Introduction to the

Constellation Query Language

(Copyright 2008 Clifford Heath)

The Constellation Query Language (**CQL**) is a language for constructing and querying semantic information models. The version described here uses English language keywords and expressions, but these may be substituted for keywords and expressions from another language. Because CQL has an open vocabulary, and because it is designed to resemble natural language, the formal grammar has many ambiguities. Some will become obvious as the syntax rules unfold, and in most cases the valid resolution is described.

The elements of a semantic model are:

- named **concepts** (data types, entity types and their subtypes)
- **fact types** which define the associations of those concepts
- **constraints**, which limit the allowed facts and instances of concepts.

Examples of concepts are Person, Name, Date, Address, Employee. Examples of fact types are “Person has given-Name”, “Person lives at Address”. A constraint is present in “Person has at most one family-Name”.

Name and Date are **data types**. Data types are lexical, meaning that they have values that may be written down, like a number, a name, etc. In CQL, a numeric data type may have an associated unit. A unit is defined either as a fundamental unit, or as a coefficient multiplied by another unit or units, each raised to some integer power.

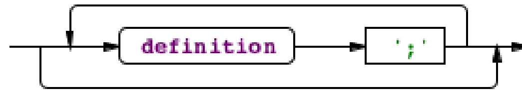
Every fact type involves one or more concepts (most often two), which are said to “**play a role**” in that fact type. Each fact type has one or more **readings** that associate those concepts through a natural-language expression, and each reading must include all the concepts of that fact type. The concepts may occur in any order in each reading, linked by arbitrary words of the natural language, as long as those linking words don’t have a special meaning to CQL in the contexts in which they will appear. For each reading, each concept may be associated with either a leading or a trailing **adjective**. The adjective may be any word that doesn’t have a special meaning, and is especially useful where a fact type might involve the same concept more than once (“Person is friend of other-Person”)

Every **entity type** has some identification scheme. An entity type that has no supertypes is identified by one or more roles played by that entity type. An entity that has a supertype will normally inherit its identification from the first (or primary) supertype, though it may instead define its own identification. The identifying roles must be sufficient to distinguish one instance of this entity type from other instances. A fact type may be named, which allows it to act as a concept, playing roles in other fact types. These **objectified** fact types act as entity types.

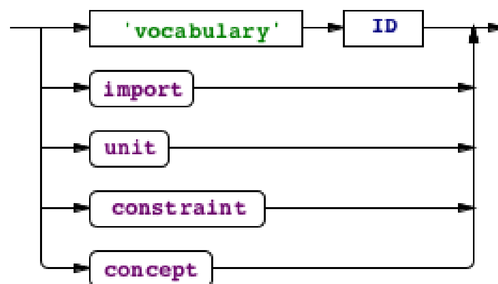
DRAFT

CQL File

A CQL file is a sequence of definitions, each terminated by a semi-colon:

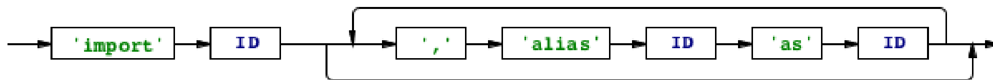


White space and comments like C and C++ are allowed: /* **comment may span lines** */ and // **introduces a comment to end of the current line.** Each CQL file must start with a vocabulary definition.



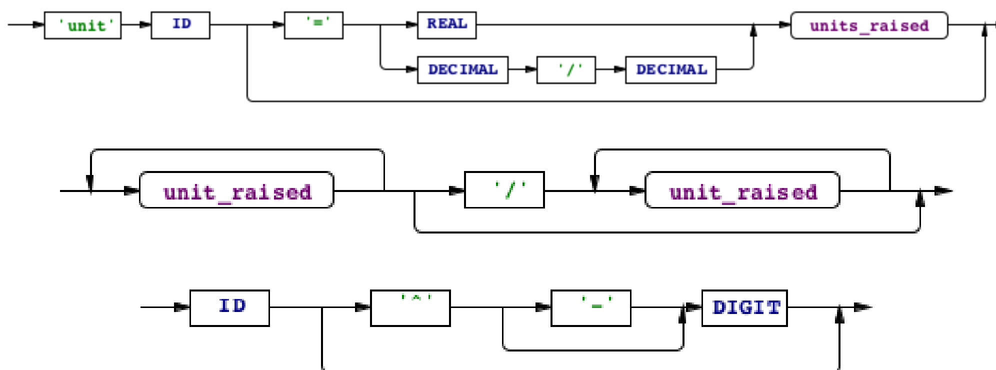
Imports

An import definition imports concept names from another vocabulary, possibly using the **alias** syntax to rename some terms:



