a reference sequence here. This is something that we know, a database of data that we know. And it consists of a sequence ID and then a whole bunch of what are called base pairs. And this can usually be several hundred long. And you'll have thousands of these, each that are a few hundred, maybe 1,000 base pairs long.

And so the standard approach to this is to take these sequences and break them up into smaller units, which are called words or mers. And a standard number is called a 10mer. So they're basically-- what you would say is you take the first ten letters, and you say, "All right. That's one 10mer." Then you move it over one, and you say that's another 10mer, and so on and so forth.

So you had, if this was a sequence of 400 long, you would have 400 10mers, OK? And then you're obviously multiplying the total data volume by a factor of ten because of this thing. And so for those of us who know signal processing, this is just standard filtering. Nothing new here. Very, very standard type of filtering approach.

So then what we do is for each sequence ID, OK, this forms the row key of an associative array. And each 10mer of that sequence forms a column key of that 10mer. So you can see here each of these rows shows you all the unique 10mers that appeared in that sequence. And then a column is a particular 10mer.

So as we can see, certain 10mers appear very commonly. And some appear in a not so common way. So that gives us an associative array, which is a sparse matrix where we have sequence ID by 10mer. And then we do the same thing with the collection. So we've collected, in this case, some unknown bacteria sample. And we have a similar set of sequences here. And we do the exact same thing. We have a sequence ID and then the 10mer.

And then what we want to do is cross correlate, find all the matches between them. And so that's just done with matrix multiply. So A1 transpose A2 will then result in a new matrix, which is reference sequence ID by unknown sequence ID. And then it will then-- the value in here will be how many matches they had.

And so generally you look, if say, 400 was the match, then you would be looking for things well above 30 or 40 matches between them. Or our true match would be maybe 50%, 60%, 70% of them match. And so then you can just apply a threshold to this to deter-- to find the true, true matches.

So very simple. And, you know, there are large software packages out there that do this. They essentially do this algorithm with various twists and variations and stuff like that. But here we can explain the whole thing knowing the mathematics of associative arrays on one slide.

So this calculation is what we call a direct match calculation. We're literally comparing every single sequence's 10mers with all the other ones. And this is essentially what sequence comparison does. And it takes a lot of computation to do this. If you might have millions of sequence IDs and millions of sequence IDs, this very quickly becomes a large amount of computational effort.

So we are, of course, interested in other techniques that could possibly accelerate this. So one of the things we're able to do using the Accumulo database is ingest this entire set of data as an associative array into the database. And using Accumulo's tally features, then tally. Have it essentially, as we do the ingestion, automatically tally the counts for each 10mer. So we can then essentially construct a histogram of all the 10mers. Some 10mers will appear in a very large number of sequences, and some 10mers will appear in not very many.

So this is that histogram, or essentially a degree distribution. We talked about degree distributions in other. This tells us that there's one 10mer that appears in, you know, 80% or 90% of all the sequences. And then there's 20 or 30 10mers that appear in just a few. And then there's a distribution, as in this case, almost a law-- of sort of almost a log normal curve that shows you that really, most of the 10mers seem to have-- appear in a few hundred sequences.

And so now one thing I've done here is create certain thresholds to say, all right. If I only wanted-- if I looked at that large, sparse matrix of the data, and I wanted to threshold, what-- how many-- how much of the data would I eliminate? So if I wanted to eliminate 50% of the

data, I could set a threshold, let's only look at things that are less than a degree of 10,000.

You might say, well, why would I want to eliminate these very, very popular things? Well because they appear everywhere, a true match is-- they're not going to give me any information that really tells me about true matches. Those are going to be clutter. Everything has them. That two sequences share that particular 10mer doesn't really give me a lot of power in selecting which one it really belongs to.

So like-- so I can do that. If I wanted to go down to only 5% of the data, I could say, you know, I only want to look at 10mers that are 100, you know, or that have-- that appear in 100 of these sequences or less. And if I wanted to go even further, you know, I could go down to is 20, 30, 40, 50, and I would only have one half percent of the data. I would have eliminated from consideration 99.5% of the data.

