smile DIGITAL HEALTH   HL7 International

CQL Basics Cheat Sheet

CQL is a Health Level Seven® Standard for the expression of clinical knowledge.

Values

Type	Description	Example
Any	The null literal	null
Boolean	The boolean literal	true, false
Integer	Sequences of digits in the range $0..2^{31}-1$	16, -28
Long	Sequences of digits in the range $0..2^{63}-1$	16000000000L, -28000000000L
Decimal	Sequences of digits with a decimal point, in the range 0.0.. $(10^{28}-1)/10^{8}$	100.015
String	Strings of any character enclosed within single-ticks (')	'pending' 'John Doe' 'complete'
Date	The at-symbol (@) followed by an ISO-8601 compliant representation of a datecalendar, irrespective of the time of day	@2014-01-25
DateTime	The at-symbol (@) followed by an ISO-8601 compliant representation of a datetime	@2014-01-25T14:30:14.559
Time	The at-symbol (@) followed by an ISO-8601 compliant representation of a time	@T12:00
Quantity	An integer or decimal literal followed by a datetime precision specifier, or a UCUM unit specifier	6 'gm/cm3' 80 'mm[Hg]' 3 months
Ratio	A ratio of two quantities, separated by a colon (:)	1:128 5 'mg' : 10 'mL'
Code	Construct consistent with the way terminologies are typically represented	Code '66071002' from "SNOMED-CT" display 'Type B viral hepatitis'

Concept	Construct to specify multiple terminologies used to code for the same concept	Concept {
		  Code '66071002' from "SNOMED-CT",
		  Code 'B18.1' from "ICD-10-CM"
		} display 'Type B viral hepatitis'

Tuple	Structured values that contain named elements, each having a value of some type	Tuple {
		    Name: 'Patrick',
		    DOB: @2014-01-01,
		    Address: Tuple { Line1: '41 Spinning Ave', City: 'Dayton', State: 'OH' },
		    Phones: { Tuple { Number: '202-413-1234', Use: 'Home' } }
		  }

List	A collection of values of any type	{ 1, 2, 3, 4, 5 } [Condition: code in "Acute Pharyngitis"]
Interval	Set of values between two boundaries that can be inclusive ([]) or exclusive (())	Interval[3, 5) // 3 and 4, but not 5 Interval[@2014-01-01, @2015-01-01) // same as Interval[@2014-01-01, @2014-12-31]

Identifiers

Type	Description	Example
Simple	Any alphabetical character or an underscore, followed by any number of alpha-numeric characters or underscores	Foo1
Delimited	any sequence of characters enclosed in backticks (`)	`Encounter, Performed`
Quoted	Any sequence of characters enclosed in double-quotes (")	"Inpatient Encounters"
Qualified	Identifiers can be combined using the qualifier operator (.)	Common.ConditionsIndicatingSexualActivity

Symbols

Symbol	Description
:	Definition operator, typically read as "defined as". Also used to separate the numerator from denominator in Ratio literals
( )	Parentheses for delimiting groups, as well as specifying and passing function parameters
[ ]	Brackets for indexing into lists and strings, as well as delimiting the retrieve expression
{ }	Braces for delimiting lists and tuples
< >	Angle-brackets for delimiting generic types within type specifiers
.	Period for qualifiers and accessors
,	Comma for delimiting items in a syntactic list
+ - * / ^	Arithmetic operators for performing calculations

Comparison Symbols

Symbol	Description
=	Equal
!=	Inequal
~	Equivalent
!~	Inequivalent
<=	Less than or Equal to
<	Less than
>	Greater than
>=	Greater than or Equal to

Comments

Type	Example
Single-line	define "Foo": 1 + 1 // This is a single-line comment
Multi-line	/* This is a multi-line comment Any text enclosed within is ignored */

Named Expressions

Type	Example
Statement	define SimpleStatement: 'This is simple!'

