Introduction

This module is for controlled vocabularies (CVs), semantic networks and ontologies, depending on which terminology you prefer.

It is intended to be rich enough to encapsulate anything in the Gene Ontology (GO) or OBO family of ontologies (see the GO website and the OBO project). The schema reflects the data model of OBO and of the OBO Edit tool currently used by these projects.

This module is also intended to be extensible to richer formalisms such as OWL (Ontology Web Language), but this is outside the current requirements.

Similarities to the GO Database schema

The schema is similar to the GO database schema, which was also developed by one of the Chado designers.

There is a *bridge* layer in the directory `modules/cv/bridges/`, which can make the Chado cv module look like the GO DB, and vice versa.

Overview

An ontology, or controlled vocabulary (cv) is a collection of classes (or concepts or terms, depending on your terminology) with definitions and relationships to other classes. Each class--a word or phrase--can only appear once in a controlled vocabulary and has a defined meaning within that vocabulary. The controlled vocabularies are chosen so that the contents do not overlap; if the same text string is used to describe two different concepts in two different cvs, these are distinct classes. These terms are housed in the cvterm table in the Chado schema.

cvterms are related to one another via cvterm_relationship. This can be thought of as a graph, or semantic network. The relationship types (the labels on the arcs of the graph) are also stored in the cvterm table. The relationship types are extensible, but the type *is a* (subtyping relationship) is assumed to be present; many OBO ontologies use the *part of* relationship, and GO also uses the *regulates* relation. Relationship types also come from a controlled vocabulary, the OBO Relation Ontology.

The cvterm_relationship can be thought of as specifying sentences about the cvterms. These sentences have 3 parts - a subject term, an object term, and a verb or type. For example in the phrase "an exon is part of a transcript" the subject of the sentence is "exon" and the object is "transcript". If you prefer to think of it as a directed graph), then you can think of the subject as the child node, and the object as the parent node.

Associating features to cvterms

This module is used by most of the Chado modules. But it is useful to describe here how this module would be used in the context of the sequence module.

Often we want to attach cvterms to features. One example is typing features with SO - this is central to the sequence module. Each feature has one primary type, stored in feature.type_id.

We can also attach an arbitrary number of non-primary cvterms to a feature.

For example, we may want to attach GO annotations to gene or protein features. We may also want to attach phenotypic terms to gene features (although the preferred way to do this is *via* a genotype using the genetics module).

Complex annotations

Note that this is something that is not handled by the current GO DB, but it is something we may want to allow for in future.

Currently in GO, all annotations are disjunctive; for example, if we have

```
gene | GO ID
-----+------
foo  | GO:001
foo  | GO:002
foo  | GO:003
```

This page or section needs to be edited. Please help by editing this page to add your revisions or additions.

The text above was taken from `modules/cv/cv-intro.txt` , which was incomplete (and no longer exists).

The sequence module makes extensive use of terms taken from various ontologies such as SO and the OBO Relations Ontology, using the *type_id* foreign key column. In addition, features can be annotated using ontologies such as GO using the feature_cvterm linking table. These terms are modelled using the cv module, the core of which is the cvterm table.

An ontology, terminology or cv (controlled vocabulary) is a collection of terms (here equivalent to what are more typically called classes, types, categories or kinds in the ontology literature in a particular domain of interest. Examples include "gene" (from SO), "transcription factor activity" (from GO molecular function) and "lymphocyte" (from OBO-Cell). The chado cv module is based on the GO Database schema. Terms are stored in the cvterm table, and relationships between terms are stored in the cvterm_relationship table. This table follows an analogous structure to the feature_relationship table, in that it has columns *subject_id*, *object_id* and *type_id*. Here, all three of these foreign keys refer to rows in the cvterm table.

A brief treatment of relationship types in biological ontologies can be found here. Of particular interest to Chado is the *is_a* relation, which specifies a sub-typing relationship between two terms or classes. Recall that tables in the sequence module frequently (such as the feature table) defined a *type_id* foreign key column to indicate the specific type or class of entity for each row in that table. The combination of the *type_id* column and the *is_a* relationship gives Chado a data sub-classing system, beyond what is possible with traditional SQL database semantics.

