machado Documentation

Release 0.1.0

Embrapa

Aug 18, 2022

Contents

1 Content **3**
1.1 Installation . . . 3
1.2 Data loading . . . 5
1.3 Visualization . . . 17
1.4 Diagrams . . . 23
1.5 Models . . . 30

machado provides users with a framework to store, search and visualize biological data.

It is a framework powered by Django that contains tools to interact with a Chado.

CHAPTER 1

Content

1.1 Installation

machado is a Django framework for Chado.

1.1.1 Prerequisite

The list bellow contains the softwares and versions required by *machado*.

PostgreSQL 12

Install PostgreSQL and create a database and user for loading the Chado schema. As postgres user run:

```
psql
create user username with encrypted password 'password';
create database yourdatabase with owner username;
alter user username createdb;
```

Don't forget to configure the PostgreSQL server to allow regular users to connect (pg_hba.conf).

Linux dependencies

Be sure to have the following dependencies installed

```
sudo apt install zlib1g-dev libbz2-dev liblzma-dev python3-dev
```

Python 3.8

We strongly recommend creating a new virtualenv for your project

```
virtualenv -p /usr/bin/python3.8 YOURPROJECT
cd YOURPROJECT
source bin/activate
```

machado

Just grab the code using GIT and install it:

```
git clone https://github.com/lmb-embrapa/machado.git src/machado
python src/machado/setup.py install
```

1.1.2 Preparation

From this point on it is assumed you have read the Django introduction and tutorial on the Django project website.

Create a Django project

Inside YOURPROJECT directory create a Django project with the following command:

```
django-admin startproject WEBPROJECT
cd WEBPROJECT
```

Then, configure the WEBPROJECT/settings.py file to connect to your Chado database.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',     # Set the DB driver
        'NAME': 'yourdatabase',                        # Set the DB name
        'USER': 'username',                            # Set the DB user
        'PASSWORD': 'password',                        # Set the DB password
        'HOST': 'localhost',                           # Set the DB host
        'PORT': '',                                    # Set the DB port
    },
}
```

Let Django know about your machado

In the WEBPROJECT/settings.py file, add chado to INSTALLED_APPS section.

```
INSTALLED_APPS = [
    ...
    'machado',
    'rest_framework',
    'drf_yasg',
    ...
]
```

(Additional information here: https://docs.djangoproject.com/en/2.1/intro/tutorial02/)

List the machado commands

```
python manage.py
```

1.1.3 Start you app and open the admin interface

You have to run the following command to create django admin tables:

```
python manage.py migrate
```

Run tests to check the instalation:

```
python manage.py test machado
```

Now, just run the DJango server to access the web interface:

```
python manage.py runserver
```

The API interface will be available at http://localhost:8000/machado/api

1.1.4 References

- http://gmod.org/wiki/Chado_Django_HOWTO

1.2 Data loading

The Chado schema heavily relies on **Ontologies** to integrate different datasets. Therefore it must be the first to load.

Inside the extras directory, there's a file named sample.tar.gz that might be helpful. It contains a few small FASTA and GFF files, together with a README file with examples of command lines.

1.2.1 Loading ontologies

The ontologies are required and must be loaded first.

Relations ontology

RO is an ontology of relations for use with biological ontologies.

- **URL**: https://github.com/oborel/obo-relations
- **File**: ro.obo

```
python manage.py load_relations_ontology --file ro.obo
```

Sequence ontology

Collect of SO Ontologies.

- **URL**: https://github.com/The-Sequence-Ontology/SO-Ontologies
- **File**: so.obo

```
python manage.py load_sequence_ontology --file so.obo
```

Gene ontology

Source ontology files for the Gene Ontology.

- **URL**: http://current.geneontology.org/ontology/
- **File**: go.obo

```
python manage.py load_gene_ontology --file go.obo
```

- Loading the gene ontology can be faster if you increase the number of threads (–cpu).
- After loading the gene ontology the following records will be created in the Cv table: gene_ontology, external, molecular_function, cellular_component, and biological_process.

Remove ontology

If, by any reason, you need to remove an ontology you should use the command *remove_ontology*. Most data files you'll load depend on the ontologies (eg. fasta, gff, blast). You should **never** delete an ontology after having data files loaded.

```
python manage.py remove_ontology --help
```

- This command requires the name of the ontology (Cv.name)
- There are dependencies between the Gene ontology records, therefore you need to delete the entry *external* first.

1.2.2 Loading taxonomy

Every data file you'll load (eg. fasta, gff, blast) must belong to an organism. These are instructions to load the NCBI taxonomy, that contains most organisms you'll need.

This step is optional. If you decide to skip this step, you should insert the organisms individually using the command *insert_organism*.

NCBI Taxonomy

Contains the names and phylogenetic lineages of more than 160,000 organisms that have molecular data in the NCBI databases.

- **URL**: https://www.ncbi.nlm.nih.gov/taxonomy
- **File**: ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

After unpacking the file you'll have the required files.

```
python manage.py load_organism --file names.dmp --name DB:NCBI_taxonomy
python manage.py load_phylotree --file nodes.dmp --name 'NCBI taxonomy tree' --
↪organismdb 'DB:NCBI_taxonomy'
```

- Loading these files can be faster if you increase the number of threads (–cpu).
- It will take a long time anyway (hours).

Remove taxonomy

If, by any reason, you need to remove a taxonomy you should use the command *remove_phylotree* and *remove_organisms*. Most data files you'll load depend on the organism database (eg. fasta, gff, blast). **If you delete an organism database, every data file you loaded will be deleted on cascade**.

```
python manage.py remove_phylotree --help
python manage.py remove_organisms --help
```

- These commands require the names of the databases (Db.name)

1.2.3 Inserting a new organism

Insert organism

Every data file you'll load (eg. fasta, gff, blast) must belong to an organism. These are instructions to insert a new organism to the database **if you did NOT** load the NCBI taxonomy or if the organism you're working with is not included.

```
python manage.py insert_organism --genus 'Organismus' --species 'ineditus'
```

- There are optional fields you can provide. Please take a look at the command help (–help).

Remove organism

If, by any reason, you need to remove an organism you should use the command *remove_organism*. Most data files you'll load depend on the organism record (eg. fasta, gff, blast). **If you delete an organism, every data file you loaded that depend on it will be deleted on cascade**.

```
python manage.py remove_organism --help
```

- These commands require the following info: Organism.genus and Organism.species

1.2.4 Loading publication files

Load publication

```
python manage.py load_publication --file reference.bib
```

- Loading this file can be faster if you increase the number of threads (–cpu).

Remove publication

If, by any reason, you need to remove a publication you should use the command *remove_publication*. **If you delete a publication, every record that depend on it will be deleted on cascade, with the exception of the Dbxref field that contains the DOI accession**.

```
python manage.py remove_publication --help
```

1.2.5 Loading FASTA files

This command is mostly used to load the reference genome. The reference sequences are exclusively used to feed JBrowse.

If the reference sequences are really long (>200Mbp), there may be memory issues during the loading process and JBrowse may take too long to render the tracks. To avoid that, it's possible to use the parameter (–nosequence) and configure JBrowse to get the reference data from a FASTA file.

Load FASTA

```
python manage.py load_fasta --file organism_chrs.fa --soterm chromosome --organism
↪'Arabidopsis thaliana'
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_fasta --help
```

–file	FASTA File *
–organism	Species name (eg. Homo sapiens, Mus musculus) *
–soterm	SO Sequence Ontology Term (eg. chromosome, assembly) *
–description	Description
–url	URL
–doi	DOI of a reference stored using *load_publication* (eg. 10.1111/s12122-012-1313-4)
–nosequence	Don’t load the sequences
–cpu	Number of threads

* required fields

Remove file

If, by any reason, you need to remove a fasta dataset you should use the command *remove_file*. **If you delete a file, every record that depend on it will be deleted on cascade**.