Units

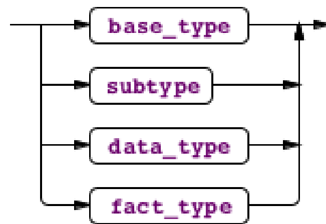
A unit definition defines a new unit identifier in terms of a real number or fraction multiplied by one or more base units, each raised to an integer power:



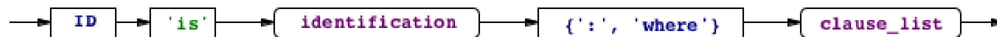
DRAFT

Concepts

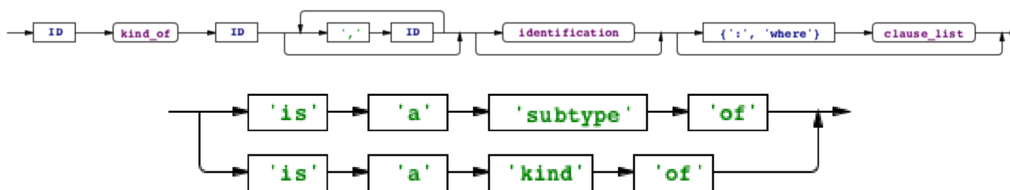
A concept definition is one of the following types, which start with the ID (name of the concept being defined). Names in CQL are case sensitive, and it's conventional practice to use initial capital letters for concept names - this is required in Object Role Modeling but not in CQL. It is however a good way of allowing concept names to be to be distinguished from the same words in lower case, where they may occur in fact type readings.



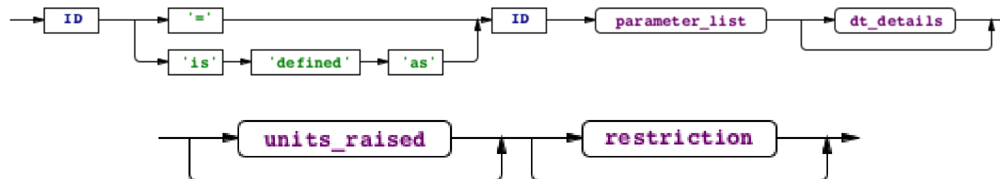
base type:



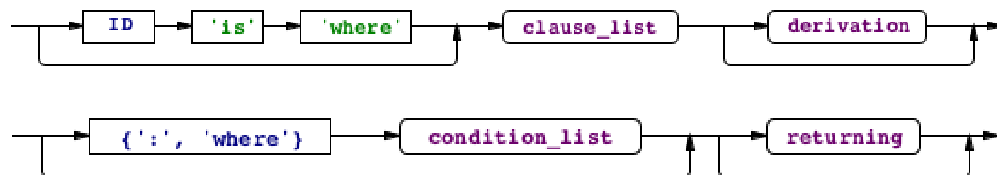
subtype:



data type:



fact type:



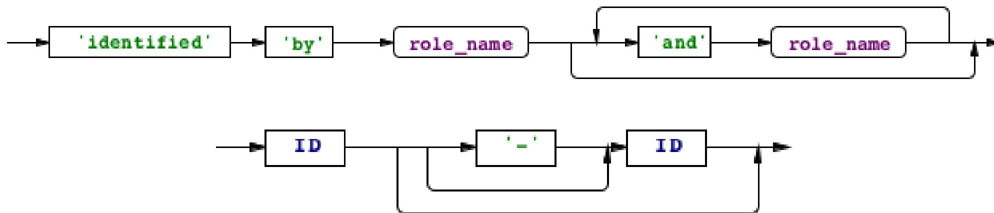
Note that a fact type does not have to be named; it can simply be a reading such as "Person was born at one birth-Place". In the case where no unique quantifier exists in the fact type, or where there are more than two roles, the fact type must have a name. For example, "Directorship is where Person

directs Company”. Technically, where a fact type isn’t named, it isn’t treated as a concept, as it cannot play roles in other fact types. Syntactically though it’s more convenient to discuss these together.