This is discussed further below. The collection of cvterms and cvterm_relationships can be considered to constitute vertices and edges in a graph. This graph is typically acyclic (a DAG), though it is not guaranteed to be as certain

relationship types are allowed to form cycles.

Sequence Ontology Examples

SO Term	SO id
exon	SO:0000147
intron	SO:0000188
mRNA	...
miRNA	...
regulatory_element	...
transcription_factor_binding_site	...

Transitive Closure

This section concerns relations between ontology terms and how defined terms and relations can be used to reason, either by humans or computers. A specialized ontology concerning these relations has been developed, the OBO Relation Ontology.

Often it is useful to know the transitive closure over a relationship type, or a collection of relationship types. The closure is the result of recursively applying the relationship. For example, if A *is_a* B, *B is_a C*, then the closure of *is_a* includes A *is_a* C.

In particular, we want the reflexive transitive closure. A term is always related to itself in a reflexive closure. Meaning:

```
X is_a X
```

This may seem odd, but it comes in useful both for doing queries and for deriving future rules. This makes it easier to ask "find me all genes of class X", and to get back genes attached to X and subtypes of X.

The closure goes in the cvtermpath table - the closure can also be thought of as a path through the graph or semantic network.

Transitivity of other Relations

Many other relations, such as *part_of* are also transitive.

If R is a transitive relation, then we can say

```
X R Z <= X R Y, Y R Z
```

For example, assume we have the following 3 *develops_from* links, and *develops_from* is a transitive relation:

```
neurectodermal cell develops_from glioblast
glioblast develops_from glial cell
```

Then it follows that glial cells develop from neurectodermal cells

Transitivity over *is_a*

It can be proved from the definition of *is_a* (proof not shown here) that:

```
X R Z <= X is_a Y, Y R Z
```

and

```
X R Z <= X R Y, Y is_a Z
```

This can be thought of as "inheritance".

For example, if an astrocyte *is_a* glial cell and a glial cell *develops_from* a glioblast, then it follows that an astrocyte *develop_from* a glioblast.

Difference between Deductive Closure and Transitive Closure

With a transitive closure we simply follow all links in the DAG, ignoring the relationship type. This works fine for ontologies such as GO that have only *is_a* and *part_of*, but is not ideal for other ontologies such as anatomical ontologies.

First of all, it may be possible for the closure to grow in size explosively.

Second of all, a closure that ignores the relations may be scientifically meaningless. It is also less useful for queries. For example, we may want to query for genes expressed in the larva or part of the larva, but not genes expressed in anatomical entities that develop from the larva.

Rules

The cvtermpath table is for calculating the reflexive transitive closure of a relationship, and any derived relationships.

Normal (direct) relationships are stored in the cvterm_relationship table. A entry in this table represents a *cvterm_relationship* S over some relation R.

```
S = Subj R Obj
```

For example:

```
S = "cardioblast" develops_from "mesodermal cell"
```

In addition to these *asserted* links, we want to be able to *deduce* links between terms.

If X *is_a* Y, then it follows that all of Y's cvterm_relationship statements are inherited by X.

Rule 1

```
If X is_a Y
and  Y R Z
then X R(inh) Z
```

For example:

```
 "cilium axoneme"  is_a "axoneme"
 "axoneme"part_of "cell projection"
```

Therefore:

```
 "cilium axoneme"  part_of(inh) "cell projection"
```

Here we use *T(inh)* to represent an inherited relationship.

Populating cvtermpath

The cvtermpath table stores the reflexive transitive closure of a relationship, taking into account subsumption or inheritance. The number of intermediate relationships is represented in the *distance* column of the table.

Here we use *T(path)* to represent the "path" or closure of a relationship. Every *T(path)* is stored in cvtermpath . We use the same *cvterm* for T, the fact that it is a path is implicit.

We use these rules:

Reflexive relationships:

```
for all relations T,
  X T(path) X
```

In this case the distance = 0.

