Chado Manual

From GMOD

Contents

Introduction

The Chado Documentation's Vocabularly

It will be useful to the reader to clarify a number of terms used throughout the Chado documentation. Because these terms have multiple meanings it is useful at the outset to clarify how they are most often used.

Database

A Database Management System (DBMS), the software that manages a database. The PostgreSQL software is an example of a DBMS. The documentation rarely uses the term *database* in this sense.

> A collection of tables or other database content stored within a particular DBMS, all of which can be queried together or otherwise mutually manimpulated -- the topmost hierarchal element in a DBMS's collection of data. By definition, data stored within different databases cannot be related, by query or otherwise. This is the sense of the term *database* in a DBMS context, such as PostgreSQL, but the Chado document rarely uses *datbase* in this sense.
>
> A set of organized data that is readable by a computer. This is the sense intended most often within the Chado documentation. Usually, the word *database* means something very specific, a particular revision or version of bioinformatic information stored in a Chado database schema.

As an example the word *database* might be used within this document to refer to a specific version of the Flybase data set, a version stored together in a Chado PostgreSQL-database along with other versions of the Flybase drosopholid bioinformatic data.

Schema

> A logical collection of tables or other DBMS-database content -- the layer below the topmost in a DBMS's collection of data. An organizing concept somewhat similar to that of a folder or directory; data stored within different schema of the same DBMS-Database can be related and otherwise mutually manipulated. This is the sense of the term *schema* in a DBMS context, such as PostgreSQL, but the Chado documentation rarely uses the term *schema* in this sense.
>
> A database design -- a set of table and other definitions that describe how and what data is to be stored, related validated, and otherwise kept in and retrieved from a DBMS. This is the sense intended most often within the Chado documentation. Chado is a schema.

As an example the phrase *loading the schema* might be used in this document to refer to creating within a DBMS the tables and other database structural elements that make up Chado.

Modularity

The Chado schema has been designed with modularity and compartmentalization of function in mind. Groups of tables concerned with a single knowledge domain are called *modules*. There is a core module, *general*, concerned with data underlying all other classes, these tables store information about databases, databases identifiers, and general information about Chado tables. Equal in importance in Chado is *cv*, the module concerned with **c**ontrolled **v**ocabularies or ontologies.

All other sets of tables, or *modules*, link to these *general* and *cv* tables directly or indirectly but are limited in scope to specific biological domains. For example, the *sequence* module is concerned with protein and nucleotide sequence, the *pub* module is concerned with articles and publications, and so on. In addition to these limitations in scope we see an effective absence of redundancy. For example, there is a module called *companalysis*, short for ”computational analysis”. Its tables are responsible for describing algorithms and the output of algorithms. The *Mage* module (for microarrays) uses *companalysis* in order to refer to algorithms in addition. The uniqueness, and generality, of the modules implies that one can rely on pre-existing modules for function if one is interested in introducing new modules.

Chado should be considered a highly extensible database due to its modular design. The clear segregation of function into modules, or sets of tables, should allow the introduction of new modules covering new domains of knowledge. The fact that new modules have been introduced into Chado since its initial release (e.g. *phylogeny*, *mage*, *stock*) confirms that the authors' design concepts were correct.

Ontologies

One of the more profound, recent changes in the nature of biology has to do with the adoption of ontologies, or controlled vocabularies, as a way to describe and organize data. Our most popular ontologies have arisen from the need to describe the remarkable variety of living things, and are very detailed and broad. Simultaneously these ontologies have served to categorize and classify the contents of entire databases that had been previously been atomized, or only partially coherent. Chado has been built from the outset to integrate with these ontologies, and this feature makes it extremely expressive. The acceptance of ontologies as general standards, and Chado's use of these ontologies, make Chado an excellent platform for annotation of biological data.

Associated Software

Chado is considered to be one of the key components in the GMOD suite. It should not be considered **the** database of GMOD, there are other ways of storing data within GMOD (e.g. BioSQL, Bio::DB::GFF (http://search.cpan.org/perldoc?Bio::DB::GFF), Bio::DB::SeqFeature (http://search.cpan.org/perldoc?Bio::DB::SeqFeature)) but it is the database of choice when complexity and breadth are required. Because of this central position in GMOD it has been the focus of much software development, not just the design of the schema itself but in terms of components using it and *adaptor* software that connect Chado to some other GMOD component. Thus we are able to use Chado with different browsers such as GBrowse and Apollo (the latter being able to both read from and write to the database). We have a variety of tools that can be used to load sequence data as GFF into Chado and there are different utilities that can move complex data in and out of Chado as XML (XORT, GMODTools).

Complexity and Detail

Part of the impetus for the creation of Chado was the need for a database that could describe **all** the detail that arises from extensive research done on model organisms. There are a number of schemas available that are built upon on the classic, and simple, concept of the *central dogma*, *from 1 gene to 1 RNA to protein*. However it can become difficult to work with simpler schemas when one wants to describe *trans*-splicing or non-coding genes or other departures from the simple model that are commonly found in viral or bacterial genomes. The Chado schema is in use at model organism databases such as FlyBase (http://flybase.org) and XenBase (http://xenbase.org) and the close ties between the Chado developers and these model organism communities has resulted in a extensively tested data model, capable of handling both detail and departures from the *standard model*.

