Pioneers in Quality Electronic Clinical Quality Measure (eCQM) Expert to Expert Webinar Series
Technical Implementation of the Clinical Quality Language (CQL)

July 23, 2019

Lisa Anderson: Hello everyone. I'm Lisa Anderson, project director for eClinical in the department of quality measurement at The Joint Commission. I would like to thank you for joining our Pioneers in Quality 2018/2019 Expert to Expert series. Today's session is focused on the technical implementation of the clinical quality language, or CQL. For closed captioning services, please use the link on this slide. This information is also accessible via the participant pane and the GoToWebcast platform. The Joint Commission and CMS designed the Pioneers in Quality Expert to Expert series to support hospitals in using electronic clinical quality measures and transitioning to the new clinical quality language. We introduced this series with a CQL basics webinar followed by six sessions covering the eligible hospital/critical access eCQMs for the 2019 reporting year.

Today's session is intended for a technical staff audience, such as EHR report writers, hospital IT staff engaged in eCQM implementation, clinical informaticists, EHR analysts, and vendor staff that support hospitals in their CQL implementation. Clinical staff attending should also be conversant in the technical concepts of eCQM implementation. This session is scheduled for 90 minutes to allow for a live Q and A session. At the end of today's session, participants should be able to describe how CQL compares to SQL, describe the logic sharing architecture of CQL, and locate resources regarding CQL technical implementation.

The slides are available in the Event Resources pane. Select the PDF to download and print the slides. These sessions are meant to be interactive. The Ask a Question pane permits participants to ask questions and view responses in real time. If possible, please reference the slide number in your question. Additionally, you can visit links or resources noted in the slides. Please note, a recording of today's presentation, the slide deck, and Q and A documents will be available on The Joint Commission website in a few weeks. We hope you find this information helpful and share it with interested colleagues.

CE credits are offered for all our pioneering quality webinars. This webinar is approved for one continuing education credit for Accreditation Council for Continuing Medical Education, American Nurses Credentialing Center, American College of Healthcare Executives, California Board of Registered Nursing, and International Association for Continuing Education and Training. CE, CME, CEU credits are available for the live audio only. Credits will not be available for webinar replays. To claim credit, you must have individually registered for the webinar, listen to the live webinar in its entirety (only those listening live are eligible to receive credit), completed a post-program evaluation/at a station. The program survey link will be sent to participants' emails after the webinar. Principle certificates will be mailed to those eligible two weeks after the session. All participant CE certificates will be sent at the same time. If you are listening with colleagues and do not use your own link or phone line to join, you can still obtain CE credit if you meet these three criteria. An automated email after the session will provide information on how to access the survey. For more information on The Joint Commission’s continuing education policies, please visit the link provided at the bottom of this slide.

The following staff and speakers have disclosed that neither they, nor their spouses or partners, have any financial arrangements or affiliations with corporate organizations that either provide educational grants for this program or may be referenced in this activity. Lisa Anderson, MSN, RN-BC project director eClinical, Department of Quality Measurement, and Bryn Rhodes, ESAC. ESAC is a Centers for Medicare and Medicaid Services subcontractor. Today's session will be divided into three sections: CQL implementation, CQL/SQL side by side, and CQL/SQL translation. I am now going to turn it over to Bryn to begin his presentation.

Bryn Rhodes: Thank you, Lisa. So my name is Bryn Rhodes. I have been working with clinical quality language for the past several years. I'm the editor of the specification, an active in the clinical decision support and clinical quality information work groups at HL7 where we steward that specification and carry on that work. So clinical quality language, so let's start with just a very high level: What is it? So we built clinical quality language to address two primary use cases. So one, how can we most effectively share clinical logic, and how can we do so in a way that streamlines consumption by both humans and machines? So CQL is an HL7 standard designed to enable automatic point-to-point sharing of executable clinical knowledge, so eCQMs are a great example of that and two, to provide a clinically-focused, author-friendly, and human-readable language.

The language was developed as a harmonization of requirements from across the quality improvement spectrum and from several different and overlapping standards for logic representation, including Arden, GELLO, QDM, HQMF. In 2014, several work groups within HL7 produced a domain analysis model based on these input specifications and informed by modern compiler and language design approaches, and this model formed the foundation for clinical quality language, and over the next several years, that specification was built with the input and involvement of a broad spectrum of clinical quality stakeholders.

At this point, we've completed the fourth FTU ballot and the language has been adopted by CMS for use in specifying eCQMS, and it's being piloted in CQL-based decision support in various locations. We're also seeing increased adoption among implementation guides as its broad utility for sharing clinical logic across the healthcare domain is being explored for use in public health, guideline development, vendors, decision support providers, and clinical research. And there's a nice, simple URL there for accessing the currently published version of the specification.

So when we think about the problem with sharing clinical logic, it's critical to separate the concerns as much as possible. So the conceptual level of CQL specification approaches this problem are considering three main components. First is the data model, so the structures of the information involved. Second is the terminology, so these are standard terminologies like SNOMED-CT, LOINC, RxNorm. And third is the logic itself, so CQL intentionally keeps these components separate. So the CQL specification is concerned with how you express logic in terms of some data model and relating to some set of terminology. If separation allows the data models and terminologies to evolve, independent of the CQL specification, it gives a lot more flexibility architecturally, and as a result, CQL is able to be used with different data models, so we can use it with the quality data model or we can use it with Ire or any other data model that you can describe the structures in a way that CQL translator can consume.