Function	define function MostRecent(observations List):
	  Last(
	    observations O
	      sort by issued
	  )

Declarations

Construct	Description	Example
Library Syntax	Header information for the library, including the name and version, if any	library AlphoraCommon version '1.0.0'
Using Syntax	Data model information, specifying that the library may access types from the referenced data model	using FHIR version '4.0.1'
Include Syntax	Referenced library information, specifying that the library may access constructs defined in the referenced library	include FHIRCommon called FC
CodeSystems	Codesystem information, specifying that logic within the library may reference the specified codesystem by the given name	codesystem "LOINC": 'http://loinc.org'
ValueSet	Valueset information, specifying that logic within the library may reference the specified valueset by the given name	valueset "Encounter Inpatient": 'http://cts.nlm.nih.gov/fhir/ValueSet/2.16.840.1.113883.3.666.5.307'
Code	Code information, specifying that logic within the library may reference the specified code by the given name	code "Blood Pressure Panel": '85354-9' from "LOINC"
Concept	Concept information, specifying that logic within the library may reference the specified concept by the given name	concept "Blood Pressure Codes": { "Blood Pressure Panel" }
Parameter	Parameter information, specifying that the library expects parameters to be supplied by the evaluating environment	parameter "Measurement Period" default Interval[@2013-01-01, @2014-01-01)
Context	Specifies the overall context, such as Patient or Practitioner, to be used in the statements that are declared in the library	context Patient
Define	The basic unit of logic within a library, a define statement introduces a named expression that can be referenced within the library, or by other libraries	define "Inpatient Encounters": [Encounter: "Encounter Inpatient"] Encounter where Common.NormalizePeriod(Encounter.period) ends during day of "Measurement Period" define "Most Recent Blood Pressure Labs": MostRecent([Observation: value in "Blood Pressure Codes"])

Function	A named expression that is allowed to take any number of arguments, each of which has a name and a declared type	define function MostRecent(observations List<Observation>):
		  Last(
		    observations O
		      sort by issued
		  )

Retrieve (Primary Source)

Concept	Description	Example
Clinical Statement	Determines the structure of the data that is returned by the retrieve, as well as the semantics of the data involved	[Encounter]
Filtering With Terminology	The retrieve expression allows the results to be filtered using terminology, including valuesets, code systems, or by specifying a single code	[Condition: severity in "Acute Severity"]

Query

Clause	Operation	Example

Relationship (with/without)	Allows relationships between the primary source and other clinical information to be used to filter the result	[Encounter: "Ambulatory/ED Visit"] E
		  with [Condition: "Acute Pharyngitis"] P
		    such that P.onsetDateTime during E.period
		      and P.abatementDate after end of E.period

Where	The where clause allows conditions to be expressed that filter the result to only the information that meets the condition	[Encounter: "Inpatient"] E where duration in days of E.period >= 120
Return	The return clause allows the result set to be shaped as needed, removing elements, or including new calculated values	[Encounter: "Inpatient"] E return duration in days of E.period
Sort	The sort clause allows the result set to be ordered according to any criteria as needed	[Encounter: "Inpatient"] E sort by start of period

smile DIGITAL HEALTH

HL7 International

CQL Advanced Cheat Sheet

CQL is a Health Level Seven® Standard for the expression of clinical knowledge.

Strings and Identifiers

Type	Use	Example
Single Quotes	1. Text	`define "string literal": 'hello'`

	2. Code system and value set URI's (OID)	`valueset "Observation Status":`
		`'http://hl7.org/fhir/ValueSet/observation-status'`

	3. Version identifiers	`valueset "Observation Status":`
		`'http://hl7.org/fhir/ValueSet/observation-status'`
		`version '6.0.0'`

	4. Code declarations	`code "code": '123' from "codesystem identifier"`
	5. Units	`define "quantity": 12.0 'L'`
Quoted Identifiers	A code system declaration	`"SNOMED CT"`
	A Defined Code	`"Inpatient Encounters"`