```
python manage.py remove_file --help
```

- This command requires the file name (Dbxrefprop.value)

1.2.6 Loading GFF files

The first column of a GFF file is the reference sequence ID. Usually, in order to load a GFF file, it’s required to have a reference FASTA file loaded. But some GFF files already have the reference features such as chromosome or scaffold. In this case, there are two options:

- Load the GFF directly, without a reference FASTA file
- Load the FASTA file and then load the GFF using the parameter ‘ignore’ to not load the reference features

The GFF file must be indexed using tabix.

Load GFF

```
python manage.py load_gff --file organism_genes_sorted.gff3.gz --organism
↪'Arabidopsis thaliana'
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_gff --help
```

–file	GFF3 genome file indexed with tabix (see http://www.htslib.org/doc/tabix.html) *
–organism	Species name (eg. Homo sapiens, Mus musculus) *
–ignore	List of feature types to ignore (eg. chromosome scaffold)
–doi	DOI of a reference stored using *load_publication* (eg. 10.1111/s12122-012-1313-4)
–qtl	Set this flag to handle GFF files from QTLDB
–cpu	Number of threads

* required fields

Remove file

If, by any reason, you need to remove a GFF dataset you should use the command *remove_file*. **If you delete a file, every record that depend on it will be deleted on cascade**.

```
python manage.py remove_file --help
```

- This command requires the file name (Dbxrefprop.value)

1.2.7 Loading Feature Additional Info

Please notice the features must be loaded in advance. If the annotation or sequence was loaded previously, it will be replaced.

Load Annotation

```
python manage.py load_feature_annotation --file feature_annotation.tab --soterm␣
↪polypeptide --cvterm display
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_feature_annotation --help
```

–file	Two-column tab separated file (feature.accession<TAB>annotation text). *
–soterm	SO Sequence Ontology Term (eg. mRNA, polypeptide) *
–cvterm	cvterm.name from cv feature_property. (eg. display, note, product, alias, ontology_term, annotation) *
–doi	DOI of a reference stored using *load_publication* (eg. 10.1111/s12122-012-1313-4)
–cpu	Number of threads

* required fields

Remove Annotation

If, by any reason, you need to remove a feature annotation you should use the command *remove_feature_annotation*.

```
python manage.py remove_feature_annotation --help
```

Load Sequence

```
python manage.py load_feature_sequence --file Athaliana_transcripts.fasta --soterm␣
↪mRNA
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_feature_sequence --help
```

–file	FASTA file. *
–soterm	SO Sequence Ontology Term (eg. chromosome, assembly, mRNA, polypeptide) *
–cpu	Number of threads

* required fields

Remove Sequence

If, by any reason, you need to remove a feature sequence you should use the command *load_feature_sequence* itself and provide a FASTA file with no sequence. For example:

```
>chr1

>chr2
```

Load Publication

```
python manage.py load_feature_publication --file feature_publication.tab
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_feature_publication --help
```

–file	Two-column tab separated file (feature.accession<TAB>DOI). *
–cpu	Number of threads

* required fields

Remove Publication

If, by any reason, you need to remove a feature publication attribution, you should use the command *remove_publication*.

```
python manage.py remove_publication --help
```

Load DBxRef

```
python manage.py load_feature_dbxrefs --file feature_dbxrefs.tab --soterm mRNA
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_feature_dbxrefs --help
```

–file	Two-column tab separated file (feature.accession<TAB>db:dbxref). *
–soterm	SOTERM SO Sequence Ontology Term (eg. chromosome, assembly, mRNA, polypeptide) *
–cpu	Number of threads

* required fields

1.2.8 Loading Blast results

In order to load a BLAST xml result file, both the query and the subject records must be previously stored (see below). The current version was tested for loading BLAST analysis on **proteins**.

Load BLAST subject records

In case you did a BLAST against a multispecies protein database, like NCBI's nr or Uniprot's trembl or swissprot, you need to load previously all subject matches before loading the result itself. To do so, use the following command:

```
python manage.py load_similarity_matches --file blast_result.xml --format blast-xml
```

- Loading this file can be faster if you increase the number of threads (–cpu).

Load BLAST

If all queries and subjects are already loaded you can run the following command to load a BLAST xml result:

```
python manage.py load_similarity --file blast_result.xml --format blast-xml --so_
↪query polypeptide --so_subject protein_match --program diamond --programversion 0.9.
↪24 --organism_query 'Oryza sativa' --organism_subject 'multispecies multispecies'
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_similarity --help
```

–file	Blast XML file *
–format	blast-xml *
–so_query	Query Sequence Ontology term. (eg. assembly, mRNA, polypeptide) *
–so_subject	Subject Sequence Ontology term. (eg. protein_match if loading a BLAST result) *
–organism_query	Query's organism name. eg. 'Oryza sativa'. Cannot be 'multispecies' *
–organism_subject	Subject's organism name eg. 'Oryza sativa'. Put 'multispecies multispecies' if using a multispecies database. *
–program	Program *
–programversion	Program version *
–name	Name
–description	Description
–algorithm	Algorithm
–cpu	Number of threads

* required fields

Remove file

If, by any reason, you need to remove a Blast result set you should use the command *remove_analysis*.

```
python manage.py remove_analysis --help
```

- This command requires the analysis name (Analysis.sourcename)

1.2.9 Loading InterproScan results

In order to load an InterproScan results file, both the query and the subject records must be previously stored. The current version was tested for loading InterproScan analysis on **proteins**.

Load InterproScan subject records

```
python manage.py load_similarity_matches --file interproscan_result.xml --format␣
↪interproscan-xml
```

- Loading this file can be faster if you increase the number of threads (–cpu).

Load InterproScan similarity

```
python manage.py load_similarity --file interproscan_result.xml --format interproscan-
↪xml --so_query polypeptide --so_subject protein_match --program interproscan --
↪programversion 5 --organism_query 'Oryza sativa' --organism_subject 'multispecies␣
↪multispecies'
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_similarity --help
```

–file	InterproScan XML file *
–format	interproscan-xml *
–so_query	Query Sequence Ontology term. (polypeptide) *
–so_subject	Subject Sequence Ontology term. (protein_match) *
–organism_query	Query's organism name. eg. 'Oryza sativa'. Cannot be 'multispecies'. *
–organism_subject	Subject's organism name eg. 'Oryza sativa'. Put 'multispecies multispecies' if using a multispecies database. *
–program	Program *
–programversion	Program version *
–name	Name
–description	Description
–algorithm	Algorithm
–cpu	Number of threads

* required fields

Remove file

If, by any reason, you need to remove an InterproScan result set you should use the command *remove_analysis*.

```
python manage.py remove_analysis --help
```

- This command requires the analysis name (Analysis.sourcename)

1.2.10 Loading OrthoMCL results

Load OrthoMCL

```
python manage.py load_orthomcl --file groups.txt
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_orthomcl --help
```

–file	Output result from OrthoMCL software.*
–cpu	Number of threads

* required fields.

Remove orthology

If, by any reason, you need to remove orthology relationships, you should use the command *remove_feature_annotation*.