Direct relationships:

These are also included in the cvtermpath table, distance = 1.

```
IfX T Y
Then X T(path) Y
```

Transitive relationships:

These have distance > 1; these also make use of inheritance rule, **Rule 1**, which gives us *T(inh)*.

```
If X T(inh)  Y
and  Y T(path) Z
Then X T(path) Z
```

Note that this rule is recursive.

These rules should be used for populating cvtermpath. Attempting to calculate a more general closure where all relations are treated equally or ignored will produce combinatorial explosions over certain ontologies (e.g. Flybase

anatomy ontology). What does this mean in practice?

For a typical database, which may only have relations *isa*, *part_of* and *develops_from*, we will end up with 3 sets of paths.

The *isa* closure, *isa (path)* will include paths over cvterm_relationships that look like this:

```
a is_a b is_a c is_a d is_a e
```

The *part_of* closure, *part_of(path)* will include paths over cvterm_relationships that look like this:

```
a is_a b part_of c part_of d is_a e part_of f
```

The *develops_from* closure, *develops_from(path)* will include paths over cvterm_relationships that look like this:

```
a develops_from b develops_from c is_a d is_a e develops_from f
```

It may be tempting to mix different non-*isa* relationships in the same path, but this should **never** be done - there will be an unacceptable combinatorial explosion in many cases. Besides, there is no use for such a cvtermpath; it is meaningless.

Note that for Amigo-like query behaviour, it is necessary only to query cvtermpath, ignoring cvtermpath.type_id (these are obtained by querying cvterm_relationship).

Advanced Usage

This section describes advanced usage of the cv module for use with OWL-DL advanced Obo format 1.2 [REF] features or elements from other ontology formalisms.

If you aren't sure what this means, you probably don't need to read this section yet.

Note that this section is liable to change; in particular the scheme below may be replaced with a simpler one. For details of the simpler scheme, along the lines of the transform used in the GO Database. See:

- http://geneontology.cvs.sourceforge.net/geneontology/go-dev/xml/xsl/oboxml_to_godb_prestore.xsl?view=mark

(search for intersection_of)

Background

See the document on Converting OBO to OWL.

Logical definitions

In a normal ontology DAG representation in Chado, the cvterm_relationship rows represent relationships between terms, or more formally, **necessary conditions**. A logical definition must have both **necessary and sufficient conditions**. A logical definition often consists of a **generic term** (also known as "genus") and one or more **discriminating characteristics** (also known as "differentiae"). The discriminating characteristics are typically relationships.

For example, the logical definition of "larval locomotory behaviour" would be a "locomotory behaviour" (genus) which "during" "tt larval stage" (where "during" could be drawn from an ontology of relations, and larval stage may come from an insect developmental stage ontology). These constitute both necessary and sufficient conditions: the conditions are necessary in that all instances of larval locomotory behavior are necessarily locomotory behaviors and are necessarily manifested at the larval stage. We could represent this using a normal DAG. However, because this is a definition it also constitutes sufficient conditions, in that any instance of locomotory behavior which manifests at the larval stage is by definition a larval locomotory behavior.

In an ontology formalism like OWL-DL or OBO-1.2, genus-differentiae are represented using set-intersections.

Here is the OBO 1.2 representation:

```
[Term]
id: GO:0008345
name: larval locomotory behavior
namespace: biological_process
is_a: GO:0007626 ! locomotory behavior
is_a: GO:0030537 ! larval behavior
intersection_of: GO:0007626  ! GENUS: locomotory behavior
intersection_of: during FBdv:00005336 ! DIFFERENTIUM: during larval stage
```

Here is the equivalent in OWL (note: RDF-XML syntax is very verbose!):

```
<owl:Class rdf:ID="GO_0008345">
    <rdfs:label xml:lang="en">larval locomotory behavior</rdfs:label>
    <rdfs:subClassOf rdf:resource="#GO_0007626"/>
    <rdfs:subClassOf rdf:resource="#GO_0030537"/>
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#GO_0007626"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#during"/>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="#FBdv_00005336"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

When converting to Chado we employ a more economical representation, in terms of the number of triples we use:

```
<!-- normal DAG relationships (necessary conditions) -->
  <cvterm_relationship>
    <type_id>is_a</type_id>
    <subject_id>GO:0008345</subject_id>
```

```
    <object_id>GO:0007626</object_id>
  </cvterm_relationship>
  <cvterm_relationship>
    <type_id>is_a</type_id>
    <subject_id>GO:0008345</subject_id>
    <object_id>GO:0030537</object_id>
  </cvterm_relationship>

  <!-- Genus/generic term -->
  <cvterm_relationship>
    <type_id>intersection_of</type_id>
    <subject_id>GO:0008345</subject_id>
    <object_id>GO:0007626</object_id> <!-- locomotory behavior -->
  </cvterm_relationship>

  <!-- Discriminating characteristics -->
  <cvterm_relationship>
    <type_id>intersection_of</type_id>
    <subject_id>GO:0008345</subject_id>
    <object_id>

      <!-- anonymous term representing  during(larval stage) -->
      <cvterm>
        <dbxref_id>
          <dbxref>
            <db_id>internal</db_id>
            <accession>restriction--OBOL:during--GO:0008345</accession>
          </dbxref>
        </dbxref_id>

        <!-- note: as this is an anon term, the name will never
             be shown to a user -->
        <name>restriction--OBOL:during--GO:0008345</name>
        <cv_id>anonymous_cv</cv_id>
        <cvtermprop>
          <type_id>is_anonymous</type_id>
          <value>1</value>
          <rank>0</rank>
        </cvtermprop>
        <cvterm_relationship>
          <type_id>OBOL:during</type_id>
          <object_id>FBdv:00005336</object_id>
        </cvterm_relationship>
      </cvterm>

    </object_id>
  </cvterm_relationship>
```

Note that in the above, we are creating **anonymous** terms. We give them fake names and fake dbxrefs. In the bbop-experimental SVN branch of chado, names and dbxrefs are nullable, so these can be omitted. With the current schema, you must provide fake dbxrefs and names that are unique, such as the above (if you are not familiar with how Chado XML maps to the Chado schema, see the explanation below).

If you wish to convert OBO-specified logical definitions to Chado XML you will need go-perl, v0.05 or higher (if you have a lower version, the *intersection_of* tags will simply be ignored).

```
go2chadoxml ont.obo > ont.chado
```

How Logical Definitions are Stored in Chado

This involves no schema changes to the cv module. Each *intersection_of* goes in as a DAG arc of type *internal:intersection_of*. The *object_id* in the arc is either a term (for the genus) or an anonymous term representing a restriction (the differentium). The restriction has a relationship of some type to another term.

For example, for "larval locomotory behavior" we would normally just have:

```
LLB is_a LocomotoryBehavior
LLB is_a LarvalBehavior
```

If we load a logical definition for this term (see /t/data/llm/obo in the go-perl package), like this:

```
[Term]
id: GO:0008345
name: larval locomotory behavior
namespace: biological_process
is_a: GO:0007626 ! locomotory behavior
is_a: GO:0030537 ! larval behavior
intersection_of: GO:0007626  ! locomotory behavior
intersection_of: during FBdv:00005336 ! larval stage
```

Then the *intersection_of*s get stored using the basic DAG tables as:

Definition stored in cvterm_relationship table

Subject	Relation	Object
LLB	*intersection_of*	Locomotory Behaviour
LLB	*intersection_of*	anon:xxx
anon:xxx	during	FBv:00005336

This uses 4 cvterm_relationships and the creation of a new **anonymous** term that is never shown directly to the user. The anonymous term represents the class of things that happen during the larval stage.

Logical Definition Views

Two views: *cvterm_genus* and *cvterm_differentium* views are in chado/modules/cv/views.

Example Use Case: Phenotypes

The idea here is that queries for composed term "syndactyly" should automatically return the same results as a boolean query for "fusion" + *inheres_in* = "finger" regardless of whether the annotation is to the composed term or is a composed annotation (provided we put the logical definition of syndactyly in the database).

Example Use Case: Feature Types

The Sequence Ontology has some logical definitions - you will need to load the file so-xp.obo.

Example use case: GO

See Obol.

Example use case: Drawing DAGs

Currently the DAGs of many OBO ontologies are highly tangled; see
http://www.fruitfly.org/~cjm/obol/doc/go-complexity.html

If all terms have logical definitions, then there is only one **true** (genus) or *isa* parent. This enables us to disentangle the DAGs and draw distinct hierarchies. For example, the GO term **cysteine biosynthesis** could be drawn as two distinct hierarchies - one process and one chemical.

Loading OWL into Chado

Not all OWL-DL features are supported. Only *intersection_of*s corresponding to genus-differentiae are loaded.

First you must convert OWL into OBO 1.2 format. There will soon be a way to do this in OboEdit. For now you can use blipkit.