So and then if I can do that, then that's very powerful because I can quickly take my sample data set. I know all the 10mers it has. And I can quickly look it up against this histogram and say, "No. I don't want to do any. I only care about this very small section of data, and I only need to do correlations from that."

So let's see how that works out. And I should say, this technique is very generic. You could do it for text matching. If you have documents, you have the same issue. Very popular words are not going to tell you really anything meaningful about whether two documents are related to each other. It's going to be clutter. And other types of records of that thing, you know?

So in the graph theory perspective, we call these super nodes. These 10mers are super nodes. They have connections to many, many things, and therefore, if you try and connect through them, it's just going to not give you very useful information. You know, it's like people visiting Google. Looking for all the records where people connect-- visited Google is not really going to tell you much unless you have more information. And so it's not a very big distinguishing factor.

So here's an example of the results of doing-- of selecting this low threshold, eliminating 99.5% of the data, and then comparing our matches that we got with what happens when we've used 100% of the data. And so what we have here is the true 10mer match and then the measured sub-sampled match here.

And what you see here is that we get a very, very high success rate in terms of we basically

detect all strong matches using only half percent of the data. And, you know, the number of false positives is extremely low. In fact, a better way to look at that is if we look at the cumular-- cumulative probability of detection. This shows this that if the actual match, if there was actually 100 matches between two sequences, we detect all of those using only 1/20 of the data.

And likewise, in our probability false alarm rate, we see that if you see more than a match of ten in the sub-sample data, that is going to be a true match essentially 100% of the time.

And so this technique allows us to dramatically speed up the rate at which we can do these comparisons. And you do have the price to pay that you have to pre-load your reference data into this special database. But since you tend to be more concerned about comparing with respect to it, that is a worthwhile investment.

So that's just an example of how you can use these techniques, use the mathematics of associative arrays, and these databases together in a coherent way. So we can do more than just find. So the matrix multiply I showed you before, A1 times A2 transposed showed us the counts, the number of matches.

But sometimes we don't want to know more than that. We want to know not just the number of matches, but please show me the exact set of 10mers that caused the match, OK? And so this is where, and I think we talked about this in a previous lecture, we have these special matrix multiplies that will actually take the intersecting key in the matrix and multiply it now, assign that to the value field or [INAUDIBLE].

And so that's why we have the special matrix multiply called CatKeyMul. And so, for instance here, if we look at the result of that, which is AK, and we say, "Show me all the value matches that are greater than six in their rows and their columns together," now we can see that this sequence ID matched with this sequence ID. And these were the actual 10mers that generated that they had in common.

Now clearly six is not a true match in this little sample data set. We don't have any true matches. But this just shows you what that is like. And so this is what we call a pedigree preserving correlation. That is, it shows you the-- it doesn't just give you the count. It shows you where that evidence came from. And you can track it back.

And this is something we do want to do all the time. If you imagined two documents that you

wanted to cross correlate, you might say, all right. I have these two documents, and now I've cross correlated their word matches. Well, now I want to know the actual words that matched, not just the counts. And you would use the exact same multiply to do that.

Likewise, you could do the word/word correlation of a document. So that would be A transpose times A instead of A, A transpose. And then it would show you two words. It would show you the list of documents that they actually had in common. So again, this is a powerful-- a powerful tool.

Again, I should remind people when using this, you do have to be careful when you do, say, CatKeyMul A1 times A1 transpose if you do a square because you will then end up with this very dense diagonal, and these lists will get extremely long. And you can often run out of memory when that happens. So you do have to be careful when you do correlations, these CatKeyMul correlations on things that are going to have very large, overlapping matches.

The regular matrix multiply, you don't have to worry about that, creating dense. You know, that's not a problem. But that's just a little caveat to be aware of. And there's really nothing we can do about that if you do square them, you know? And we've thought about creating a new function, which is basically squaring with and basically not doing the diagonal. And we may end up making that if we can figure that one out.