```
python manage.py remove_feature_annotation --help
```

–cvterm TERM	'orthologous group'*
–organism ORGANISM	Species name. (eg. Homo sapiens, Mus musculus)

* required fields

1.2.11 Loading RNA-seq data

Load RNA-seq information

Before inserting RNA-seq count tables, it is needed to input information about the experiments and samples from which the data was generated. In Machado we will focus on the GEO/NCBI database as a source for RNA-seq experiments information and data. From the GEO/NCBI database we are supposed to get identifiers for different *series* (e.g.: GSE85653) that describe the studies/projects as a whole. From the GEO series we can get identifiers for biosamples or, in CHADO lingo, biomaterials (e.g.: GSM2280286). From the GEO biosamples we get identifiers for RNA-seq experiments (or assays), usually from the SRA database (e.g.: SRR4033018). SRA identifiers have links for the raw data one can be interested to analyse.

In Machado, it is necessary to input a .csv file with information for all SRA datafile regarding RNA-seq assays that will be input.

This file must have the following fields in each line:

"Organism specific name (e.g.: 'Oryza sativa')", "GEO series identifier (e.g: GSE85653)", "GEO sample identifier (e.g: GSM2280286)", "SRA identifier (e.g: SRR4033018)", "Assay description (e.g. Heat stress leaves rep1)", "Treatment description (e.g: 'Heat stress')", "Biomaterial description (e.g.: 'Leaf')", "Date (in format '%b-%d-%Y': e.g.: Oct-16-2016)".

A sample line for such a file can be seen below:

```
Oryza sativa,GSE85653,GSM2280286,SRR4033018,Heat leaf rep1,Heat stress,Leaf,May-30-
↪2018
```

To load such a file an example command can be seen below. The databases for the project, biomaterial and assay are required.:

```
python manage.py load_rnaseq_info --file file.csv --biomaterialdb GEO --assaydb SRA
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_rnaseq_info --help
```

–file	.csv file *
–biomaterialdb	Biomaterial database info (e.g.: 'GEO')
–assaydb	Assay database info (e.g.: 'SRA')
–cpu	Number of threads

* Any text editor can be used to make such a file.

Remove RNA-seq information

If, by any reason, you need to remove RNA-seq information relationships, you should use the command *remove_file –name*. **Every relations from filename (e.g. 'file.csv') will be deleted on cascade**.

```
python manage.py remove_file --help
```

- This command requires the file name 'file.csv.txt' used before as input to load RNA-seq information.

Load RNA-seq data

To load expression count tables for RNA-seq data, a tabular file should be loaded, that can contain data from several RNA-seq experiments, or assays, per column. This file should have the following header:

"Gene identifier" "SRA Identifier 1" "SRA Identifier 2" … "SRA Identifier n"

Example of a header for such a sample file, that contains two assays/experiments:

```
gene    SRR5167848.htseq        SRR2302912.htseq
```

The body of the table is composed of the gene identifier followed by the counts for each gene, in each experiment.

Example of a line of sucha a sample file:

```
AT2G44195.1.TAIR10      0.0     0.6936967934559419
```

Note that the count fields can have floats or integers, depending on the normalization used (usually TPM, FPKM or raw counts).

The gene identifier is supposed to already be loaded as a feature, usually from the organism's genome annotation .gff file.

We used the output of the LSTrAP program as standard format for this file.

```
python manage.py load_rnaseq_data --file file.tab --organism 'Oryza sativa' --
↪programversion 1.3 --assaydb SRA
```

- As default the program name is 'LSTrAP' but can be changed with –program
- The data is by default taken as normalized (TPM, FPKM, etc.) but can be changed with –norm
- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_rnaseq_data --help
```

–file	tabular text file with gene counts per line.
–organism	Scientific name (e.g.: 'Oryza sativa')
–programversion	Version of the software (e.g.: '1.3') (string)
–name	Optional name (string)
–description	Optional description (string)
–algorithm	Optional algorithm description (string)
–assaydb	Optional assay database info (e.g.: 'SRA') (string)
–timeexecuted	**Optional Date software was run. Mandatory format: e.g.:** 'Oct-16-2016' (string)
–program	Optional Name of the software (default: 'LSTrAP') (string)
–norm	**Optional Normalized data: 1-yes (tpm, fpkm, etc.); 0-no (raw** counts); default is 1) (integer)

Remove RNA-seq data

If, by any reason, you need to remove RNA-seq data relationships, you should use the command *remove_file –name*. **Every relations from filename (e.g. 'file.tab') will be deleted on cascade**.

```
python manage.py remove_file --help
```

- This command requires the file name 'file.tab' used before as input to load RNA-seq information.

1.2.12 Loading Coexpression analyzes

Load Coexpression correlated pairs of features

Coexpression analyzes usually generate correlation statistics regarding gene sets in a pairwise manner. For example, LSTrAP can generate a table with the Pearson correlation coefficient for every pair of genes in a RNA-seq experiment.

To load such a table into Machado, the file must be headless and tab separated, with the two first columns containing the correlated pair of IDs for the genes/features and the third column must contain the correlation coefficient among them. In the case of the output of the LSTrAP software, the coefficient is subtracted by 0.7 for normalization sakes.

A one line sample from such a table is showed below:

```
AT2G44195.1.TAIR10      AT1G30080.1.TAIR10      0.18189286870895194
AT2G44195.1.TAIR10      AT5G24750.1.TAIR10      0.1715779378273995
```

Note: The feature pairs from columns 1 and 2 need to be loaded previously.

To load such a table, one can run the command below:

```
python manage.py load_coexpression_pairs --file pcc.mcl.txt
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_coexpression_pairs --help
```

–file	‘pcc.mcl.txt’ File *
–soterm	‘mRNA’ sequence ontology term *default*
–cpu	Number of threads

* example output file from LSTrAP software.

Remove coexpression pairs

If, by any reason, you need to remove coexpression pair analyzes, all you need to do is pass the filename used to load the analyzes to the remove_relationships command: remove_relationship –file <coexpression_file>’ **Every coexpression relations from coexpression_file (e.g. ‘pcc.mcl.txt’ from LSTrAP) will be deleted on cascade**.

```
python manage.py remove_relationship --help
```

–file	‘mcl.clusters.txt’ file *

* example output file from LSTrAP software.

Load Coexpression clusters

Other type of coexpression analyzes involve clustering features using its correlation values. LSTrAP does that using the ‘mcl’ software. To load data from such analyzes, the input file must be headless and fields tab separated, with each line representing one cluster, and each column representing one gene/feature from that cluster. The first column should represent the cluster name and must have the format: “<cluster_name>:”. Three-cluster sample from such a file is shown below, the first line represents a cluster with three individuals, the second line a cluster with two, and the third line an orphaned cluster with only one individual (obs: orphaned clusters are discarded).:

```
ath_1:    AT3G18715.1.TAIR10      AT3G08790.1.TAIR10      AT5G42230.1.TAIR10
ath_2:    AT1G27040.1.TAIR10      AT1G71692.1.TAIR10
ath_3:    AT5G24750.1.TAIR10
```

Note: The genes/features from each column must be loaded previously.

To load such a file, one can run the command below:

```
python manage.py load_coexpression_clusters --file mcl.clusters.txt --organism
↪'Arabidopsis thaliana'
```

- Loading this file can be faster if you increase the number of threads (–cpu).

```
python manage.py load_coexpression_clusters --help
```

–file	‘mcl.clusters.txt’ file *
–organism	Scientific name (e.g.: ‘Arabidopsis thaliana’)
–soterm	‘mRNA’ sequence ontology term *default*
–cpu	Number of threads

* example output file from LSTrAP software.

Remove coexpression clusters

If, by any reason, you need to remove coexpression cluster analyzes, you need to pass the controlled vocabulary term ‘coexpression group’ and the organism scientific name to the command remove_feature_group: *remove_feature_group –cvterm ‘coexpression group’ –organism ‘Arabidopsis thaliana’ **

*Every coexpression group relations from that organism will be deleted on cascade**.

```
python manage.py remove_feature_annotation --help
```

–cvterm TERM	mandatory: ‘coexpression group’
–organism ORGANISM	Scientific name (e.g.: ‘Oryza sativa’)

1.3 Visualization

The machado web interface requires the instalation and configuration of third party software.

1.3.1 Index and search

Haystack

The Haystack software enables the Django framework to run third party search engines such as Elasticsearch and Solr. Even though you can use any search engine supported by Haystack, machado was tested using Elasticsearch.

Elasticsearch

The latest Elasticsearch supported by Haystack is version 5.x.x Install Elasticsearch following the instructions. “Elasticsearch requires Java 8 or later. Use the official Oracle distribution or an open-source distribution such as OpenJDK.” So before continuing you should probably want to have java installed using the following command (ubuntu 20.04):

```
sudo apt install openjdk-8-jdk
```

Now, proceding with elasticsearch instalation, run the following commands:

```
cd YOURPROJECT
source bin/activate
cd src
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.17.3-amd64.
↪deb
sudo dpkg -i elasticsearch-7.17.3-amd64.deb
sudo systemctl daemon-reload
```

(continues on next page)

(continued from previous page)

```
sudo systemctl enable elasticsearch.service
sudo systemctl start elasticsearch.service
```

Django Haystack

Install django-haystack following the official instructions.

```
pip install 'elasticsearch>=7,<8'
```

In the WEBPROJECT/settings.py file, add haystack to INSTALLED_APPS section.

```
INSTALLED_APPS = [
    ...
    'haystack',
    ...
]
```

The settings.py file should contain the search engine configuration.