So looking broadly at the architecture of CQL, at the highest level you have a syntax that authors can use to produce libraries that contain human-readable, but precise logic, so this is when you look at the human-readable narrative for any CQM, for example, you'll see the clinical quality language statements. The second layer is what we call expression logical model, so statements of CQL are translated into ELM, and ELM is an XML representation of the logic involved, and it's a machine friendly rendering. It's designed to support and to streamline language processing applications, and the development of things like translators and engines to support consumption of that logic automatically.

Digging into that a little deeper, this is a depiction of a standard compiler pipeline. You start at the conceptual level with the syntax. In this example just a simple arithmetic expression. CQL as a language is defined at this level. The specification has a grammar that defines all of the tokens involved and the parsing rules. So the first stage is lexical analysis to break that grammars down into a stream of tokens, then to parse those tokens into an abstract syntax tree. The expression logical model is defined at this level, the level of an abstract syntax tree. That allows language processing applications to more easily deal with the logic expressions involved.

The next page then is semantic analysis, where we validate that the expressions in the language make sense. You're adding integers, you're not adding strings and integers. So what you get out of semantic analysis is then a verified syntax tree. From there you can use that to compile, or translate, or run whatever your particular environment supports.

This pipeline is based on modern compiler theory. In general this is how compilers work. By linking in at this level and defining the specification in this way, we've taken as much as possible of the leg work out of turning CQL into something that is a machine executable without making any platform specific assumptions about where it's going to run, and how that's going to happen. That results in a lot of flexibility for implementation.

Let's dig a little deeper into expression logical model. It's essentially a byte-code representation, or if you're familiar with .net, it's an intermediate language representation. It carries sufficient semantics to enable execution independent of the CQL that produced it. The ELM that's shared, and if you look at the ECQ packages, they have both CQL and ELM content. You can use the ELM directly to actually perform translation or execution activities. So ELM is a canonical representation. It has some more primitive operations that are focused on supporting implementation use cases.

So in CQL you'll see a lot of ... we'll dig into them a bit later, but you'll see a lot of high level construct that enable authors to write very, very clear and high level expressions. Those are all translated down in the ELM to simpler representations in terms of implementation. Fewer operators involved, and fewer choices about implementation.

Conceptually this is the same abstraction that underlies HTML. So an HTML webpage, for example, can describe a document independent of any particular platform. Then platform specific browsers render that webpage, so users get the same experience regardless of the technology they're using. So in the same way, when we share the CQL description of and ECQM, you can render that in any platform that supports usage of CQL.

So expression logical model, digging in just a little bit. ELM expressions are built as trees of nodes. Each kind of expression is representative of different node type. So two plus two, for example, is an add node with a literal node of two, and a literal node of two. This format allows implementations. A naive implementation of an engine, for example, can just deserialize this graph and execute each node where each node understands how do execute its operation.

In general then, operations and functions and CQL have an equivalent ELM representation. The equal operator and CQL is the equal ELM node type and add, etc. If you look in the logical specification chapter of CQL, there's a complete description of all of the node types for ELM, as well as the description of the mapping from CQL to ELM.

So, type categories. In order to describe logic we need a basic type system. This type system is the kind of minimum set of data types that you need in order to describe information. Data models that are expressed for usage in CQL map to these to provide a representation that CQL operators can consume. The primitive types are Boolean, string, integer, the basic types you'd expect.

Then we have collection types, so you can have lists of any type. Then you have structured types. These are class types like encounter, or patient and tuple types, which is just a same as a class type, but it's anonymous. There's no name for it. Its type is just the list of properties. Then interval types. You can have interval on any type that is ordered, meaning it supports comparison. So if you have strings, or integers, or decimals, many times you can define intervals over those because they support ordering over the type.

Next, let's look at a simple example of a retrieve. Within CQL, one of the most important constructs is the retrieve. Anytime you see the square brackets within CQL, that's the retrieve. If you're familiar with QDM logic, this should be readily familiar to you. The retrieve consists of a type, so this is in terms of the data model. You're saying "I'm going to retrieve values from this type." That that might correspond to a table in a relational system that might correspond to a document store in a document based system. Then you say, "Within that diagnosis, I want to match codes that are in the acute pharyngitis value set."

This is a reference to a value set. That retrieve is designed to ensure that the only points in CQL where you can actually access data all go through this retrieved structure. You can think of it as the definition of the data access layer within CQL. It's focused on only those types of retrieve that can be accelerated through the use of indexes. We chose terminology and date range as the most selective indexes across this type of information, across healthcare information. That then is expressed in ELM using a retrieve node. You can see the data type corresponds to diagnosis, but it's resolved to a particular a model type.

There's a template ID. This can be used to specify a template or a profile, and then the code property. You'll notice that code isn't specified in the CQL. This is the model info for QDM. It specifies that for diagnosis, the primary code filter type is code. And then it indicates that the codes are coming from a reference to the acute pharyngitis value set.