Null and String Operators

Operator	Type	Example
Nullological Operators	*Null* Test is used to test whether an expression is `null`	`X is null`
		`x is not null`
	Coalesce operator is used to return the first non-null result among two or more expressions	`Coalesce(X, Y, Z)`
String Operators	*Length* Operator is used to determine the length of string	`Length(X)`
	PositionOf operator is used to determine the position of a string and will return the index	`PositionOf('cde', 'abcdefg') // returns 2`
	Combine operator is used to combine a list of strings	`Combine({ 'ab', 'cd', 'ef' }) // returns 'abcdef'`

	Split operator is used to split a list of strings	`Split('completed;refused;pending', ';')`
		`// returns { 'completed', 'refused', 'pending'}`

Interval Operators

Type	Example
General Interval Operators	`point from Interval[3, 3] // returns 3`
	`width of Interval[3, 5] // returns 2`
	`end of Interval[3, 5) // returns 4`
Comparing Intervals	`Interval[2, 6] same as Interval[2, 6] // returns true`
	`Interval[1, 5] before Interval[6, 10] // returns true`
	`Interval[1, 7] overlaps before Interval[5, 10] // returns true`
	`Interval[1, 5] meets before Interval[6, 10] // returns true`
Timing Operations on Intervals	`Date(2014) same year as Date(2014, 7, 11)`

	`["Encounter": "Inpatient"]`
	`  where Encounter.period during "Measurement Period"`

	`Date(2014, 4) same month or before Date(2014, 7, 11)`
	`Date(2014, 4) same month or after Date(2014, 7, 11)`

	`starts within 3 days of start (2014, 7, 11)`

	`[Condition: "Other Female Reproductive Conditions"] C`
	`  where Interval[C.onsetDate, C.abatementDate] overlaps "Measurement Period"`

Conditional Expressions

Type	Description	Example
if	The if expression allows one single condition to be selected	`if Count(X) > 0 then X[1] else 0`

case	The case expression allows multiple conditions to be tested.	`case`
		`  when X > Y then X`
		`  when Y > X then Y`
		`  else 0`
		`end`

Type Operators

Operator	Description	Example

as	Allows the result of an expression to be cast as a given target type	`define "Former smoker observation":`
		`  "Most recent smoking status observation" O`
		`    where (O.value as CodeableConcept) ~ "Former Smoker"`

is	Allows the type of a result to be tested	`define "Patient is Active":`
		`  Patient P`
		`    where P.active.value is true`

Logical Operators

Description	Example
Only takes boolean values as input and returns boolean values.	`AgeInYears() >= 18 and AgeInYears() < 24`

	`define "Blood Pressure Measured":`
	`  exists("Systolic Blood Pressure Observation")`
	`    and exists("Diastolic Blood Pressure Observation")`

	`define "Fecal Occult Blood Test During Measurement Period":`
	`  AC.MostRecent([Observation: "Fecal Occult Blood Test (FOBT)"]).effective during "Measurement Period"`

Advanced Query Clauses

Clause	Operation	Example

Let	Introduces content that can be referenced within the scop of the query without impacting the type of the result, unless referenced within a return clause	`define "Medication Ingredients":`
		`  "Medications" M`
		`    let ingredients: GetIngredients(M.rxNormCode)`
		`    return ingredients`

Aggregate	Allows an expression to be repeatedly evaluated for each element of a list. *Important Note: The aggregate clause is a new feature of CQL 1.5, and is trial-use.*	`define FactorialOfFive:`
		`  ({ 1, 2, 3, 4, 5 }) Num`
		`    aggregate Result starting 1: Result * Num`

From	Allows for the simple expression of complex relationships between different sources/data (multi-source query)	`define "Encounters with Warfarin and Parenteral Therapies":`
		`  from "Encounters" E,`
		`    "Warfarin Therapy" W,`
		`    "Parenteral Therapy" P`
		`  where W.effectiveTime starts during E.period and P.effectiveTime starts during E.period`