Data Integration

One characteristic of biology over the past decades has been the major impact of methods or technologies. With each new adopted approach biologists have created sizable data repositories, either as private collections or in the public domain. The value and meaning of these data types is fully realized when they are tied to other types of data. For example, an evolutionary biologist may be studying the evolution of non-coding, regulatory sequences. In order to do this she will consider integrating expression data from microarrays (using the *mage* module) with sequence data (stored using the *sequence* module) from various species (the *organism* module) with phylogenetic trees (*phylogeny* module) with results from sequence comparison studies (*companalysis* module). This is not to suggest that such an effort is easy, rather that with a schema like Chado such a proposition is actually *possible*.

Support

The community using Chado, and GMOD, is extensive and growing. Chado is in use at a number of sites worldwide (see GMOD Users) and is being considered by researchers in related fields, notably molecular evolution and ecology. The Chado community is active and supports a number of different mailing lists. It is also notable that there are close ties not just between GMOD and Chado users but also between these two groups and the community of ontology developers (for example, OBO (http://obofoundry.org), NCBO (http://www.bioontology.org/)).

There is also Chado Documentation in this Wiki:

- Chado - Getting Started
- Introduction to Chado
- FAQ for Chado
- Chado Tables
- Chado Best Practices

Modules

We organised the tables into distinct modular components with tightly defined dependencies. This is recognised as good software engineering practice, it allows different software components to focus on the specific data compartments required. It allows for extensibility and schema evolution within specific modules without disrupting the rest of the schema. Finally, it allows for a mix and match approach - it is the authors' hope that the schema modules will be adopted by other model organism and bioinformatics groups; these groups may want to swap in their own table variants within specific modules, or add modules of their own.

- Audit - for database audit trails
- Companalysis - for data from computational analysis
- Contact - for people, groups, and organizations
- Controlled Vocabulary (cv) - for controlled vocabularies and ontologies
- Expression - for summaries of RNA and protein expresssion
- General - for identifiers
- Genetic - for genetic data and genotypes
- Library - for descriptions of molecular libraries
- Mage - for microarray data
- Map - for maps without sequence
- Natural Diversity (ND) - for multiple experiments, such as phenotyping and genotyping
- Organism - for taxonomic data
- Phenotype - for phenotypic data
- Phylogeny - for organisms and phylogenetic trees
- Publication (pub) - for publications and references
- Sequence - for sequences and sequence features
- Stock - for specimens and biological collections
- WWW -

Module Dependencies

There is one module, general, that does not depend on or inherit from any other module. All other modules require general or some other module. Many modules require the sequence module or the cv module, or both.

Module	Depends on
general	*none*
organism	general cv
pub	general cv

cv	general
sequence	cv general pub organism
genetic	sequence cv general pub phenotype
expression	sequence cv pub
map	sequence cv pub
rad	sequence cv pub organism contact general companalysis
companalysis	sequence cv
contact	cv
library	sequence cv pub organism
phenotype	cv sequence
phylogeny	sequence cv pub organism general
stock	cv pub general organism genetic
www	sequence cv pub phenotype organism expression general genetic

Inter-module Linking Tables

These can be thought of as floating outside of the respective modules they bridge, although they are generally bundled with one or the other module.

Linking Table	Module	Module
biomaterial_dbxref	rad	general
cvterm_dbxref	cv	general
environment_cvterm	phenotype	cv
expression_cvterm	cv	expression
expression_pub	pub	expression
feature_cvterm	cv	sequence
feature_cvterm_dbxref	sequence	general
feature_cvterm_pub	sequence	pub
feature_dbxref	general	sequence
feature_expression	sequence	expression
feature_genotype	sequence	genetic
feature_organism	organism	sequence
feature_phenotype	sequence	phenotype
feature_pub	sequence	pub
feature_relationship_pub	sequence	pub
feature_relationshipprop_pub	sequence	pub
feature_synonym	general	sequence
featureloc_pub	sequence	pub
featuremap_pub	sequence	pub
featureprop_pub	pub	sequence
gene_synonym	general	sequence
journal_dbxref	general	pub
library_cvterm	library	cv
library_feature	library	sequence
library_pub	library	pub

organism_dbxref	general	organism
phenotype_cvterm	cv	genetic
phylonode_dbxref	phylogeny	general
phylonode_organism	phylogeny	organism
phylonode_pub	phylogeny	pub
phylotree_pub	phylogeny	pub
pub_dbxref	general	pub
stock_cvterm	stock	cv
stock_dbxref	stock	general
stock_genotype	stock	genetic
stock_pub	stock	pub
stock_relationship_pub	stock	pub
stockprop_pub	stock	pub
wwwuser_author	www	pub
wwwuser_cvterm	www	cv
wwwuser_expression	www	expression
wwwuser_feature	www	sequence
wwwuser_genotype	www	genetic
wwwuser_organism	www	organism
wwwuser_phenotype	www	phenotype
wwwuser_project	www	general
wwwuser_pub	www	pub

Chado Naming Conventions

Case sensitivity

We use lowercase in all tables and column names - DBMSs differ in how they treat case sensitivity. For example, Oracle will automatically capitalize everything. So it's best to be neutral and use lowercase.