Digging into model info, this is the structure that the CQL translator uses to describe data models. So for QDM, this is a snippet of the QDM diagnosis in class in the model info there. As you can see it specifies the name, an identifier, and a label. That label is the label that you actually use in CQL, whether or not it's retrievable, so the model info supports describing structures that don't necessarily correspond to things that can be retrieved, so only things that are retrievable can appear in retrieves. It's a hierarchical structure, so you can specify base types, and those types will inherit their content from the parent and then define the elements available, so onset, abatement, anatomical location, et cetera. That's just an example of what model info looks like.

System model defines the base types, so for all those primitives and the structured types that are supported by the type system, as well as quantity, code, and concept. And these are the clinically relevant data types, so quantities appear all over in healthcare data and terminologies, of course, are ubiquitous. So then, a CQL library, the named versioned grouping of CQL components. So each library has a name and a version, specifies any number of data models. Typically you'll see just one, but there's an implicit usage of system in all of them. Then there's a terminology section. You can specify code systems, value sets, and codes. Note that these are just declarations. We're just referring to terminology that is defined elsewhere, so you don't use the CQL content to define the terminology, you use it to declare references to the terminology so that you can use this name, inpatient, as a reference to this value set anywhere in this logic. And then you have parameters. These can be any kind of parameters in an eCQM. You'll see a measurement period defaulting to an interval of the initial effective period for the measurement. And then a context patient, we'll talk more about that. And then the named expressions.

So patient context, CQL has the notion of a context. This is an implicit filter. It allows authors to write from a particular perspective, so instead of having to say medications for patient X, they can just say medications and they're in the patient context, and so that retrieve only returns medications for a particular patient. ECQMs are typically written from a patient perspective and that simplifies the logic in the expression. The data access layer then is responsible for resolving that pattern. So then here's an example of a rendering of an expression in CQL to ELM. So inpatient encounters, this is a retrieve of the encounter performed in the inpatient value set. Length of stay is less than or equal to 120 days, and the discharge is during the measurement period. So you see this renders in ELM as a query. The source alias of E, that source is the expression that is a retrieve of encounter perform data in the inpatient value set, and then there's a filter, a where, that consists of an and of the lesser equal of length of staying, which is a property reference, and the in of discharge date time, which is again a property reference and the measurement period which is a parameter reference. So the CQL to ELM translator is taking CQL and producing this ELM output, and if you look again on an eCQM package, you'll see both of these representations of the measure are present.

So going back to evaluation approaches and this general CQL architecture. When you go from CQL to ELM, you either have something that can evaluate that ELM natively, or you need to translate it into something that runs in your particular environment, so there's a JavaScript translator, or interpreter rather, you could translate it into a Drools production system. For this presentation, we're focusing on the SQL. So SQL is one of the, kind of, design inspirations for CQL, so there are a lot of similarities between CQL and SQL, and translation from the ELM to SQL is a fairly straightforward process intentionally.

So a comparison of CQL and SQ in terms of constructs. So one of the primary central language constructs in CQL is a query. It's a clinically focused query language where SQL is a generalist query language, but the clauses are very similar. There's a source clause that's equivalent to the from clause in SQL. There are relationship clauses with and without, that's similar to a join. We'll get into some of the differences, and then the where clause is the same, and then there's a return which is equivalent to the select clause, and a sword that's equivalent to order by.

So if you look at the broad structure of CQL query as very similar to SQL, and you can use a lot of the same ideas and thinking about those queries and how they're implemented. So if we look at that example we were just looking at and you pull out another tree of execution, it starts at the top of the query and then you evaluate the retrieve, and for each element in that retrieve, you evaluate the where in terms of the and property and the quantity, and that property reference is a reference to the current iteration of the retrieve. So you can imagine that as the pipelined execution, that's a fairly straightforward representation of the query plan for that approach.

So one of the tools that is available for translating ELM is a .net project that was built as part of the healthy decisions initiative. It is a framework for building ELM language processing applications, and so what it does is just describe the generic structure for the serializing ELM tree into a graph of nodes that perform different operations. They might have an engine set of operations, you might have a translation set and one of the projects that is built there currently is a translation into SQL. It was used as part of the pilots during healthy decisions initiative. It was used to validate the knowledge authoring specification examples and to translate a chlamydia screening measure into SQL against an OMOP data model. And so that tooling is somewhat outdated in terms of the specific...outdated in terms of the specification, but the core ELM is the same largely and so that is a reasonable starting point if people are interested and there's a lot of I think, good kind of reference implementation code at that repository.

So if we think a little bit about calculation architecture for CQL, we need these basic components. You know at the highest level you need at least a calculation engine that actually performs the calculations. You need a description of the logic, how the measure calculates against the clinical information. You need some representation of the model, the actual data involved. You need a data access interface, so some way for the logic to actually retrieve data in terms of that model. You need a terminology interface. Some way to actually reference the terminologies that are involved. A CQL by design does not define terminology, it just references them. So you need some interface to a terminology server to resolve when you've said, give me the inpatient value set, what does that mean? And you need some interface to libraries. You know, the CQL is structured around libraries and you can share logic between measures, between decision supports. And so you need some way when you're evaluating a CQL to reference libraries that might be used within that logic.