```
blip io-convert my.owl -to obo -o my.obo
```

Once you have an OBO file you can run go2chadoxml, as above.

Post-coordinating Terms

Sometimes we want to be able to refer to a term such as "plasma membrane of spermatocyte", but no such term exists in the ontology. Introducing these as **pre-coordinated** cross-product terms would make the ontology unwieldy due the large number of possible combinations.

Chado allows the **post-coordination** or **post-composition** of terms using the same formalism as described above. Briefly: we would create an **anonymous** term. This anonymous term would be defined using the terms "plasma membrane" and "spermatocyte", using a genus-differentia definition as above.

```
<!-- Genus/generic term -->
  <cvterm_relationship>
    <type_id>intersection_of</type_id>
    <subject_id>anon_1</subject_id>
    <object_id>GO__plasma_membrane</object_id>
  </cvterm_relationship>

  <!-- Discriminating characteristics -->
  <cvterm_relationship>
    <type_id>intersection_of</type_id>
    <subject_id>anon_1</subject_id>
    <object_id>
```

```
    <!-- anonymous term representing  part_of(spermatocyte) -->
    <cvterm>
      <dbxref_id>
        <dbxref>
          <db_id>internal</db_id>
          <accession>restriction--part_of--spermatocyte</accession>
        </dbxref>
      </dbxref_id>

      <!-- note: as this is an anon term, the name will never
           be shown to a user -->
      <name>restriction--part_of--spermatocyte</name>
      <cv_id>anonymous_cv</cv_id>
      <cvtermprop>
        <type_id>is_anonymous</type_id>
        <value>1</value>
        <rank>0</rank>
      </cvtermprop>
      <cvterm_relationship>
        <type_id>OBO_REL:part_of</type_id>
        <object_id>CL__spermatocyte</object_id>
      </cvterm_relationship>
    </cvterm>