Data types are derived from other data types, and the original data types are predefined in an imported vocabulary. They may refine the supertype by the use of **length** and **scale** parameters, where the supertype allows it. A value restriction might also apply, and these are discussed below.

Entity Type identification

The instances of each entity type are uniquely identified by the combination of their identifying roles. A subtype may inherit its identification from the first supertype, or it may define new identification. When a fact type is named and hence becomes an entity type, it is identified in the same way all fact instances are, by all its roles (or all-but-one, where a uniqueness constraint applies). Because of the use of adjectives, role names may consist of two words (the concept name and the adjective) or a role name defined using the role-name definition syntax discussed below).



All the fact type clauses of an entity type definition must involve the entity type and one of the identifying roles, and no other roles. There is one binary fact type for each identifying role, but there may be more than one reading for each fact type:

> *Person is identified by given Name and family Name:*
>> *Person is called given-Name, given-Name is of Person,*
>> *Person has family-Name, family-Name is of Person;*

Subtypes

An entity type may be declared to be a subtype (or more informally, using the word “kind”) or one or more other entity types, the supertypes:

> *Apple is a kind of Fruit;*

No additional identification is required, as the supertype must already have an identification scheme, and the subtype inherits the identification of its first

DRAFT

supertype, unless an alternate identification scheme is declared. Any subtype may play the roles of all of its supertypes.

Perishable has at most one ShelfLife;
Fruit has one Price per kg;
Apple is a kind of Fruit, Perishable;

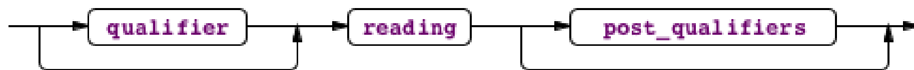
In these fact types, each apple must have a price and may record a shelf life.

Subtyping is itself a fact type, which is useful when subtyping relationships must be constrained.

Fact Types

Fact types are defined by one or more readings extracted from fact type clauses, and all the clauses must have the same set of role players. If an adjective is used with a concept in one reading, and that concept plays other roles in the same reading, then all readings should have the same adjective for that occurrence of that concept; or at least the concept occurrences must not be ambiguous.

Fact type clauses may contain quantifier phrases, which assert mandatory or uniqueness constraints over the allowed population of instances of that fact. Qualifiers that assert ring constraints may also follow fact type clauses (see below). In some cases, a preceding qualifier “maybe” may also occur:



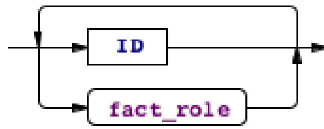
The post qualifiers are encased in square brackets, and are most commonly used for ring constraints. See the section on constraints for more details.

DRAFT

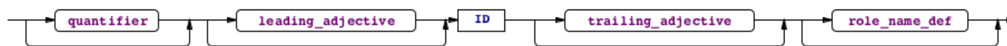
Fact Type Readings

Fact type readings are a sequence of roles and linking words. The definition of roles given here is slightly simplified for reasons that will be explained.

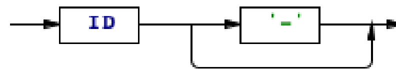
reading:



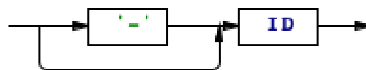
fact role:



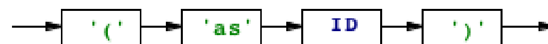
leading adjective:



trailing adjective:



role name def:



That's clearly ambiguous… especially since adjectives don't always have to have an associated dash character. So how is the ambiguity resolved?

First of all, CQL looks through all the fact roles of all readings in this definition, and finds where every role name is defined (using the last syntax shown). The preceding ID that's a defined concept name must be the real name of the role player, but elsewhere in this definition, it will be known by the role name. In the process, any adjectives that are marked with a dash are remembered with the associated role player. Adjectives may not be the same as the name of any defined concept, or of any local role name.