So that gives kind of an overall, that was kind of a whirlwind tour of CQL and ELM and how they relate to each other, and how you might use a CQL and a calculation engine environment. So next we're going to look at what different queries in CQL look like in SQL. So if you were to run them through a translator from CQL to SQL, what that would actually look like.

So looking at just an example of a query, this is a straight forward encounter performed non-elective inpatient encounter. Length in days is lacking, I guess 120 this retrieve is the source of the query and this nonelective encounter is the alias. So this non-elective encounter is the name that is scoped to this query and so whenever it's referenced within this query, it refers to an iteration of this source. The non-elective encounter is referring to whichever encounter performed we're currently ranging over and so that's how the logic relates to the source.

So the equivalent example in SQL, this one is represented using an exist, shall we say this is the equivalent to select star from encounter performed called non-elective encounter? Okay. Where the non-elective encounters patient ID is that patient ID? So this is that context where we've said using context a patient, the evaluation environment, it provides that patient ID either by ranging over all the patients that are available or by actually translating it into a patient level or into a population level query with the patient filters. Are the patient relationships established? It joins. Then we say and exists the select star from value set codes, so this kind of is imagining a terminology implementation where you have a value set codes table that has for each value set name the code and system. And so this retrieve could be represented in an SQL database using this structure.

So an alternative representation would be to say, do this with a join. It's the same structure where you're just saying select star from encounter performed called non-elective encounter. Join the value set codes on the value set name is non-elective inpatient encounter and the VSC code matches the non-elective encounter code.

So just kind of an aside about terminologies. All the terminology is referenced in the measure are included in the terminology section of the narrative description of the measure. They're also all defined at the top of any given ELM library. So the value set references, the code system references, and the code references. Those all will be present in the terminology section.

Looking at the terminology referenced by any given CQL library is a matter of looking at the ELM and you can easily pull out all the terminology that's referenced. There's no way to talk about terminology in the body of the CQL without it being declared as a value set code system or code within the terminology section. So that makes a terminology analysis much easier in CQL.

So filtering with where, you'll see timing crazes like tens during, those are translated effectively to eight comparisons once you get down to the ELM. And so the resulting SQL, I've left out the value set filters there, but we will effectively say, and global length in days, this is a reference to that function here. Length in days, a non-elective encounter, relevant period start and relevant period end, so notice that I've got a relevant period here attribute. This is an interval valued attribute in CQL and this would be typically represented in SQL as two different columns, a start and an end.

So digging a little more into types of timing relationships. This is just a survey to describe kind of what kinds of comparisons are available. You can compare just date/times. You can compare a date/time and an interval. So you can say that author date/time is during encounter relevant period. You can compare an interval with a date/time so that you can say the relevant period includes the author date/time, and you can also compare two intervals directly relevant period during a measurement period. And so within the representation in SQL, those would all need to be represented either as direct comparisons, as some SQL equivalent of that operation. For example, you might define during a function user defined function in SQL and use that as the implementation for a translation or you might just in line the actual comparison so that the author date time is between the start and stop of the relevant period.

Then for comparing two intervals, the same applies. You would either define user defined functions to support the description of the during operation or you would translate that directly to the equivalent comparisons in terms of the boundaries of the intervals.

Intervals and timing phrases, there are several timing phrases available within CQL and in general the translator is going to resolve those timing phrases in terms of more primitive ELM operations. So that a translation layer doesn't have to deal with all of the different possible combinations of timing phrases. They only have to deal with the defined operations within the ELM and the CQL to ELM translator represents the timing phrases in terms of those more basic primitives. So you'll see operations like prevalence period overlaps. That's a direct interval operator. You'll see phrases using starts and ends. So starts before the start or starts on or before the end of. Those are actually translated into boundary accesses on the interval involved. You'll see timing phrases with offsets. So you can say, author date time 24 hours or less before the start. These are translated variously into date, time, arithmetic operations or boundaries depending on the timing phrase. But in ELM you won't see this timing phrase, you'll see a canonical representation of it. And finally, timing phrases with precision. So you can actually say what precision you want the comparisons to occur. So the relevant period ends one day after the day of the start of the qualifying encounters relevant period. That 'day of' means that you want that comparison to be performed only to the day. You want to ignore time in that comparison. And the precision will be carried through, and that's what's represented in the ELM. So all the operations in ELM, you can specify a precision.

So as an example of relationships, there is a width relationship which is in SQL. This is, in relational languages, it's called a semi-join. It's effectively a join, except that you only return the left side of the join. So in this case, we're saying ischemic stroke encounter event that thrombotic therapy at discharge such that the author date time is during the relevant period. But you can do this with an exists. You can say, select star from ischemic stroke encounter. So I'm assuming here the existence of a view named "ischemic stroke encounter" and that that view would correspond to that "ischemic stroke encounter" expression definition. And then where exists, select star from antithrombotic therapy at discharge, where that author date time is greater than or equal to the relevant period starts and less than or equal to the relevant period ends. You could also use a between there if your SQL dialects supported that.