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.elasticsearch7_backend.Elasticsearch7SearchEngine
↪',
        'URL': 'http://127.0.0.1:9200/',
        'INDEX_NAME': 'haystack',
    },
}
```

- In the settings.py file, set the variable MACHADO_VALID_TYPES to restrict the types of features that will be indexed. Otherwise, every feature will be indexed.

```
MACHADO_VALID_TYPES = ['gene', 'mRNA', 'polypeptide']
```

Indexing the data

Go to the WEBPROJECT directory and use the manage.py. Please notice that it's necessary to run rebuild_index if additional data is loaded to the database or if changes are made to the settings files.

```
python manage.py rebuild_index
```

- Rebuilding the index can be faster if you increase the number of workers (-k).

The Elasticsearch server has a 10,000 results limit by default. In most cases it will not affect the results since they are paginated. The links to export .tsv or .fasta files might truncated the results because of this limit. You can increase it using the following command line:

```
curl -XPUT "http://localhost:9200/haystack/_settings" -d '{ "index" : { "max_result_
↪window" : 500000 } }' -H "Content-Type: application/json"
```

1.3.2 Web server

Django manage runserver

Check if your server is working, test the *machado* server:

```
python manage.py runserver
```

Now, open your browser and go to http://127.0.0.1:8000

Use CTRL+C to stop the webserver.

Django Apache WSGI

Before starting, make sure you have Apache installed in your system. In Ubuntu 20.04, do:

```
sudo apt install apache2
```

In order to have Apache2 hosting the Django applications, it's necessary to use WSGI. By doing this it won't be necessary to run the runserver command anymore, Apache will take care of this process. It will be necessary to install the following package:

```
sudo apt install libapache2-mod-wsgi-py3
```

Now symlink the directory of YOURPROJECT to '/var/www/' (tested in Ubuntu 20.04):

```
sudo ln -s /FULL/PATH/TO/YOURPROJECT /var/www/
```

- Make sure this directory and sub-directories have 755 permissions

Now configure Apache to use the WSGI module. Here is the configuration file (/etc/apache2/sites-available/YOURPROJECT.conf)

```
<Directory "/var/www/YOURPROJECT/WEBPROJECT/WEBPROJECT">
<Files "wsgi.py">
    Require all granted
</Files>
</Directory>

Alias /YOURPROJECT/static/ /var/www/YOURPROJECT/WEBPROJECT/static/

<Directory "/var/www/YOURPROJECT/WEBPROJECT/static">
    Require all granted
</Directory>

WSGIDaemonProcess WEBPROJECT
WSGIPythonHome /var/www/YOURPROJECT
WSGIPythonPath /var/www/YOURPROJECT/WEBPROJECT
WSGIScriptAlias /YOURPROJECT /var/www/YOURPROJECT/WEBPROJECT/WEBPROJECT/wsgi.py
```

- In this example the whole project is in /var/www/YOURPROJECT, but it's not required to be there.
- This directory and sub-directories must have 755 permissions

There must be a symlink of your config file in the sites-enabled directory

```
sudo ln -s /etc/apache2/sites-available/YOURPROJECT.conf /etc/apache2/sites-enabled/
↪YOURPROJECT.conf
```

- In the WEBPROJECT/settings.py file, set the following variables:

```
ALLOWED_HOSTS = ['*']
MACHADO_URL = 'http://localhost/YOURPROJECT'

MACHADO_EXAMPLE_TXT = "kinase"
MACHADO_EXAMPLE_AA_ACC = "AT1G01030.1"
MACHADO_EXAMPLE_AA = 1869098
MACHADO_EXAMPLE_NA = 1869093

MACHADO_VALID_TYPES = ['gene', 'mRNA', 'polypeptide']

STATIC_URL = '/YOURPROJECT/static/'
STATIC_ROOT = '/var/www/YOURPROJECT/WEBPROJECT/static'
```

Now, run collectstatic to gather the static files from all libraries to STATIC_ROOT.

```
python manage.py collectstatic
```

It's necessary to restart the Apache2 service everytime there are modifications on configuration files or source code updates.

```
sudo systemctl restart apache2.service
```

Now, open your browser and go to http://localhost/YOURPROJECT

1.3.3 JBrowse (optional)

The JBrowse software renders genomic tracks from Chado using the *machado* API.

JBrowse

Before installing Jbrowse you should probably have Bioperl installed in your system (tested in Ubuntu 20.04):

```
sudo apt install bioperl
```

Then install JBrowse following the official instructions:

In Ubuntu 20.04:

Install some prerequisites:

```
sudo apt install build-essential zlib1g-dev curl
```

Also *Node.js* is needed:

```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Finally, proceed with JBrowse installation

```
wget https://github.com/GMOD/jbrowse/releases/download/1.16.9-release/JBrowse-1.16.9.
↪zip
unzip JBrowse-1.16.9.zip
sudo mv JBrowse-1.16.9 /var/www/html/jbrowse
cd /var/www/html
sudo chown `whoami` jbrowse
cd jbrowse
./setup.sh # don't do sudo ./setup.sh
```

configuring JBrowse

The *machado* source contains the directory extras. Copy the file extras/trackList.json.sample to the JBrowse data directory, inside a subdirectory with the name of the organism (e.g. /var/www/html/jbrowse/data/Arabidopsis thaliana) and rename it to trackList.json This directory structure will enable machado to embed JBrowse in its pages.

```
mkdir -p "/var/www/html/jbrowse/data/Arabidopsis thaliana"
cp extras/trackList.json.sample "/var/www/html/jbrowse/data/Arabidopsis thaliana/
↪trackList.json"
```

Edit the file trackList.json to set the **organism** name you have loaded to the database.

In case you have WSGI apache module configured and running, change the *baseUrl* variable in trackList.json to refer to the proper address:

```
baseUrl":    "http://localhost/YOURPROJECT/api/jbrowse"
```

- Now repeat the steps above for as many other organisms as you may have loaded to the database.
- Remember to restart the Apache server after the modifications.

machado API

In case you don't have the WSGI module installed under Apache (and did change the *baseUrl* variable in track-List.json), start the *machado* API framework:

```
python manage.py runserver
```

Once the server is running, just go to your browser and open your JBrowse instance (e.g.: http://localhost/jbrowse/?data=data/Arabidopsisthaliana).

machado

The settings.py file should contain these variables in order to have the jbrowse visualization embedded to the feature page. Don't forget to restart apache to make the settings changes up to date.

```
MACHADO_JBROWSE_URL = 'http://localhost/jbrowse'

MACHADO_JBROWSE_TRACKS = 'ref_seq,gene,transcripts,CDS,SNV'

MACHADO_JBROWSE_OFFSET = 1200
```

MACHADO_JBROWSE_URL: the base URL to the JBrowse instalation. The URL must contain the protocol (i.e. http or https)

MACHADO_JBROWSE_TRACKS: the name of the tracks to be displayed ('ref_seq,gene,transcripts,CDS,SNV' if not set).

MACHADO_OFFSET: the number of bp upstream and downstream of the feature (1000 if not set).

Use reference from FASTA file (optional)

If the reference sequences are really long (>200Mbp), there may be memory issues during the loading process and JBrowse may take too long to render the tracks. To avoid this, follow instructions to create and use and indexed fasta file as source the reference sequences https://jbrowse.org/docs/tutorial.html.

Follow the steps below.

Put the genome's assembly fasta file into your jbrowse organism's 'data/seq' directory (for example: /var/www/html/jbrowse/data/'Glycine max'/data/seq/Gmax.fa), change to this directory, and run the command:

```
samtools faidx Gmax.fa
```

- A 'Gmax.fa.fai' indexed fasta file will be created.

