Clinical Quality Language (CQL) Basics

Shanna Hartman
Centers for Medicare & Medicaid Services

Deborah Krauss
Centers for Medicare & Medicaid Services (CMS)

Bryn Rhodes
ESAC, Inc.
CQL Basics: Agenda

• Welcome and Background
• CQL Language Tour
  ▪ Accessing Clinical Data
  ▪ Using Queries
  ▪ Computation
  ▪ Date/Times, Intervals, and Timing Phrases
  ▪ Combining Queries
  ▪ Aggregate Computation
CQL Basics: Evolving eCQM Standards

Definitions:
HQMF – Health Quality Measure Format
CQL – Clinical Quality Language
QDM – Quality Data Model

Updated 12/8/2017
CQL Basics: Proposed Timeline For Updating Standards

Measure Development – Expected Timelines

<table>
<thead>
<tr>
<th>Year</th>
<th>2016</th>
<th>2017</th>
<th>2018</th>
<th>2019</th>
<th>2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Measure Structure – HQMF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>CQL-based HQMF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>Measure Report – QRDA Category I (individual report), QRDA Category III (aggregate report)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- CQL in MAT and Bonnie Test Environments
- MAT and Bonnie Tools updated for CQL
- Develop Measures
- Implement
- Report

Updated 12/8/2017
CQL Basics: Clinical Quality Language (CQL)

• Health Level 7 (HL7) standard designed to:
  ▪ Enable automated point-to-point sharing of executable clinical knowledge
  ▪ Provide a clinically focused, author-friendly, and human-readable language

• Currently a Draft Standard for Trial Use (DSTU) publication
CQL Basics: Accessing Clinical Data

- Clinical data models contain “statements” of clinical data, e.g.,
  - Patient had a routine check-up on April 3rd
  - Patient was administered an antibiotic
  - Patient had an appendectomy
  - Patient was diagnosed with Type II Diabetes
CQL Basics: Accessing Clinical Data (cont’d)

• Statements can be organized into different types, e.g.,

  Patient had a routine check-up on April 3rd
  Patient was administered an antibiotic
  Patient had an appendectomy
  Patient was diagnosed with Type II Diabetes

  Encounter
  Medication, Administered
  Procedure, Performed
  Diagnosis
CQL Basics: Accessing Clinical Data (cont’d)

• Within these types, different kinds of statements can be represented with codes from code systems.

• Within Encounter, e.g.,

  Patient had a routine check-up on April 3rd
  Patient was admitted to the ED
  Patient was admitted for elective surgery

  SNOMEDCT|185349003
  SNOMEDCT|4525004
  SNOMEDCT|8715000
• The codes describing different kinds of statements are then grouped with *value sets*, allowing classes of specific kinds of statements to be referenced, e.g.,

Patient had a routine check-up on April 3rd
Patient was admitted to the ED
Patient was admitted for elective surgery

Encounter Inpatient
Emergency Department Visit
Elective Encounter
This *retrieve* expression results in only the highlighted encounters, because they have codes that match the “Inpatient” value set.
CQL Basics: Filtering with Where

Introducing an alias, Encounter in this case, allows you to reference elements of the statement for further filtering using a where clause. The above filter results in only the highlighted rows, because they are during the “Measurement Period” (2015 year).
The **with clause** allows you to define relationships with other data based on specific criteria. In this case, only the October encounter is returned, because it has a LabTest that resulted during the encounter.
Statements can also be excluded based on relationships using the `without clause`. In this case, the October encounter is excluded, because it has a LabTest that resulted during the encounter.
You can return only a subset of the elements in a statement using the *return clause*. In this case, only the relevantPeriod element is returned.
You can order the results using the *sort clause*. In this case, the result is sorted by the start of the `relevantPeriod` element, ascending.
CQL Basics: Naming Expressions

You can name any expression so that it can be reused in subsequent expressions using the `define declaration`. In this case, the result of “Sorted Encounters” is now the same as the result of the defined expression.

```
define "Sorted Encounters":
    ["Encounter, Performed": "Inpatient"] Encounter
    sort by start of relevantPeriod
```
The results of retrieves and queries are lists, so you can pick items based on order using `First()` and `Last()`.

Because “Sorted Encounters” is ordered by the start of the `relevantPeriod`, `First()` returns the oldest encounter, while `Last()` returns the most recent.
You can also use the `indexer ([ ])` to pick out any item by its index in the list.

Indexes in CQL are 0-based, so the first item is index 0, the second item is index 1, and so on.

Note that whenever you’re performing operations that rely on the order of elements in the list, be sure to use a `sort clause` to get the appropriate ordering.
CQL Basics: Strings

CQL supports strings using single-quotes (').

You can escape characters (such as single-quotes, tabs, carriage returns, and line feeds, using standard escape characters.