For multiple relationships, if you have an encounter or a query like this, I'm seeing a live birth encounter with gestational age 37 weeks or more with substance administered breast milk and without substance administered dietary intake other than breast milk. Both of those criteria apply in order to satisfy. So I need to say where exists. That's the same pattern that we used up in the previous example. And when not exists, select "substance administered, other feeding".

So an example of alternative relationships, if you want to say in this case, encounter with discharge disposition to home or police custody with an asthma management plan completed, or an encounter with no asthma management plan due to patient refusal. So that's in CQL, generally expressed as a union, since you can't say "with and with", both of those would have to apply. So if you want either of them to apply, state each criteria and then union them. And the same applies in SQL, the equivalent there is to translate the select and then union the results.

So an example of a multi-source query. CQL makes a distinction between a single source and a multi-source query. Most queries in CQL are single source, where you just ... And those are the kinds we've been looking at so far. Where you just reference the source and an alias. For a multi-source query, you introduced that with the "from", and then you can list any number of sources. So I can say, from delivery encounter near-term, medical induction medication, and is in labor. Those three sources are now all available within this query. So in SQL that's equivalent to a times. So delivery encounter from delivery encounter near-term, cross join medical induction medication, cross join is in labor.

And then the where clause is able to reference any of the aliases introduced in the from clause in SQL, and the same is true in CQL. The from clause introduces all of these aliases, and the where clause and the return clause can use of those aliases. So you'll notice this return here on a multi-source query, the default return is to include a tuple that has an element for each alias. And so if you've just excluded the return, the result would include delivery encounter, induction medication, and labor with the tuple for each of those sources, for each combination of those sources. By saying return delivery encounter, I'm saying I only want the delivery encounter results here. We do that in SQL with delivery encounter dot star. That's not always available in SQL dialects. Some dialects support that, and if it didn't support that, it would need to list all of the elements in that select but a ... So that's the return clause there on a from multi-source query.

So for combining withs, CQL supports unions of different types. So in SQL, a union can only be done between the same kinds of table, or the same kinds of result. Within CQL, we allow something like this example, intervention order comfort measures union intervention performed comfort measures. Those are two different types of things, and the result in CQL will be a list that includes both intervention orders and interventions performed. So the way to represent that in SQL is to build what we typically see called an outer union, where you build out the columns that are in common and then the columns that are not common are null in the side that they're not present and specified in the side that they are present. So we don't have a relevant period start in intervention order, and so that shows up as a null named the same thing. And so what you get is a table in SQL that has some columns that are null from both sides of that union.

And once you have nulls then the coalesce operator can be used to say combine elements that may or may not be present. So in this case, when you reference this list in a subsequent expression, you would typically see the usage of coalesce to say if the current row has a relevant period, use that. If that's not present, then use the author date time.

So the return clause can also be used to shape results. In this case, we want an assessment performed. And then we went to union laboratory test performed, but we want that laboratory test performed to look like an assessment performed so that subsequent operations can all treat that content the same way. So we're effectively constructing an assessment performed using the content from this laboratory test performed and using this assessment performed instructor here. And then we can union those together, and instead of having a list with things of different types, now I have a list of things with the same type. And we support both of those approaches to allow authors to mm express the logic and the way in the where that makes the most sense for their use case. If there are significant number of subsequent references, it is typically better to construct a single table. But if there are minimal subsequent references, then allowing unions with multiple types prevents rewrite.

Okay. An example of intersect and accept. So, these are the same that operators in CQL that are in the SQL. And the translation is straightforward. It's the same syntax. You just, for each reference, use a select star from assuming the definition of a view that corresponds to that expression definition. The translation is almost the same. But for this, select star from in front of the V name. And the order of operations and the semantics of those operations are the same between a SQL and CQL.

So, another example; you're using Let: local definitions. So, one of the constructs or clauses within a CQL query is the Let clause. This allows you to introduce expressions that are evaluated only within the context of the query. So, in this case we're saying start with the initial population named qualifying encounter. Let the first PCI be the first instance of a PCI procedure where the relevant period starts on or after the hospital arrival time of that qualifying encounter, sorted by the start of the relevant period. They've given me the most recent PCI related to this qualifying encounter.

And then I can use this first PCI definition anywhere else in the query to talk about that. And because it's part of the query, so where first PCI refers to the left that was calculated for this iteration of the query. But then I can introduce expressions and use those throughout the rest of the query.

So in a TSQl, the dialect that's used in Microsoft SQL server and Sybase and MySQL has a flavor of it, there's an operation called Outer Apply and that is effectively the same thing as a Let where you construct the content and then that Outer Apply is available within the rest of the query. You're introducing, effectively, a result that's XS on each iteration. So the first PCI for this qualifying encounter, and that allows you to translate that example.

So, hopefully that is helpful. That was just a kind of list of lots of different types of CQL expressions and what they're equivalent expression in SQL looks like. If page, it's just a list of resources that can help them be available. Obviously a link to the specification. This is the product brief for CQL on HL seven, the new CQI resource center. There are a lots of excellent resources available there, as well as ongoing education and outreach seminars and events. There are educational resources for past events that had been provided.