Now, modify your default trackList.json file. You need to make two modifications, first replace the "refSeqs" entry line (probably the second line of the file) with the following line:

```
"refSeqs" : "data/seq/Gmax.fa.fai",
```

And then change a whole code chunk, as follows:

```
{
 "category" : "1. Reference sequence",
 "faiUrlTemplate" : "data/seq/Gmax.fa.fai",
 "key" : "Reference sequence",
 "label" : "ref_seq",
 "seqType" : "dna",
 "storeClass" : "JBrowse/Store/SeqFeature/IndexedFasta",
 "type" : "SequenceTrack",
 "urlTemplate" : "data/seq/Gmax.fa",
 "useAsRefSeqStore" : 1
}
```

- The code above should replace all code from the "1. Referece sequence" category track code chunk.
- make sure "urlTemplate" points to the fasta file path, not the .fai indexed one.

Now restart the apache daemon for changes to take effect.

```
systemctl restart apache2
```

1.3.4 Cache (optional)

The machado views are pre-configured to enable local-memory caching for 1 hour. In order to move the cache to another location, you'll to set Django's cache framework following the official instructions.

Example

Include the following in the settings.py to store the cache in the database:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'machado_cache_table',
    }
}
CACHE_TIMEOUT = 60 * 60 * 24  # 86400 seconds == 1 day
```

Create the cache table:

```
python manage.py createcachetable
```

Clearing the cache table

It is a good idea to clear the cache table whenever you made changes to your machado installation. For this intent, install the django-clear-cache tool *https://github.com/rdegges/django-clear-cache*