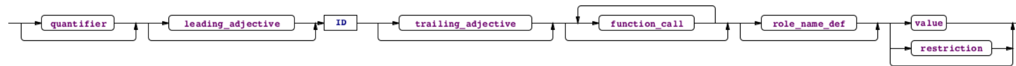
Then, all role players can be identified (the ID in the fact role syntax diagram), because all will either be defined concepts or local role names. Having identified all role players, unmarked adjectives are picked up in one of two ways; either they're used with this concept elsewhere in the same definition, or they occur in a reading of a previously defined fact type that uses those adjectives.

All the discussion so far has only referred to fact type readings being used to make new definitions, but the last paragraph reveals another usage. A fact type reading may be re-iterated in order to link that existing fact type into a new definition. This allows the introduction of the full definition of fact roles.

DRAFT

Fact Roles Revisited

The previous definition of fact roles omitted discussion of some of the optional syntactic elements. We discuss the quantifier here.



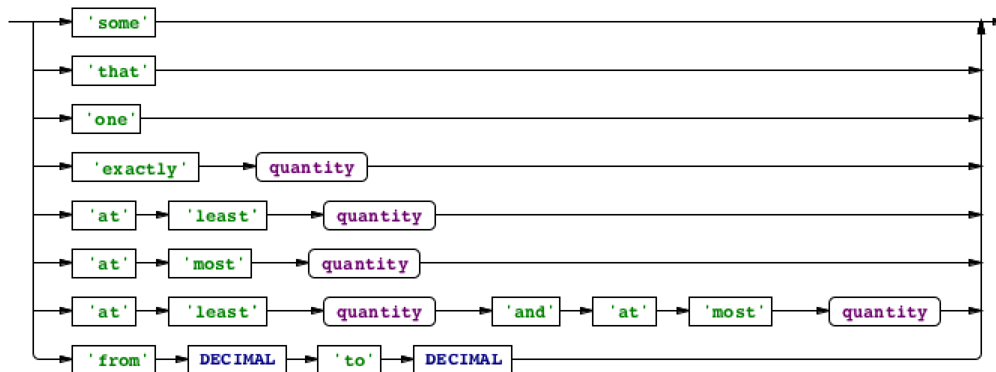
The quantifier is used to introduce a constraint on the number of times the **other** role players in this fact type may occur in conjunction with an instance of this concept. As a simple example, consider

Person has exactly one family-Name;

The fact type reading is “Person has family-Name”, and the quantifier “exactly one” requires that each Person occurs exactly once in the fact population; in other words, a family name must be recorded for each person, and only one family name may be recorded for each person.

Quantifiers

The full list of possible quantifiers is shown below (though some are only used in constraints):



quantity:

In this way, CQL absorbs the uniqueness, mandatory and frequency constraints of Object Role Modeling. The only ORM characteristic that cannot be expressed this way is a non-mandatory constraint having a minimum frequency above one; that is, a constraint that allows zero, or more than two, occurrences. For example, in a footy tipping competition, it might be the case that if a participant submits no tips this week, they get the tips published by a known tipster, but if they do submit tips, they must submit at least eight. This

DRAFT

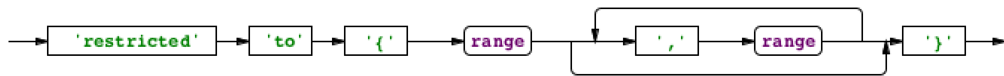
kind of non-mandatory frequency constraint may be expressed in CQL using the **maybe** qualifier:

maybe Participant entered at least 8 Tips

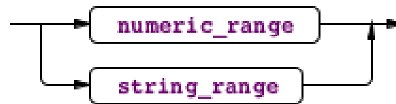
Value Restrictions

In addition to the value restrictions that can apply to data types, a role value may be restricted. The value restriction mentioned previously in data type definitions may also be used in a role value. This limits the allowed values that may populate that role:

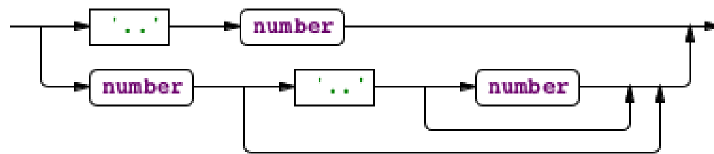
restriction:



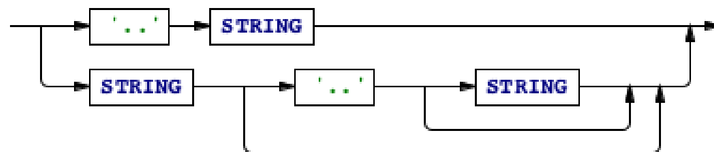
range:



numeric range:



string range:



Note that the ranges in a value restriction may be open ended at one end.