There's a CQL space that digs into the current end specifications used for program measures and then there's a CQL formatting in usage Wiki. There's a lot of content here on specifically some guidance around end patterns for usage of CQL. There's a get hub tools repository, CQ framework, clinical quality language, in general. The CQ framework organization has several related repositories, tooling and otherwise around use of CQL and then there is the measure offering tool that uses CQL and QDM to produce the CQM specifications, and there's a bonding testing tool where you can use that environment to test ECQMS produced through the measure authoring tool. And have you have issues, please do submit tickets for CQL through that CQL IT project on the UNC project tracking site. Also included at the end of the deck, several slides that just talk about different potential approaches to implementation given different implementation environments. And with that, I will turn it back over to Lisa.

Lisa Anderson: Thank you Bryn. We will now move into the live Q and A segment for the session. To ask a question, please type your question and slide reference number in the question pane. We will answer as many questions as possible in the remaining time. All questions submitted will be addressed in a follow-up Q and a document that will be posted on the Pioneers in Quality portal.

Our first question, Bryn, comes from Edgardo at Infomatica. 'Could you tell me where we can take CQL training?' We did already provide in the chat box a link to the ECQ resource center as a good place to start with some introductory information for CQL, but he did have a follow-up question to see is there any sort of physical training or online training that he could take?

Bryn Rhodes: So, HL7 does offer a CQL class at their working group meetings. You know, it's a session you can sign up for.

Don't know if they do that other than at the work group meetings. Okay. We could certainly, I would suggest that that's something that there's interest in.

Lisa Anderson: Thanks, Bryn. Our next question comes from Pam at Community Healthcare System. "How do you interpret patients on observation in the hospital? Are observation patients be considered inpatient encounters since they are hospital admission? The ops patients later become inpatient at discharge and they're having some issues with timing," and I'm kind of truncating that question. We did answer that, Pam, in your response already, but you are correct. In the 2019 reporting year for our ECQMs, observation was not accounted for. However, if you look at the specifications for the 2020 reporting year, you will see that that should be resolved. We did implement a new global dot hospitalization with observation function that helps tie together ED, observation, and inpatient encounters. So, hopefully that helps you out. All right. Our next question is for Brynn. My name's got cut off. One second. And this comes from Dan at the VA. "Are there any open source CQL to SQL transpilers based on ANSI sequel, which can be customized to meet platform's specific needs?"

Bryn Rhodes: Yes. So, I'm going to go back to slide 29. This link is exactly that. So, like I said, it's a little bit out of date, but bringing it up to the latest job specification would be a fairly light list. And the SQL a representation, it's not fancy. What it is is something that we call common SQL, which is the lowest common denominator of all known SQL dialects, which was a challenge just to decipher and come up with. But what it puts out would be ANSI SQL. I just don't want to say that it's ANSI SQL because it's based on survey of all known dialects and the lowest common denominator of all of those. But then, within that there are extensions for like if you're translating to Oracle, there are Oracle extensions. If you're translating to IBM DB2, you know, dialect-specific extensions that can be used to support that.

Like I said, it was used in the ELM pilots but the base structure is there and it's functional, yeah.

Lisa Anderson: Thanks, Bryn. On slide 37, there's a question about that slide specifically. Let's go to that. Okay, on the timing relationship. Sorry, my screen [inaudible 00:01:27]. So the DURING keyword acts similarly to the BETWEEN function in SQL, and the INCLUDES works like the IN function in SQL. Does SQL have the ability to do NOT IN or does it have a NOT IN equivalent? I'm guessing something like an EXCLUDE?

Bryn Rhodes; So, there's not a NOT IN. What you say is NOT in, if that makes sense. So SQL allows you to put the NOT modifier in front of several of the keywords. CQL doesn't do that. Instead we just have a generic NOT operator and so any expression that results in a Boolean you can do a NOT. So to say, "X NOT IN Y" in CQL you say, "NOT X IN Y", if that makes sense. The DURING question, so DURING does act like a BETWEEN if the left side of the DURING is a date. The DURING can also be used with intervals, so relevant period during measurement period and then it's actually an interval operation where you're saying that the left interval is entirely included in the right interval. And then, NOT INs and EXCLUDEs, like, for each operation for intervals there's a complement and so you can say DURING and you can also say INCLUDES and those are complementary operations where relevant period includes measurement period would mean the opposite of relevant period during measurement period. Does that help?

Lisa Anderson: Thanks, Bryn. Our next question comes from Lynne at Meditech, and she's saying, "We retrieved the qdm-modelinfo.xml from Github" and she provided a link to Github, and she says, "We used the qdm-modelinfo.xml to produce SQL files. We have a couple questions about that. One, who is the author? We cannot find it on the [eCQI 00:03:53] site, only guidance. And two, is there a schema against which it is validated other than W3C standard? For example, we infer a class info element contains a negation rationale element. It has two profile info child elements, positive and negative, but cannot point to that rule in an XSV.