So once you have those actual specific matches here, so for example, we have our two reference samples. And we looked at the ones that were larger. So here's our two sequences. This, if we look back at the original data, which actually stored the locations of that. So now we're saying, oh, these two have six matches between them. Let me look them up through this one line statement here. Now I can see the actual 10mers that match and their actual locations in the real data.

And from that, I can then deduce that, oh, actually this is not six separate 10mer matches, but it's really two sort of-- what is this, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. One 10mer match, and then I guess-- I think this is like a 17 match, right? And likewise over here. You get a similar type of thing.

So that's basically stitching it back together because that's really what you're trying to do is find these longer sequences where it's identical. And then people can then use those to determine functionality and other types of things that have to do with that. So this is just part of the process of doing that. And it just shows by having this pedigree-preserving matrix multiply and

the ability to store string values in the value, that we can actually preserve that information and do that.

So that just shows you, I think, a really great example. It's kind of one of the most recent things we did this summer in terms of the power of D4M coupled with databases to really do algorithms that are completely kind of new and interesting. And as algorithm developers, I think that's something that we are all very excited about.

So now I'm going to talk about how this fits into an overall pipeline. So once again, just for reminders, you know, D4M, we've talked mostly about working over here in this space, in the Matlab space, for doing your analytics with associative arrays. But they also have ways-- we have very nice bindings too. In particular, the Accumulo database, but we can bind to just about any database.

And here's an example. If I have a table that shows all this data, and I just wanted to get, please give me the column of this 10mer sequence, I would just type that query and it would return that column for me very nicely. So it's a very powerful binding to databases.

It's funny because a lot of people, when I talk to them, are like-- they think D4M is either just about databases or just about this associative array mathematics. And because they usually kind of-- people take sort of-- they usually see it sort of more used in one context or another. And it's really about both. It's really about connecting the two worlds together.

That particular application that we showed you, this genetic sequence comparison application is like you-- is a part of a pipeline, like you would see in any real system. And so I'm going to show you that pipeline here.

And so we have here the raw data. So Fasta is just the name of the file format that all of this DNA sequence comes in, which basically looks pretty much like a CSV file of what I just saw you. I guess that deserves a name as a file format, but, you know. So it's basically that.

And then one thing we do is we parse that data from the Fasta data into a triples format, which allows us to then really work it with this associative array. So it basically creates the 10mers and puts them into a series of triple files, or a row triple file, which holds the sequence ID, the actual 10mer itself, a list of those, and then the position of where that 10mer appeared in the sequence.

And then typically what we do then is we read-- we write a program. And this can be a parallel program if we want it that will then, usually in Matlab using D4, that will read these sequences in and will often just directly insert them without any additional formatting. Just-- we have a way of just inserting triples directly into the Accumulo database, and which is the fastest way that we can do inserts. And then it will also convert these to associative arrays in Matlab and will save them out as mat files because this will take a little time to convert all of these things into files.

You're like, well why would we do that? Well, there's two very good reasons for doing that. One, the Accumulo database, or any database, is very good if we want to look up a small part of a large data set, OK? So if we have billions of records, and we want millions of those records, that's a great use of a database.

However, there are certain times we're like, no. We want to traverse the entire data set. Well, databases are actually bad at doing that. If you say, "I want to scan over the entire database," it's like doing a billion queries, you know? And so there's overheads associated with that.

In that instance, it's far more efficient to just save the data into these associative array files, which will read in very quickly. And then you can just-- if you're like, "I want to do an analysis of the data in that way where I'm going to want-- I really want to work with 10% or 20% of the data." Then having these-- this data in this already parsed into this binary format, is a very efficient way to run an application or an analytic that will run over all of those files.

It also makes parallelism very easily. You just get-- let different processors process different files. You know that's very easy to do. We have lots of support for that type of model. And so that's a good reason to do that.

And at worst, you're doubling the size of the data and your database. Don't worry about it. We double data and databases all the time. Databases are notoriously, if you put a certain amount of data in, they make it much larger. Accumulo actually does a very good job of that. It won't really be that much larger. But there's no reason to not, as you're doing the ingest, to save those files.