```
pip install django-clear-cache
```

Then modify your settings.py file to add clear_cache:

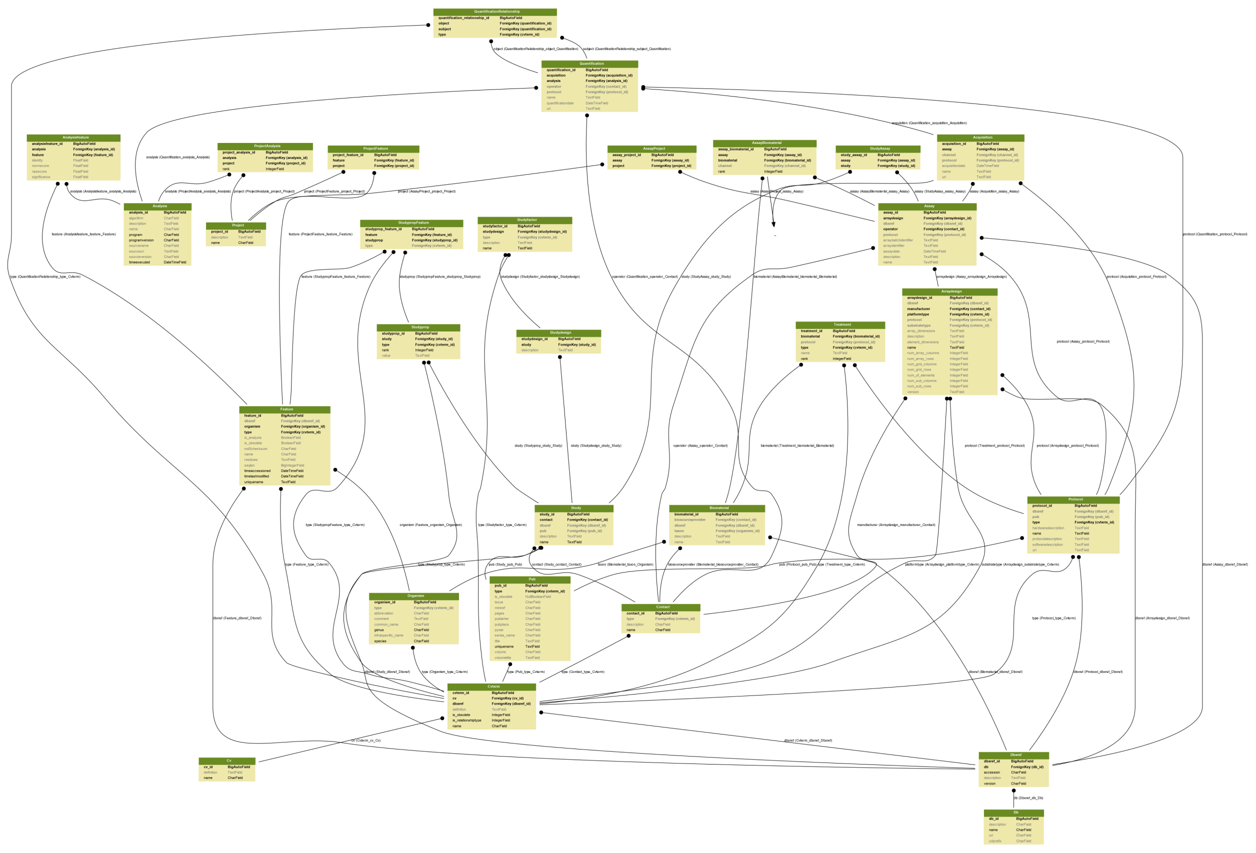
```
INSTALLED_APPS = (
# ...
'clear_cache',
)
```

Then to clear the cache just run

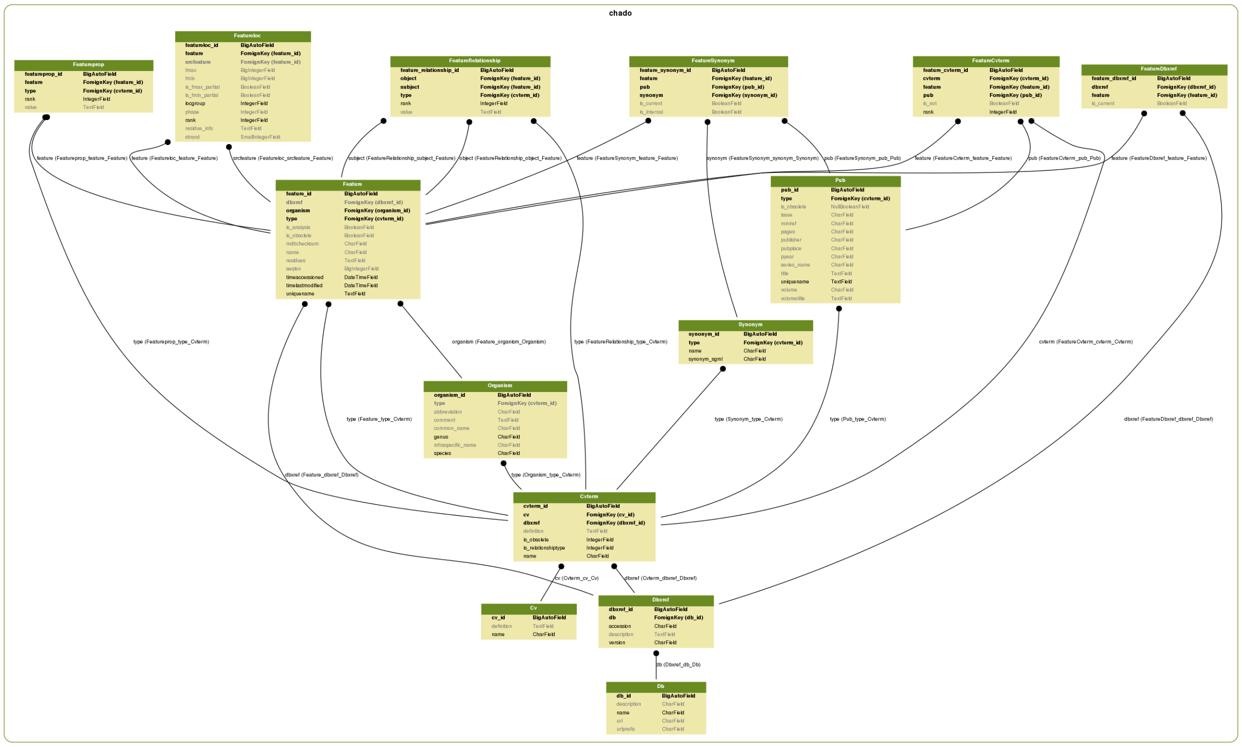
```
python manage.py clear_cache
```

1.4 Diagrams

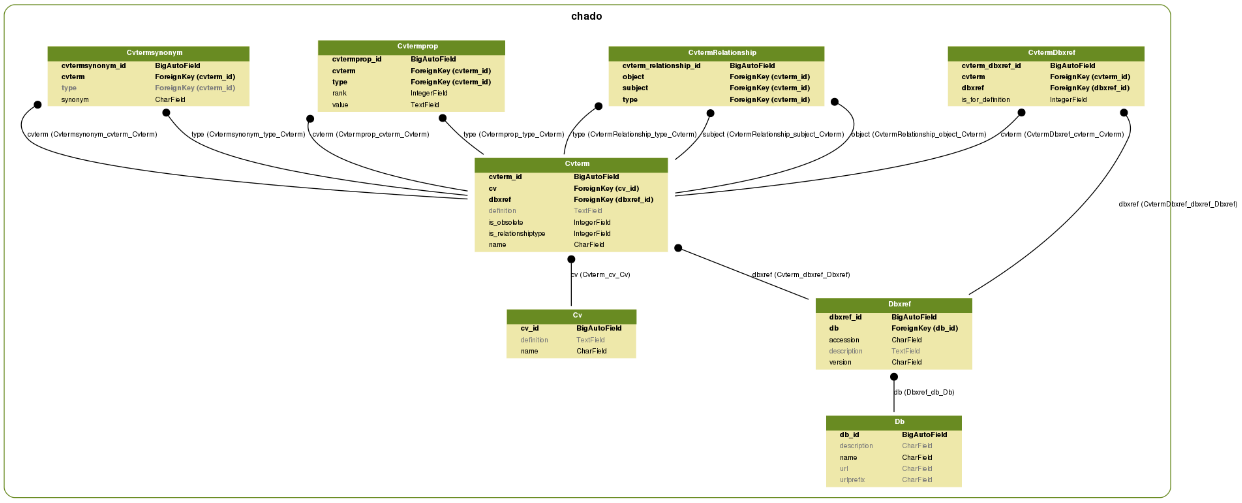
1.4.1 AnalysisFeature



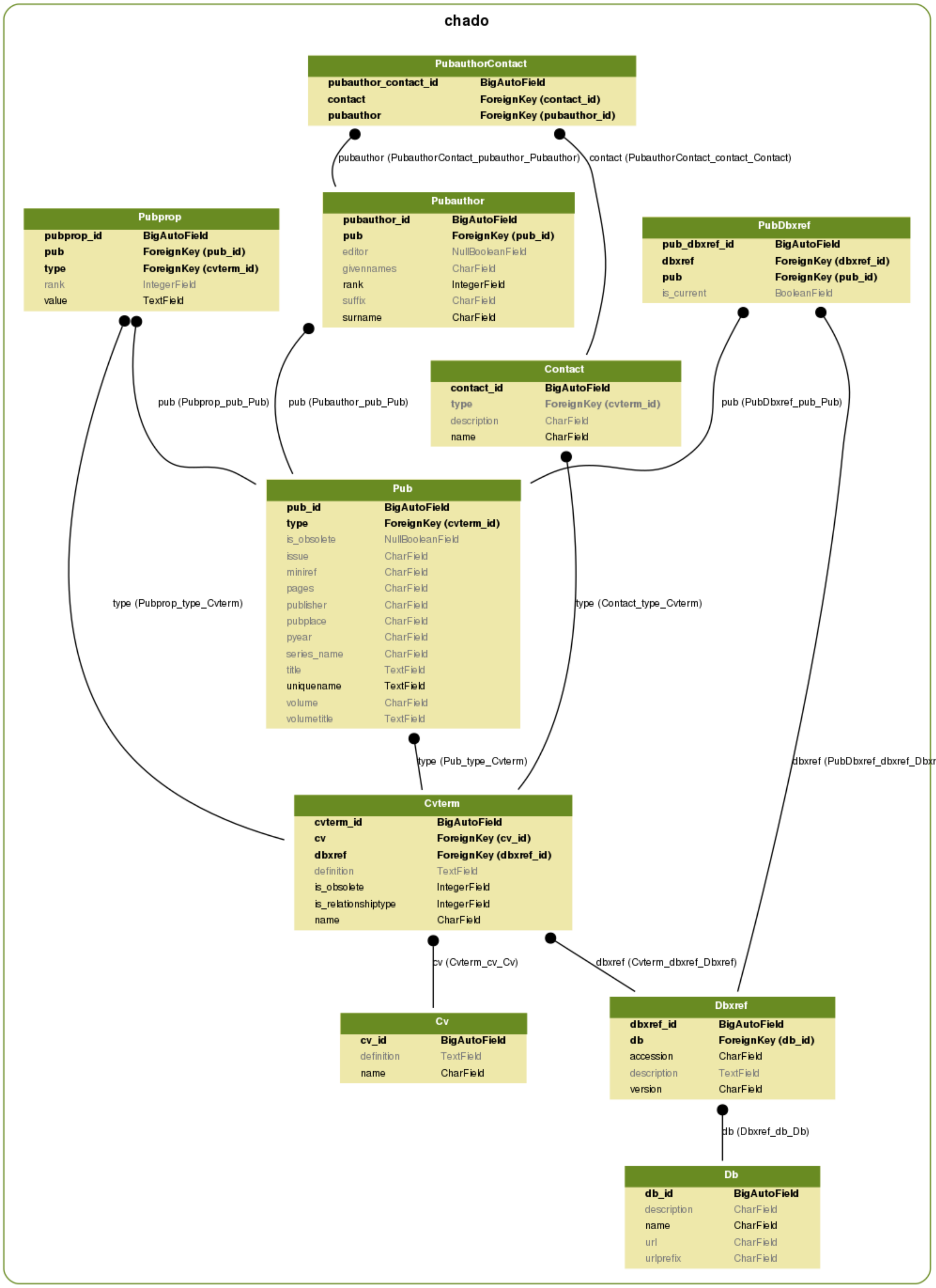
1.4.2 Feature



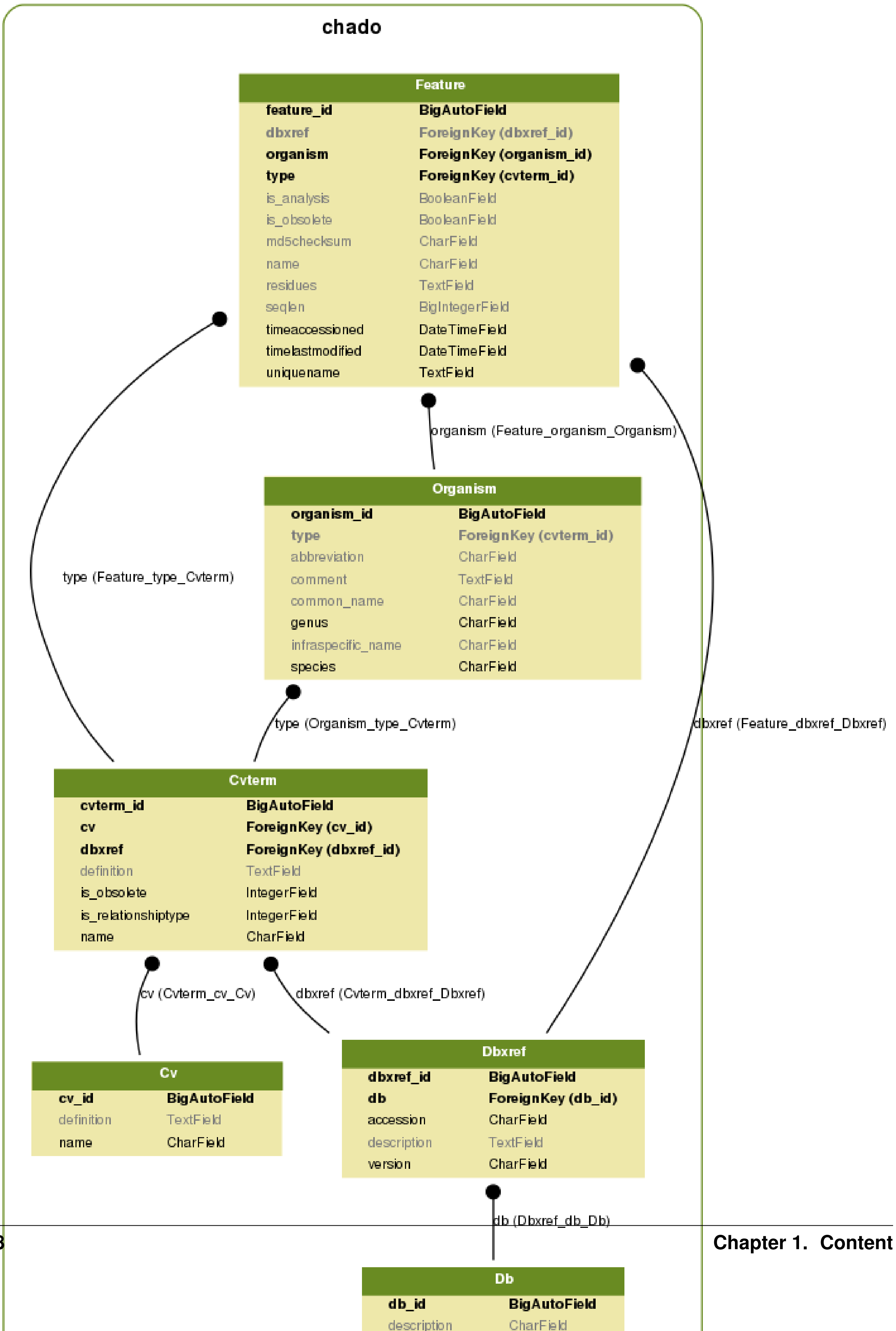
1.4.3 Ontology


chado
Cvtermsynonym
cvtermsynonym_id BigAutoField
cvterm ForeignKey (cvterm_id)
type ForeignKey (cvterm_id)
synonym CharField
Cvtermprop
cvtermprop_id BigAutoField
cvterm ForeignKey (cvterm_id)
type ForeignKey (cvterm_id)
rank IntegerField
value TextField
CvtermRelationship
cvterm_relationship_id BigAutoField
object ForeignKey (cvterm_id)
subject ForeignKey (cvterm_id)
type ForeignKey (cvterm_id)
CvtermDbxref
cvterm_dbxref_id BigAutoField
cvterm ForeignKey (cvterm_id)
dbxref ForeignKey (dbxref_id)
is_for_definition IntegerField
cvterm (Cvtermsynonym_cvterm_Cvterm)
type (Cvtermsynonym_type_Cvterm)
cvterm (Cvtermprop_cvterm_Cvterm)
type (Cvtermprop_type_Cvterm)
type (CvtermRelationship_type_Cvterm)
subject (CvtermRelationship_subject_Cvterm)
object (CvtermRelationship_object_Cvterm)
cvterm (CvtermDbxref_cvterm_Cvterm)
dbxref (CvtermDbxref_dbxref_Dbxref)
Cvterm
cvterm_id BigAutoField
cv ForeignKey (cv_id)
dbxref ForeignKey (dbxref_id)
definition TextField
is_obsolete IntegerField
is_relationshiptype IntegerField
name CharField
cv (Cvterm_cv_Cv)
dbxref (Cvterm_dbxref_Dbxref)
Cv
cv_id BigAutoField
definition TextField
name CharField
Dbxref
dbxref_id BigAutoField
db ForeignKey (db_id)
accession CharField
description TextField
version CharField
db (Dbxref_db_Db)
Db
db_id BigAutoField
description CharField
name CharField
url CharField
urlprefix CharField

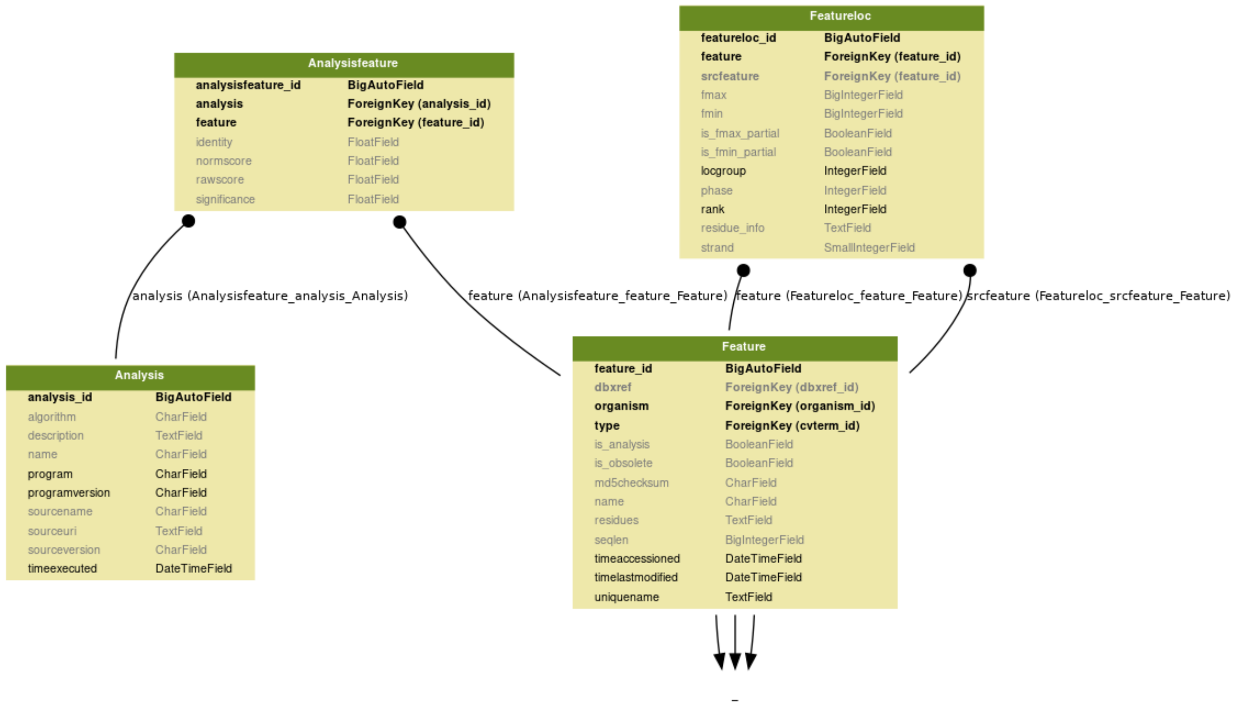
1.4.4 Publication



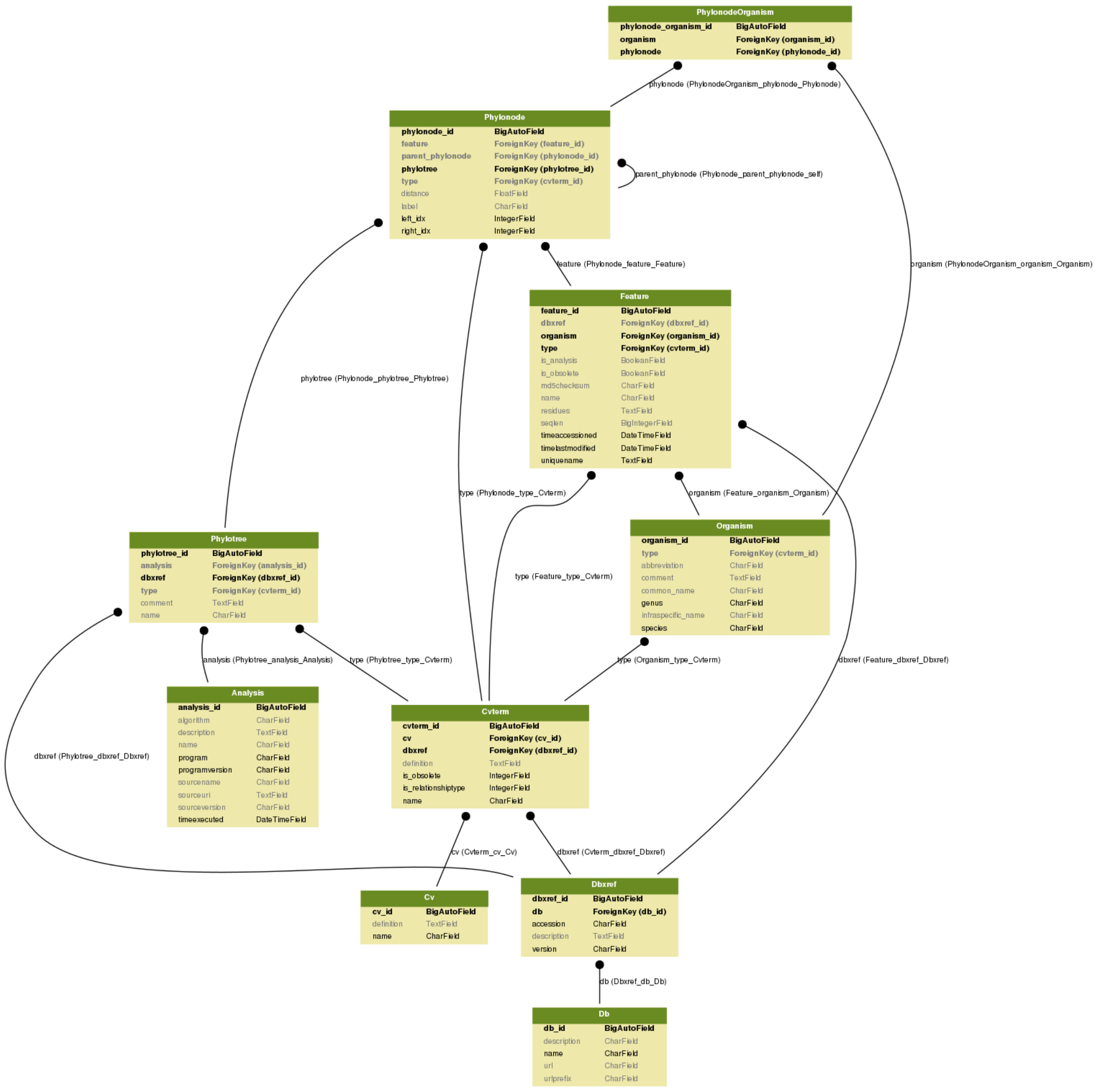
1.4.5 Sequence



1.4.6 Similarity



1.4.7 Taxonomy


PhylonodeOrganism
phylonode_organism_id BigAutoField
organism ForeignKey (organism_id)
phylonode ForeignKey (phylonode_id)
phylonode (PhylonodeOrganism_phylonode_Phylonode)
Phylonode
phylonode_id BigAutoField
feature ForeignKey (feature_id)
parent_phylonode ForeignKey (phylonode_id)
phylotree ForeignKey (phylotree_id)
type ForeignKey (cvterm_id)
distance FloatField
label CharField
left_idx IntegerField
right_idx IntegerField