Fact Instances

When a fact reading is re-iterated with values, a fact instance is created. The simplest is where a declaration is just a concept name followed by a value:

Name 'Fred';

This form is allowed for any data type, or any entity type that's identified by a single data type (or an entity identified by a single entity identified by a single

DRAFT

data type, etc). In more complex cases, it might be necessary to invoke more than one fact type to define the instance:

Person is called given name ‘Fred’, Person has family Name ‘Bloggs’;

The Person instance being defined is a reference to the same instance in each fact type reading.

Fact Derivation

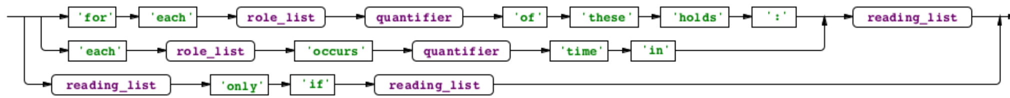
When a fact type has the optional condition clauses, the population of that fact type is derived from a **query** over the fact types it invokes. Each condition clause either joins in a new fact type with role players that relate to ones previously mentioned, or it applies a comparison operator over the values of roles declared in these clauses. The values may be modified by functions, such as the built-in functions on the Date data type which can extract the year, month, day, etc from the Date.

It’s possible to negate any invoked fact type by inserting the word **no** into it as a quantifier.

Fact derivation is a large subject by itself, so we’ll leave it for a future paper.

Constraints

Quantifiers allow the definition of the most common kinds of constraints, the mandatory, uniqueness and frequency constraints (collectively, CQL calls these **presence constraints**). Often there are constraints that cannot be expressed in this form however, such as when a concept must play one of many unrelated roles. This is handled in CQL by the use of an external constraint definition, or with a ring constraint qualifier.



Mandatory (and either-or) constraints

When a single role player must play one and only one (or at least one) of a set of roles, we can say:

each Range occurs at least one time in
Range has minimum-Bound,
Range has maximum-Bound;

for each ReceivedItem exactly one of these holds:
ReceivedItem is for PurchaseOrderItem,
ReceivedItem is for TransferRequest;

Where such constraints must be declared over fact types in which the same role player occurs in unrelated roles, the role players may be separated by the use of the keywords **some** and **that**. “That” means that this role player is the same one referred to earlier, and “some” means it’s not. If that’s still not enough, you can introduce adjectives and role names to separate the different instances of the same concept:

for each Unit exactly one of these holds:
Unit is fundamental,
that Unit is derived from some base-Unit;

External Uniqueness Constraints

For example, supposing that we were to identify Person instances by given name and family name (not a good idea in a real system!) we need to ensure that the **combination** given name, family name is unique. We can say:

each family Name, given Name occurs at most one time in
Person is known by given-Name,
Person has family-Name;

DRAFT

Subset Constraints

When one role may be played only if another is, you can use a subset constraint:

Address has third-StreetLine only if Address has second-StreetLine;

Note that this example didn't use the first and second StreetLine, as we assume that the first StreetLine is a mandatory part of the address; so the subset constraint would be redundant.

Ring Constraints

When a fact type includes the same concept more than once, or includes a supertype and its subtype, there's the possibility of the same instance playing both roles. This is often not desired, but further it introduces a whole class of further situations which can be restricted using ring constraints. The CQL keywords used in fact clause qualifiers for ring constraints are the following: **intransitive**, **transitive**, **acyclic** and **symmetric**. Intransitive means that just because "A relates to B", and "B relates to C", that doesn't mean that "A relates to C". Transitive means the opposite. Acyclic means that no A may relate to itself, or to any B that has that relation to A, and so on. Symmetric means that if A relates to B, B also relates to A (so there is only one fact instance possible between A and B).

Further constraints

More constraints are possible than are covered here, such as join constraints, and constraints over multiple roles. This is an advanced topic beyond the scope of this introductory paper.

DRAFT