And then you can do various comparisons. So for example, we then can-- if this-- if we save the sample data as a Matlab file, we could read that in. And then do our comparison with the reference data that's sitting inside the database to get our top matches. And that's exactly how this application actually works.

So this pipeline from raw data to an intermediate format to a sort of efficient binary format and insertion to then doing the analytics and comparisons. And eventually, you'll usually often have another layer, which is you might create a web interface to something. We're like, OK. This is now a full system. A person can type in your-- or put in their own data. And then it will then call a function.

So again, this is very standard how this stuff ends up fitting into an overall pipeline. And it's certainly a situation where if you're going to deploy a system, you might decide, you know what? I don't want Matlab to be a part of my deployed system. I want to do that, say, in Java or something else that's sort of more universal, which we certainly see people do.

It still makes sense to do it in this approach because the algorithm development in testing is just much, much easier to do in an environment like D4M. And then once you have the algorithm correct, it's now much easier to give that to someone else who is going to do the implementation in another environment and deal with all the issues that are associated with maybe doing a deployment type of thing.

So one certainly could use the Matlab or the new octave code in a production environment. We certainly have seen that. But often the case, one has limitations about what can deploy. And so it is still better to do the algorithm development in this type of environment than to try and do the algorithm in a deployment language like Java.

One of the things that was very important for this database, and it's true of most parallel databases. So if we want to get the highest performance insert, that is, we want to read the data and insert it as quickly as possible in the database, typically we'll need to have some kind of parallel program running, in this case, maybe each reading different sets of input files and all inserting them into the parallel database.

And so in Accumulo, they divide your table amongst the different computers, which they call tablet servers. And it's very important to avoid the situation where everyone is inserting and all the data is inserting into the same tablet server. You're not going to get really very good performance.

Now, the way Accumulo splits up its data is similar to many other databases. It's called sharting, sometimes they call it. It just means they split up the table, but the term, in the database community, uses-- they call it sharting. What they'll do is they'll basically take the

table and they'll assign certain row keys, in this c-- in Accumulo's case, certain contiguous sets of row keys to particular tablet servers.

So, you know, if you had a data set that was universally split over the alphabet, then-- and you were going to split it, the first split would be between m and n. And so this is called splitting. And it's very important to try and get good splits and choose your splits so that you get good performance.

Now D4M has a native interface that allows you to just say, here are the-- I want these to be the splits of this table. You can actually compute those and assign them if you have a parallel instance.

In the class, you will only be working on the databases that will be set up for you. We have set up for you are all single node instances. They do not have multiple tablet servers, so you don't really have to do-- deal with splitting. It's only an issue in parallel, but it's certainly something to be aware of and is often the key.

People will often have a very large Accumulo instance. And they may only be getting not very-- the performance they would get on-- in just a two node instance because-- and usually it's because their splitting needs to be done in this proper way. And this is true of all databases, not just Accumulo. But other databases have the exact same issue.

Accumulo is called Accumulo because it has something called accumulators, which is what actually makes this whole bioinformatics application really go, which is that I can denote a column in a table to be an accumulator column, which means whenever a new entry is hit, some action will be performed. In this case, the default is addition.

And so a standard thing we'll do in our schema, as we've already talked about these exploded transpose pair schemas that allows do fast lookups in rows and columns is we'll, OK, create two additional tables, one of which holds the sums of the rows, and one of which holds the sums of the columns which then allows us to do these very fast lookups of the statistics, which is very useful to know how do we avoid accidentally looking up columns that will have-- that are present in the whole database.

An issue that happens all the time is that you'll have a column, and it's essentially almost a dense column in the database, and you really, really, really don't want to look that up because it's basically giving you the whole database. Well, with this accumulator, you can look up the

column first, and it will give you a count and be like, oh, yeah. And then you can just say, no. I want to look up everything but that column. So very powerful, very powerful tool for doing that.

That's also another reason why we construct the associative array when we load it for insert because when we construct the associative array, we can actually do a quick sum right then and there of whatever piece we've read in. And so then when we send that into the database, we've dramatically reduced the number of-- the amount of tallying it has to do.