Bryn Rhodes: Yes. So, in the same repository that you downloaded the modelinfo, there's actually a schema that that modelinfo is used to generate. And the ELM that comes out of the translator written against any given modelinfo should be schema valid against that schema, or at least operate against data that is schema valid against that schema. Does that make sense? And as far as who is the author, as part of building the CQL ELM translator and the reference tooling that supports the clinical quality language specification, we build out modelinfo that is based on the version updates of QDM. So whenever QDM releases a new version, we create a new XSD for that QDM and then updated the modelinfo based on that XSD.

Lisa Anderson: Thanks, Bryn. So our next question comes from Brian at Campbell County, it keeps cutting off. Somewhere. Help? On slide 38, when it states, "One day after day of", does that start at 00:00 midnight? Or does it count back from a specific time? So we are going back to slide 38, here we go.

Bryn Rhodes: So the way this actually results in the SQL, sorry, the way this actually comes out in the ELM is that this is the end of the relevant period is between start of the qualifying relevant period and the start of the qualifying relevant period plus one day, where those comparisons are actually performed at the day. And so it's not that it counts backward from zero, it's that the comparisons actually ignore the time component.

Lisa Anderson: Great, thanks, Bryn. Our next question comes from Diana at Kettering Health. When these CQL statements interpret into SQL, do they take performance into consideration? In other words, would they generate a good execution plan?

Bryn Rhodes: So the short answer is yes. The longer answer is, the CQL retrieve is specifically designed to correspond to the most likely indexes in our relational representation of that content. The fact that the SQL that would be the result of the output has filters on the indexed access path to that data means that the SQL compiler in your target environment will take advantage of those access paths as part of its query optimization planning for executing a query. Does that help?

Lisa Anderson: Great. Thanks, Bryn. Our next question comes from Edgardo at Infomedica. Will the measures continue to be tied to a specific version for the future?

Bryn Rhodes: So, just like QDM updates, until CQL goes normative we continue to improve the specification and incorporate feedback from implementation. We are targeting a normative ballot sometime next year. But until the specification goes normative, it's likely that the specifications will continue to use the most current version. That's just a likely and that's just my opinion. If that makes sense.

Lisa Anderson: Thanks, Bryn. All right, our next question comes from Aetna. What is the significance of value sets file for each measure? I see value sets file is a must while executing measure using QDM-based execution engine whereas FHIR-based execution engine does not need value sets for each measure. So, want to understand the significance of value sets file.

Bryn Rhodes:	I need clarification on that. I'm not sure what that question is referring to as far as value sets file. We don't have anything in the ECQM packages that's called value sets file. I wonder if there's a secondary processing step that's happening or something? I'm not sure what that's referring to, I am sorry.
Lisa Anderson:	Great. Thanks, Bryn. So whoever put that question in, if you could clarify that in an additional question we'll get to it at the end. Next question comes from Edgardo at Infomedica again. It says, "The 1.3.10 release of the CQL to ELM translator has an issue that results in an incorrect translation error when attempting to use any of the CONVERTS TO () operators. When is the correction going to be available?
Bryn Rhodes:	There is currently a 1.3.17 release that I'm fairly sure addresses that issue but I can double-check. There's also 1.4.6 release.
Lisa Anderson:	Great. Thanks, Bryn. Our next question comes from Mark. Is it recommended to evaluate a CQM for one client at a time? It seems like that's what the SQL example showed.
Bryn Rhodes:	So you can do that either way. I've seen approaches where you would take a pipelined approach and evaluate for each patient individually. But I've also seen the approach where you could take the SQL and translate it to a population-level query. In doing so, you would have to make sure that you included filters for all the relationships within the query. So when you move from a patient-specific rendering of that SQL, to the population-level rendering, you have to make sure that if it talks about medications you're linking it to the patient. But it's certainly possible to express the SQL in terms of the population even though it's expressed patient-specific within the eCQM. And then if you have a population-level expression, obviously your query planner and your query optimization strategy can take that into account, potentially get significantly better performance by using a population-level approach. So both approaches are supported.
Lisa Anderson:	Great. Thanks, Bryn. Our next question comes from Prime Clinical Systems. How would you implement FIRST and LAST, implemented in a neat way in SQL? Specifically in Oracle?
Bryn Rhodes:	So Oracle, you would probably use ROWNUM. Yeah, that would work. You would use ROWNUM in Oracle. In Microsoft SQL Server you'd use a TOP, you know, SELECT TOP * and then in Oracle, if I remember right, that would be a ROWNUM.
Lisa Anderson:	Great. Thanks, Bryn. Our next question comes from Deb at Meditech. For the latest CQL release, SQ3, is there an area to download a PDF of the specifications along with the ELM schema definitions?
Bryn Rhodes:	So part of what we did between the SQ2 and SQ3 release of CQL was to make it a web-based specification, so there's not a single PDF that contains all of the CQL specification. But you can download the entire specification as a local website, if you want to have a local copy of it. On the Downloads page there's a link for the entire specification.

Lisa Anderson: Great, thank you. Our next few questions come in from David at CVS/Aetna. The model type example in ELM's slide showed retrievable = TRUE. Can you give an example of a use case were something not retrievable and why it would need to be marked as such?