CQL supports string comparison for all the comparison operators (=, !=, <=, >=, <, and >).

String comparison is case-sensitive, and based on the Unicode value of each character.

'John Doe'
'John O\'Mally'
'John Doe' = 'john doe'  // false
'Deer' < 'Doe'         // true
CQL Basics: Numbers and Calculation

CQL supports *Integers* (whole numbers), and *Decimals*.

In calculations and comparisons, integers are implicitly converted to decimals when necessary.

Comparison of decimals ignores precision.

CQL uses standard mathematical operator precedence.

Use parentheses to force precedence.

Division in CQL always returns a decimal, use *div* to perform integer division.

The *mod* operator returns the remainder of an integer division.
CQL Basics: Rounding and Exponents

CQL supports standard rounding, 0.5 and above rounds up, 0.4 and below rounds down. The second argument, if supplied, specifies the precision of the result.

- `Truncate()` returns the integer component of a decimal.
- `Floor()` returns the greatest integer less than a decimal.
- `Ceiling()` returns the least integer greater than a decimal.

CQL supports exponents and roots with ^.

- Logarithms to a given base use `Log()`.
- Natural logarithms use `Ln()` and `Exp()`.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Round(5.5)</code></td>
<td>6</td>
</tr>
<tr>
<td><code>Round(5.55, 1)</code></td>
<td>5.6</td>
</tr>
<tr>
<td><code>Truncate(5.5)</code></td>
<td>5</td>
</tr>
<tr>
<td><code>Truncate(-5.5)</code></td>
<td>-5</td>
</tr>
<tr>
<td><code>Floor(5.5)</code></td>
<td>5</td>
</tr>
<tr>
<td><code>Floor(-5.5)</code></td>
<td>-6</td>
</tr>
<tr>
<td><code>Ceiling(5.5)</code></td>
<td>6</td>
</tr>
<tr>
<td><code>Ceiling(-5.5)</code></td>
<td>-5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>5 ^ 2</code></td>
<td>25</td>
</tr>
<tr>
<td><code>25 ^ 0.5</code></td>
<td>5</td>
</tr>
<tr>
<td><code>Log(25, 5)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>Log(5, 25)</code></td>
<td>0.5</td>
</tr>
<tr>
<td><code>Ln(10)</code></td>
<td>2.30258209288405</td>
</tr>
<tr>
<td><code>Exp(2.30258509288405)</code></td>
<td>10</td>
</tr>
</tbody>
</table>
Quantities in CQL are a number followed by a UCUM unit.

CQL supports arithmetic and comparison operators for quantities.

Implementations are required to respect units, but not necessarily conversions between units.

Arithmetic operators, in particular, must return quantities with appropriate units, but not necessarily converted.

An implementation may throw a run-time error for an unsupported unit conversion operation.
CQL Basics: DateTime and Time

CQL supports DateTime, a point-in-time on the Western calendar, specified with integers for year, month, day, hour, minute, second, and millisecond, plus a timezone.

CQL also supports Time, a point-in-time in a 24-hour period, specified with integers for hour, minute, second, and millisecond, plus timezone.

Both DateTime and Time support partial values, but only for trailing precisions (i.e., if you specify a day, you must also specify a year and month.

If not supplied, timezone is assumed based on the evaluation context.
CQL Basics: DateTime and Time (cont’d)

CQL also supports construction of dateTime and Time values as expressions.

You can use `date from` to extract the date (with no time components) from a dateTime value.

You can use `time from` to extract the time from a dateTime value.

You can use the name of a component to extract it from a dateTime or Time value.

`Now()`, `Today()`, and `TimeOfDay()` return the dateTime, Date, and Time, respectively, of the evaluation context.
You can compare dateTime and Time values using the standard comparison operators: =, !=, <=, >=, <, and >.

You can also perform precision-based comparisons using same as, before/after of, and same or before/after.
CQL Basics: Date Arithmetic

CQL supports time-valued quantities with the name (singular or plural) of the precision as the unit.

UCUM units can also be used (with quotes).

Durations can then be added to or subtracted from dateTime and Time values, with the expected semantics for durations with variable days such as years and months.

Updated 12/8/2017
The *duration in..between* operator determines the number of whole periods between two dateTime or Time values.

This expression returns 0 because there are no whole months between the two dates.

The *difference in..between* operator determines the number of boundaries crossed between two dateTime and Time values.

This expression returns 1 because 1 month boundary was crossed between the two dates.
CQL Basics: Intervals

CQL supports Intervals for numbers and date/time values.

Intervals use standard mathematical notation to indicate open and closed (i.e., whether the endpoint is included in (closed) or excluded from (open) the interval).

You can test for membership with `contains` and `in`, and you can determine the boundaries of an interval using `start of` and `end of`.