And this can be a huge time saver because if you don't do that, you're essentially reinserting the data two more times because, you know, when you've inserted it into the table and it's transpose, and to get the accumulation effect, you would have to directly insert it. But if we can do a pre-sum, that reduces the amount of work on that accumulator table dramatically. And so again, that's another reason why we do that.

So let's just talk about some of the results that we see. So here's an example on a-- let's see, this is an eight node database. And this shows us the ingest rate that we get using different numbers of ingesters. So this is different programs all trying to insert into this database simultaneously. And this just shows the difference between having 0 splits and, in this case, 35 splits was the optimal number that we had. And you see, it's a rather large difference. And you don't get-- you get some benefit with multiple inserters, but that sort of ends when you don't do this. So this is just an example of why you want to do that.

Another example is with the actual human DNA database. This shows us our insert rate on doing-- using different numbers of inserters. And yeah, so this is eight tablet servers. And this just shows the different number of ingesters here. And since we're doing proper splitting, we're getting very nice scaling.

And this is actually the actual output from Accumulo. It actually has a nice little meter here that shows you what it says you're actually getting, which is very, very nice to be able to verify that you and the database both agree that your insert rate is-- so this is insert right here. So you see, we're getting about 4,000-- 400,000 entries per second, which is an extraordinarily high number in the database community.

Just to give you a comparison, it would not be uncommon if you were to set up a MySQL instance on a single computer and have one inserter going into it to get maybe 1,000 inserts per second on a good day. And so, you know, this is essentially the core reason why people use Accumulo is because of this very high insert and query performance.

And this just shows our extrapolated run time. If we wanted to, for instance, ingest the entire human Fasta database here of 4.5 billion entries, we could do that in eight hours, which would be a very reasonable amount of time to do that.

So in summary, what we are able to achieve with this application is this shows one of these diagrams. I think I showed one in the first lecture. This is a way of sort of measuring our productivity and our performance. So this shows the volume of code that we wrote. And this shows the run time. So obviously this is better down here.

And this shows BLAST. So BLAST is the industry standard application for doing sequence match. And I don't want, in any way, to say that we are doing everything BLAST does. BLAST is almost a million lines of code. It does lots and lots of different things. It handles all different types of file formats and little nuances and other types of things. But at its core, it does what we did. And we were able to basically do that in, you know, 150 lines of D4M code. And using the database, we were able to do that 100 times faster.

And so this is really the power of this technology is to allow you to develop these algorithms and implement them and, in this case, actually leverage the database to essentially replace lookups with computations in a very intelligent way, in a way that's knowledgeable about the statistics of your data. And that's really the power. And these are the types of results that people can get.

And this whole result was done with one summer student, a very smart summer student, but we were able to put this whole system together in about a couple of months. And this is from a person who knew nothing about Accumulo or D4M or anything like that. They were a good-- they were smart. They were good, solid Java background and very energetic. But, you know, these are the kinds of things we've been able to see.

So just to summarize, we, again, we see-- I think just to-- this types of techniques are useful for document analysis and network analysis and DNA sequencing. You know, this is really summarizing all the applications that we've talked about. We think there's a pretty big gap between doing the data analysis tools that our really algorithm developers need. And we think D4M is really allowing us to use a tool like Matlab for its traditional role in this new domain, which is algorithm development. Figure things out, getting things right before you can then hand it on to someone else to implement and actually get enough production environment.

So with that, that brings the end to the lecture. And then there's some examples where we show you the-- this stuff. So-- and there's no homework other than I sent you all that link to see if your access to the database exists. Did everyone-- raise your hand. Did you check the link out? Yes? You logged in? It worked? You saw a bunch of databases there? OK.

If it doesn't work, because the next class is sort of the penultimate class, I'm going to go through all these examples. And the assignment will be to run those examples on those databases. So you will be really using D4M, touching a database, doing real analytics, doing very complicated things. The whole class is just going to be a demonstration of that. Everything we've been doing has been leading up. So the concepts are in place so that you can understand those examples fairly easily.

All right, so we will take a short break here. And then we will come back, and I will show you the demo, this week's demo.