  </object_id>
</cvterm_relationship>
```

The above assumes XORT macro IDs defined for *GO__plasma_membrane* and *CL__spermatocyte*.

Allow post-coordinated terms places a greater burden on applications that use the cv module. More documentation will be provided here on this.

This page or section needs to be edited. Please help by editing this page to add your revisions or additions.

Tables

Table: cv

A controlled vocabulary or ontology. A cv is composed of cvterms (AKA terms, classes, types, universals - relations and properties are also stored in cvterm) and the relationships between them.

cv Structure

F-Key	Name	Type	Description
	cv_id	serial	*PRIMARY KEY*
	name	character varying(255)	*UNIQUE NOT NULL* The name of the ontology. This corresponds to the obo-format -namespace-. cv names uniquely identify the cv. In OBO file format, the cv.name is known as the namespace.

definition	text	A text description of the criteria for membership of this ontology.

Tables referencing this one via Foreign Key Constraints:

- cvterm
- cvtermpath

Table: cvterm

A term, class, universal or type within an ontology or controlled vocabulary. This table is also used for relations and properties. cvterms constitute nodes in the graph defined by the collection of cvterms and cvterm_relationships.

cvterm Structure

F-Key	Name	Type	Description
	cvterm_id	serial	*PRIMARY KEY*
cv	cv_id	integer	*UNIQUE#1 NOT NULL* The cv or ontology or namespace to which this cvterm belongs.
	name	character varying(1024)	*UNIQUE#1 NOT NULL* A concise human-readable name or label for the cvterm. Uniquely identifies a cvterm within a cv.
	definition	text	A human-readable text definition.
dbxref	dbxref_id	integer	*UNIQUE NOT NULL* Primary identifier dbxref - The unique global OBO identifier for this cvterm. Note that a cvterm may have multiple secondary dbxrefs - see also table: cvterm_dbxref.
	is_obsolete	integer	*UNIQUE#1 NOT NULL* Boolean 0=false,1=true; see GO documentation for details of obsoletion. Note that two terms with different primary dbxrefs may exist if one is obsolete.
	is_relationshiptype	integer	*NOT NULL* Boolean 0=false,1=true relations or relationship types (also

		known as Typedefs in OBO format, or as properties or slots) form a cv/ontology in themselves. We use this flag to indicate whether this cvterm is an actual term/class/universal or a relation. Relations may be drawn from the OBO Relations ontology, but are not exclusively drawn from there.

Tables referencing this one via Foreign Key Constraints:

- acquisition_relationship
- acquisitionprop
- analysisprop
- arraydesign
- arraydesignprop
- assayprop
- biomaterial_relationship
- biomaterial_treatment
- biomaterialprop
- contact
- contact_relationship
- control
- cvterm_dbxref
- cvterm_relationship
- cvtermpath
- cvtermprop
- cvtermsynonym
- dbxrefprop
- element
- element_relationship
- elementresult_relationship

- environment_cvterm
- feature
- feature_cvterm
- feature_cvtermprop
- feature_genotype
- feature_pubprop
- feature_relationship
- feature_relationshipprop
- featuremap
- featureprop
- library
- library_cvterm
- libraryprop
- organism_relationship
- organismpath
- organismprop
- phendesc
- phenotype
- phenotype_comparison
- phenotype_cvterm
- phenstatement
- phylonode
- phylonode_relationship
- phylonodeprop
- phylotree

- protocol
- protocolparam
- pub
- pub_relationship
- pubprop
- quantification_relationship
- quantificationprop
- stock
- stock_cvterm
- stock_relationship
- stockcollection
- stockcollectionprop
- stockprop
- studydesignprop
- studyfactor
- synonym
- treatment
- wwwuser_cvterm

Table: cvterm_dbxref

In addition to the primary identifier (cvterm.dbxref_id) a cvterm can have zero or more secondary identifiers/dbxrefs, which may refer to records in external databases. The exact semantics of cvterm_dbxref are not fixed. For example: the dbxref could be a pubmed ID that is pertinent to the cvterm, or it could be an equivalent or similar term in another ontology. For example, GO cvterms are typically linked to InterPro IDs, even though the nature of the relationship between them is largely one of statistical association. The dbxref may be have data records attached in the same database instance, or it could be a "hanging" dbxref pointing to some external database. NOTE: If the desired objective is to link two cvterms together, and the nature of the relation is known and holds for all instances of the subject cvterm then consider instead using cvterm_relationship together with a