parent_phylonode (Phylonode_parent_phylonode_self)
feature (Phylonode_feature_Feature)
organism (PhylonodeOrganism_organism_Organism)
phylotree (Phylonode_phylotree_Phylotree)
Feature
feature_id BigAutoField
dbxref ForeignKey (dbxref_id)
organism ForeignKey (organism_id)
type ForeignKey (cvterm_id)
is_analysis BooleanField
is_obsolete BooleanField
md5checksum CharField
name CharField
residues TextField
seqlen BigIntegerField
timeaccessioned DateTimeField
timelastmodified DateTimeField
uniquename TextField
type (Phylonode_type_Cvterm)
organism (Feature_organism_Organism)
Organism
organism_id BigAutoField
type ForeignKey (cvterm_id)
abbreviation CharField
comment TextField
common_name CharField
genus CharField
infraspecific_name CharField
species CharField
type (Feature_type_Cvterm)
Phylotree
phylotree_id BigAutoField
analysis ForeignKey (analysis_id)
dbxref ForeignKey (dbxref_id)
type ForeignKey (cvterm_id)
comment TextField
name CharField
analysis (Phylotree_analysis_Analysis)
type (Phylotree_type_Cvterm)
type (Organism_type_Cvterm)
dbxref (Feature_dbxref_Dbxref)
Analysis
analysis_id BigAutoField
algorithm CharField
description TextField
name CharField
program CharField
programversion CharField
sourcename CharField
sourceuri TextField
sourceversion CharField
timeexecuted DateTimeField
Cvterm
cvterm_id BigAutoField
cv ForeignKey (cv_id)
dbxref ForeignKey (dbxref_id)
definition TextField
is_obsolete IntegerField
is_relationshiptype IntegerField
name CharField
dbxref (Phylotree_dbxref_Dbxref)
cv (Cvterm_cv_Cv)
dbxref (Cvterm_dbxref_Dbxref)
Cv
cv_id BigAutoField
definition TextField
name CharField
Dbxref
dbxref_id BigAutoField
db ForeignKey (db_id)
accession CharField
description TextField
version CharField
db (Dbxref_db_Db)
Db
db_id BigAutoField
description CharField
name CharField
url CharField
urlprefix CharField

Note: Instructions to generate the diagrams: https://django-extensions.readthedocs.io/en/latest/graph_models.html (eg. python manage.py graph_models –pygraphviz -a -I Analysis,Analysisfeature,Feature -o analysis_feature.png)

1.5 Models

This document describes how the Django models.py file for the Chado schema was created. You **don't need** to create it again since *machado* already contains a copy of this file.

1.5.1 Prerequisite

The list bellow contains the softwares and versions required by *machado*.

PostgreSQL 9.5

Install PostgreSQL and create a database and user for loading the Chado schema. As postgres user run:

```
psql
create user username with encrypted password 'password';
create database yourdatabase with owner username;
```

Chado 1.31

Download Chado schema, unpack the file and load the chado-1.31/schemas/1.31/default_schema.sql to the database.

```
psql -h localhost -U username -W -d yourdatabase < chado-1.31/schemas/1.31/default_
↪schema.sql
```

Python 3.5.2

We strongly recommend creating a new virtualenv for your project

```
virtualenv -p /usr/bin/python3 YOURPROJECT
cd YOURPROJECT
source bin/activate
```

machado

Just grab the code using GIT and install it:

```
git clone https://github.com/lmb-embrapa/machado.git src/machado
python src/machado/setup.py install
```

1.5.2 The Django project

Inside YOURPROJECT directory create a Django project with the following command:

```
django-admin startproject WEBPROJECT
cd WEBPROJECT
```

Then, configure the WEBPROJECT/settings.py file to connect to your Chado database.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',     # Set the DB driver
        'NAME': 'yourdatabase',                                 # Set the DB name
        'USER': 'username',                                     # Set the DB user
        'PASSWORD': 'password',                                 # Set the DB password
        'HOST': 'localhost',                                    # Set the DB host
        'PORT': '',                                             # Set the DB port
    },
}
```

1.5.3 The model

Django has a command to generate a Models file:

```
python manage.py inspectdb > unsortedmodels.py
```

This will create a raw models.py with a model for each table and view in the specified Postgres database. This file needs to be fixed as each foreign key relation should have a unique name in Django to support reverse relationships. The following Python code will create these unique names. The code rewrites the models and also generate a admin.py file:

```
fixChadoModel.py --input unsortedmodels.py
```

The resulting files, models.py and admin.py, are ready.

1.5.4 References

- http://gmod.org/wiki/Chado_Django_HOWTO
- search