You can determine the width of an interval using `width of`.

```
Interval[3, 5)    // 3, 4
Interval[3.0, 5.0) // >= 3.0, < 5.0
Interval[@2014-01-01T00:00:00.0, @2015-01-01T00:00:00.0]
```

```
Interval[3, 5) contains 4    // true
4 in Interval[3, 5)    // true

start of Interval[3, 5) // 3
end of Interval[3, 5)  // 4
width of Interval[3, 5) // 2
```
# CQL Basics: Comparing Intervals

<table>
<thead>
<tr>
<th>Relation</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>before</td>
<td><img src="before.png" alt="Diagram" /></td>
</tr>
<tr>
<td>meets before</td>
<td><img src="meets_before.png" alt="Diagram" /></td>
</tr>
<tr>
<td>overlaps before</td>
<td><img src="overlaps_before.png" alt="Diagram" /></td>
</tr>
<tr>
<td>includes</td>
<td><img src="includes.png" alt="Diagram" /></td>
</tr>
<tr>
<td>starts</td>
<td><img src="starts.png" alt="Diagram" /></td>
</tr>
<tr>
<td>same as (=)</td>
<td><img src="same_as.png" alt="Diagram" /></td>
</tr>
<tr>
<td>ends</td>
<td><img src="ends.png" alt="Diagram" /></td>
</tr>
<tr>
<td>included in (during)</td>
<td><img src="included_in.png" alt="Diagram" /></td>
</tr>
<tr>
<td>overlaps after</td>
<td><img src="overlaps_after.png" alt="Diagram" /></td>
</tr>
<tr>
<td>meets after</td>
<td><img src="meets_after.png" alt="Diagram" /></td>
</tr>
<tr>
<td>after</td>
<td><img src="after.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Updated 12/8/2017
CQL also supports timing phrases that make it easier to express precise relationships between intervals using natural language.

The *before* and *after* operators can have a prefix of *starts* or *ends*, and a suffix of *start* or *end*. For example,
The *before* and *after* operators can also take an offset that indicates how far away a given relationship should be.

This offset can be absolute, indicating that the boundary of the interval must be on the offset, or it can be relative, indicating that the boundary must be at least on the offset.
You can also specify a range for the boundary relationship using the *within..of* operator.

Interval\(X\) starts within 3 days of start Interval\(Y\)

starts within 3 days of
You can test for membership of items in a list using the *contains* and *in* operators.

You can compare lists using equality (=), and the *includes* and *included in* operators.
CQL Basics: Union

The *union* operator combines two lists, eliminating duplicates.

\[
\begin{array}{ccc}
1 & 4 & 1 \\
2 & 5 & 2 \\
3 & 6 & 3 \\
4 & 7 & 4 \\
5 & 8 & 5 \\
\end{array}
\]

\[
X \text{ union } Y
\]

NOTE: In the current specification (CQL 1.1), union does not eliminate duplicates, so a distinct must be used. However, this is a DSTU comment to change this behavior to support the more intuitive duplicate elimination semantics.
The *intersect* operator results in a list containing only the elements that appear in both lists.
The *except* operator results in a list containing only the elements of the first list that are not present in the second list.

\[
\begin{array}{c}
1 & 4 \\
2 & 5 \\
3 & 6 \\
4 & 7 \\
5 & 8 \\
\end{array}
\]

\[
\begin{array}{c}
1 \\
2 \\
3 \\
\end{array}
\]

X except Y
CQL Basics: Resources

• CQL Specification - CQL Release 1, Standard for Trial Use (STU) 2

• eCQI Resource Center
  ▪ CQL Space, including the QDM v5.3 and v5.3 Annotated
    • [https://ecqi.healthit.gov/cql](https://ecqi.healthit.gov/cql)
  ▪ Check the eCQI Resource Center Events page and CQL Educational Resources page for more information
    • [https://ecqi.healthit.gov/ecqi/ecqi-events](https://ecqi.healthit.gov/ecqi/ecqi-events)
    • [https://ecqi.healthit.gov/cql/cql-educational-resources](https://ecqi.healthit.gov/cql/cql-educational-resources)

Updated 12/8/2017
CQL Basics: Resources (cont’d)

• CQL Formatting and Usage Wiki

• CQL GitHub Tools Repository
  ▪ [https://github.com/cqframework/clinical_quality_language](https://github.com/cqframework/clinical_quality_language)

• Measure Authoring Tool
  ▪ [https://www.emeasuretool.cms.gov/](https://www.emeasuretool.cms.gov/)

• Bonnie Testing Tool
  ▪ [https://bonnie.healthit.gov/](https://bonnie.healthit.gov/)

• To submit an issues ticket for CQL, please visit the ONC JIRA site
  ▪ [https://oncprojecttracking.healthit.gov/support/projects/CQLIT](https://oncprojecttracking.healthit.gov/support/projects/CQLIT)