Bryn Rhodes: Yes. So for example, in the FHIR model, the patient has elements that are of type ADDRESS. So we needed a way to talk about the ADDRESS type, but there is no ADDRESS table. Addresses are just data types that provide reuse of structure within the definitions but don't actually correspond to anything persisted on its own. So in those cases we use a non-retrievable data type or class type to define that structure.

As far as an example of context other than patient and where that might be used? Let's say you want to run a measure on the number of beds in a hospital. Your context would be either location or organization, something like that, and your criteria would be expressed in terms of the hospital rather than in terms of any given patient. Maybe you're interested in number of staff at a clinic, those types of measures.

And then for translating CQL to ELM, where are these translators and how do we access them? So on the Resources page, let me go back to that slide. On the Resources page, the Github tool's repository, Clinical Quality Language, the CQL to ELM translator is also available on Maven, if you're using a Java environment you can just include those and get them directly from the Maven central repository. Is that all of the questions there? Yes.

Lisa Anderson: Yes, thank you. Our next question comes from Brian at Health Catalyst, and says, "Why not simply develop the measures in SQL? This seems overly complex for little value."

Bryn Rhodes: So SQL does not have the operation that we need for the most common use cases. The interval operations is the first thing, and the use of quantity as a first class element, ratios as a first class element, and the ability to reference terminology. So when we first set out on this road, that was one of the things we considered very strongly, was why not just use SQL? That was the reason. Once you take those things into account and you actually write SQL that enables the kinds of queries you're using in measure development and completely specify that, the SQL is much more involved than the CQL. There's a lot to the use of clinically focused elements that SQL just doesn't support.

Lisa Anderson: Great, thank you. Our next question comes from Greg at Pro Healthcare. "I was a bit late to start so I apologize if this was covered. But is the CQL version intended to replace the QRD version for both TJC and CMS?"

Bryn Rhodes: I'm not sure I understand the question. [crosstalk 00:18:46]

Lisa Anderson: Yeah so I think they're asking is the CQL version the measure replacing the QRDA?

Bryn Rhodes: Well, I mean, we use QRDA to report CQL measures, so QRDA was updated alongside HQMF to able to use CQL-based measures to report. But it's not intended to replace QRDA because it's not the same kind of specification. CQL is a query language specification, QRDA is a document reporting specification.

Lisa Anderson: Great. Thanks, Bryn. Our next question comes from BWH. Is OVERLAPS equivalent to DURING?

Bryn Rhodes: No. So OVERLAPS means, is there any overlap between interval A and interval B, where DURING means, is the interval entirely contained within interval B.

Lisa Anderson: Great. Thanks, Bryn. Our next question comes from Menish at IVM. On slide 38, he has a question about the time phrases with offsets. Where in CQL does it explain what time difference method to use for items provided in three and four on the slide. There has been a lot of confusion with timing calculations since it seems there may be several ways to calculate time difference, each yielding different results. It goes on to say he's created several GR tickets and been provided guidance which conflicts with Appendix H of HL7 CQL specification, and according to this example one on slide 38, he thinks item three will be TRUE even if labor.authordatetime is 24 hours and 59 minutes.

Bryn Rhodes: So I would need to dig into that specifically. Within the SQL specification there's a chapter on translation symantecs that talks about how timing phrases are converted into the canonical ELM representations, how those timing phrases are represented. And then there are the timing and calculation examples that he's referring to in Appendix H. So you're seeing conflicting results there.

Lisa Anderson: Okay, so we can follow up with that in the Q&A that we do later after he's reviewed it further. And the next question comes from GC Winters at Conduit. In the old HQMF framework, the measure representation could support proportion, interval and statistical measures. What capability is in, or will be in, I think it's CQL to incorporate statistical functionality.

Bryn Rhodes: So currently CQL defines a full complement of statistics operators, the same that you would generally find in any base implementation of SQL. So MIN, MAX, standard deviations, variants, those kinds of basic statistics operators. In addition, the [inaudible 00:22:40] specification does support the inclusion of external, so you can define an operation that you're bringing in an external library to provide the implementation for. So in short, we support already more than the HQMF framework supported, and with the potential to add additional functionality if it's needed.

Lisa Anderson: Great. Thanks, Bryn. We are running close to session end time. Any questions unanswered today will be addressed in a follow-up Q&A document. Oop, it didn't change. I'm sorry, there we go.

A few closing remarks before we end the session. As a reminder, the slides are available for download now. Please visit the Expert to Expert landing page which includes presentation replays, slide decks and Q&A's for all webinars in the series. Although this is the last scheduled Expert to Expert webinar for 2019, we are launching the 2019 Proven Practices webinar series starting in August. This series highlights solutions and tips from expert Proven Practice contributors for peer-to-peer learning regarding eCQM utilization for performance improvement. Registration is now open for sessions on August 27, September 12, and

September 24. You can click on the link in the slides, or visit the webinar series landing page at the provided link to register for all three of these sessions.

A survey link will be emailed to participants tomorrow. If you qualify for CE credits, complete the survey and include the email to which you would like your certificate sent. When the evaluation closes two weeks from today, all those eligible for CEs will receive an email with a link to a PDF certificate.

Thank you, Bryn, for presenting today, and thanks to all of you who listened in. Have a great day.