Table names

In table names, we use underscores for linking tables; e.g. *feature_dbxref* is a linking table between *feature* and *dbxref*.

Where a table name is a noun phrase rather than a single noun, we concatenate the words together. For instance the table for describing feature properties is called *featureprop*. It could be argued this is harder to read, but it does allow consistent usage of underscores as above. FeatureProp could be used where it is known the DBMS is case insensitive.

Column names

In column names, we also use concatenated noun phrases, except in the case of primary or foreign keys, e.g. *dbxref_id*.

We try to keep column names unique where appropriate, which is useful for large join statements or views, in avoiding column name clash between different tables. The convention is to use an abbreviated form of the table name plus a noun describing the column, for instance *fmin* in the feature table. By consistently using abbreviated forms we stop column names getting too big (many DBMSs will complain about long column names).

Primary and foreign key names

We use the same column name for primary and foreign key columns - very useful for NATURAL JOIN statements.

Constraints

Constraint names are a concatentation of table name, underscore, the letter *c*, and a digit. For example: *feature_phenotype_c1*.

Indexes

Index names are a concatentation of table name, underscore, the string *idx*, and a digit. For example: *feature_phenotype_idx1*.

Views

The names of views are lowercase. Where a table name is a noun phrase rather than a single noun, we concatenate the words together using the *underscore*. For example the view used to query for nucleotide motifs is called *nucleotide_motif* and the view used to find exons from pseudogenes is called *pseudogenic_exon*.

Design Patterns

This page or section needs to be edited. Please help by editing this page (http://gmod.org/mediawiki/index.php?title=Chado_Manual&action=edit) to add your revisions or additions.

Module System

Module Metadata

View Layers

Views can be thought of as virtual tables. They provide a powerful abstraction layer over the database. All views should be portable across all DBMSs

Views in chado are defined on a per module basis. View definitions are maintained in the chado/modules/MODULE-NAME/views directory.

Included in the view directory are report views. These can usually be found in a file called chado/modules/MODULE-NAME/views/MODULE-NAME-report.sql

Collections of view definitions are bundled into packages, each package is a .sql file.

Inter-schema Bridges

GODB Bridge

BioSQL Bridge

DBMS Functions

This page or section needs to be edited. Please help by editing this page (http://gmod.org/mediawiki/index.php?title=Chado_Manual&action=edit) to add your revisions or additions.

DBMS Functions in Chado are entirely optional.

Functions in chado are defined on a per module basis. Function definitions are maintained in the chado/modules/MODULE-NAME/functions directory.

Collections of function definitions are bundled into packages. Each package comes with an **interface descriptions** and one or more **implementations**.

Function Interface Definitions

The interface descriptions are stored in a *.sqlapi file. The syntax used is a variant of SQL and is intended primarily as a consistent way of providing information for human, although it should be parseable by software.

Here is an example, taken from the top of the chado/modules/sequence/functions/subsequence.sqlapi package. This package provides basic subsequencing functions. It has dependencies on two other function packages, declared at the top of the file. The package declares multiple functions, only the first of which is show here, a function for extracting subsequences from the sequence of a feature.

```
IMPORT reverse_complement(TEXT) FROM 'sequtil';
IMPORT get_feature_relationship_type_id(TEXT) FROM 'sequence-cv-helper';

-----------------------------------
-- basic subsequencing functions --
-----------------------------------

DECLARE FUNCTION subsequence(
srcfeature_id  INT REFERENCES feature(feature_id),
fmin   INT,
fmax   INT,
strandINT
)
 RETURNS TEXT;

COMMENT ON FUNCTION subsequence(INT,INT,INT,INT) IS 'extracts a
subsequence from a feature referenced by srcfeature_id, within the
interbase boundaries determined by fmin and fmax, reverse
complementing if strand = -1. The sequence can be DNA or AA. Strand
must always by >0 for AA sequences';
```

Function Implementations

The goal is to provide implementations for different dialects of procedural SQL. Currently only PostgreSQL dialect is supported. The psql implementations are stored in *.plpgsql files.

Glossary

This document has a glossary of technical, non-biological, terms.

Retrieved from "http://gmod.org/mediawiki/index.php?title=Chado_Manual&oldid=25634"
Categories: Chado Modules | Needs Editing | Chado

- Last updated at 13:43 on 14 March 2014.
- 133,907 page views.
- Content is available under a GNU Free Documentation License unless otherwise noted.