List Operators

Operation	Description	Example
Operating on Lists	If a list contains a single element, the *singleton* from operator can be used to extract it.	`singleton from { 1 }`
	To obtain the *index* of a value within the list	`IndexOf({'a', 'b', 'c' }, 'b') // returns 1`
	To *count* (obtain) the number of elements in a list	`Count({ 1, 2, 3, 4, 5 }) // returns 5`

	Membership in lists can be determined using the *in* operator and its inverse, *contains*	`{ 1, 2, 3, 4, 5 } contains 4`
		`4 in { 1, 2, 3, 4, 5 }`

	To test whether a list *exists* or contains any elements	`exists( { 1, 2, 3, 4, 5} )`
	The *First* and *Last* operators can be used to retrieve the first and last elements of a list.	`First({ 1, 2, 3, 4, 5 }) // returns 1`

Comparing Lists	*include* returns true if if every element in list Y is also in list X	`{ 1, 2, 3, 4, 5 } includes { 5, 2, 3 } // returns true`
		`{ 1, 2, 3, 4, 5 } includes { 4, 5, 6 } // returns false`

	included in returns true if every element in list X is also in list Y	`{ 5, 2, 3 } included in { 1, 2, 3, 4, 5 } // returns true`
		`{ 4, 5, 6 } included in { 1, 2, 3, 4, 5} // returns false`

	properly includes returns true if every element in list Y is also in list X, and list X has more elements than list Y	`{ 1, 2, 3 } properly includes { 1, 2, 3, 4, 5} // returns true`
		`{ 1, 2, 3} properly includes { 1, 2, 3 } // returns false`

	properly included in return true if if every element in list X is also in list Y, and list Y has more elements than list X	`{ 2, 3, 4 } properly included in { 1, 2, 3, 4, 5 } // returns true`
		`{ 1, 2, 3 } properly included in { 1, 2, 3 } // returns false`

Computing Lists	To eliminate duplicates we use *Distinct*.	`Distinct({ 1, 2, 3, 4, 4}) // returns { 1, 2, 3, 4 }`
	To combine lists and eliminate duplicates we use *union*	`{ 1, 2, 3 } union { 3, 4, 5 } // returns { 1, 2, 3, 4, 5 }`
	To only return the elements that are in both lists we use *intersect*.	`{ 1, 2, 3} intersect { 3, 4, 5} // returns { 3 }`
	To flatten lists of lists we use *flatten*.	`flatten { { 1, 2, 3 }, { 3, 4, 5 } } // returns { 1, 2, 3, 3, 4, 5 }`

Arithmetic Operators

Type	Operation	Example

Addition (+) and Subtraction (-)	Performs numeric addition and subtraction of it's arguments	`5 + 10 // returns 15`
		`100 - 5 // returns 95`

Multiply (*) and Divide (/)	Performs numeric multiplication and division of it's arguments	`3 months * 2 months // returns 6 months`
		`12 months / 2 months // returns 6 months`

Truncate ()	Returns the integer component of its argument	`Truncate(12.4) // returns 12`
Round ()	Rounds to the nearest whole value	`Round(123.5) // returns 124`
Floor ()	Rounds to the first integer less than or equal to it's argument (decimal)	`Floor(123.456) // returns 123`
Ceiling ()	Rounds to the first integer greater than or equal to it's argument (decimal)	`Ceiling(123.456) // returns 124`
Convert ()	Converts a value from one type to another	`convert 5000 'g' to 'kg' // returns 5.0 'kg'`
Count ()	Returns the number of elements in it's argument	`Count ({1, 2, 3, 4, 5}) // returns 5`
Sum ()	Returns the sum of values	`Sum({1, 2, 3, 4, 5} // returns 15`