well-defined relation.

cvterm_dbxref Structure

F-Key	Name	Type	Description
	cvterm_dbxref_id	serial	*PRIMARY KEY*
cvterm	cvterm_id	integer	*UNIQUE#1 NOT NULL*
dbxref	dbxref_id	integer	*UNIQUE#1 NOT NULL*
	is_for_definition	integer	*NOT NULL* A cvterm.definition should be supported by one or more references. If this column is true, the dbxref is not for a term in an external database - it is a dbxref for provenance information for the definition.

Table: cvterm_relationship

A relationship linking two cvterms. Each cvterm_relationship constitutes an edge in the graph defined by the collection of cvterms and cvterm_relationships. The meaning of the cvterm_relationship depends on the definition of the cvterm R refered to by type_id. However, in general the definitions are such that the statement "all SUBJs REL some OBJ" is true. The cvterm_relationship statement is about the subject, not the object. For example "insect wing part_of thorax".

cvterm_relationship Structure

F-Key	Name	Type	Description
	cvterm_relationship_id	serial	*PRIMARY KEY*
cvterm	type_id	integer	*UNIQUE#1 NOT NULL* The nature of the relationship between subject and object. Note that relations are also housed in the cvterm table, typically from the OBO relationship ontology, although other relationship types are allowed.
cvterm	subject_id	integer	*UNIQUE#1 NOT NULL* The subject of the subj-predicate-obj sentence. The cvterm_relationship is about the subject. In a graph, this typically corresponds to the child node.
cvterm	object_id	integer	*UNIQUE#1 NOT NULL* The object of the subj-predicate-obj sentence. The cvterm_relationship refers to the object. In a graph, this typically corresponds to the parent node.

Table: cvtermpath

The reflexive transitive closure of the cvterm_relationship relation.

cvtermpath Structure

F-Key	Name	Type	Description
	cvtermpath_id	serial	*PRIMARY KEY*
cvterm	type_id	integer	*UNIQUE#1* The relationship type that this is a closure over. If null, then this is a closure over ALL relationship types. If non-null, then this references a relationship cvterm - note that the closure will apply to both this relationship AND the OBO_REL:is_a (subclass) relationship.
cvterm	subject_id	integer	*UNIQUE#1 NOT NULL*
cvterm	object_id	integer	*UNIQUE#1 NOT NULL*
cv	cv_id	integer	*NOT NULL* Closures will mostly be within one cv. If the closure of a relationship traverses a cv, then this refers to the cv of the object_id cvterm.
	pathdistance	integer	*UNIQUE#1* The number of steps required to get from the subject cvterm to the object cvterm, counting from zero (reflexive relationship).

Table: cvtermprop

Additional extensible properties can be attached to a cvterm using this table. Corresponds to -AnnotationProperty- in W3C OWL format.

cvtermprop Structure

F-Key	Name	Type	Description
	cvtermprop_id	serial	*PRIMARY KEY*
cvterm	cvterm_id	integer	*UNIQUE#1 NOT NULL*
cvterm	type_id	integer	*UNIQUE#1 NOT NULL* The name of the property or slot is a cvterm. The meaning of the property is

			defined in that cvterm.
	value	text	*UNIQUE#1 NOT NULL DEFAULT '::text* The value of the property, represented as text. Numeric values are converted to their text representation.
	rank	integer	*UNIQUE#1 NOT NULL* Property-Value ordering. Any cvterm can have multiple values for any particular property type - these are ordered in a list using rank, counting from zero. For properties that are single-valued rather than multi-valued, the default 0 value should be used.

Table: cvtermsynonym

A cvterm actually represents a distinct class or concept. A concept can be refered to by different phrases or names. In addition to the primary name (cvterm.name) there can be a number of alternative aliases or synonyms. For example, "T cell" as a synonym for "T lymphocyte".

cvtermsynonym Structure

F-Key	Name	Type	Description
	cvtermsynonym_id	serial	*PRIMARY KEY*
cvterm	cvterm_id	integer	*UNIQUE#1 NOT NULL*
	synonym	character varying(1024)	*UNIQUE#1 NOT NULL*
cvterm	type_id	integer	A synonym can be exact, narrower, or broader than.

Table: dbxrefprop

Metadata about a dbxref. Note that this is not defined in the dbxref module, as it depends on the cvterm table. This table has a structure analagous to cvtermprop.

dbxrefprop Structure

F-Key	Name	Type	Description
	dbxrefprop_id	serial	*PRIMARY KEY*
dbxref	dbxref_id	integer	*UNIQUE#1 NOT NULL*

cvterm	type_id	integer	*UNIQUE#1 NOT NULL*
	value	text	*NOT NULL DEFAULT '::text*
	rank	integer	*UNIQUE#1 NOT NULL*