# grasp Documentation

***Release 0.4.0b1***

**Michael Dacre**

**Oct 26, 2016**

Contents

A Simple GRASP (grasp.nhlbi.nih.gov) API based on SQLAlchemy and Pandas.

| | |
|---|---|
| Author | Michael D Dacre <mike.dacre@gmail.com> |
| License | MIT License, made at Stanford, use as you wish. |
| Version | 0.4.0b1 |

For an introduction see the github readme

For a community discussion of the data itself see the wiki. This contains some important disclaimers regarding the quality of the data in GRASP itself, that fundamentally limit the utility of this package.

CHAPTER 1

# Basic Usage

This module contains a Python 3 API to work with the GRASP database. The database must be downloaded and initialized locally. The default is to use an sqlite backend, but postgresql or mysql may be used also; these two are slower to initialize (longer write times), but they may be faster on indexed reads.

The GRASP project is a SNP-level index of over 2000 GWAS datasets. It is very useful, but difficult to batch query as study descriptions are heterogeneous and there are more than 9 million rows. By putting this information into a relational database, it is easy to pull out bite-sized chunks of data to analyze with pandas. Be aware that GRASP does not include all of the SNPs that they should (see the wiki for info), so use this software with caution.

Commonly queried columns are indexed within the database for fast retrieval. A typical query for a single phenotype category returns several million SNPs in about 10 seconds, which can then be analyzed with pandas.

To read more about GRASP, visit the official page.

For a community discussion of the data itself see the wiki. This contains some important disclaimers regarding the quality of the data in GRASP itself, that fundamentally limit the utility of this package.

For complete API documentation, go to the documentation site

For a nice little example of usage, see the BMI Jupyter Notebook

**Contents**



## 1.1 Installation

The best way to install is with pip:

```
pip install https://github.com/MikeDacre/grasp/archive/v0.4.0b1.tar.gz
```

When this code reaches version 0.4.1, it will be placed on pypi.

Alternatively, you can clone the repo and install with setuptools:

```
git clone https://github.com/MikeDacre/grasp.git
cd grasp
python ./setup.py install --user
```

This code requires a grasp database. Currently sqlite/postgresql/mysql are supported. Mysql and postgresql can be remote (but must be set up with this tool), sqlite is local.

Database configuration is stored in a config file that lives by default in ~/.grasp. This path is set in *config.py* and can be changed there is needed.

A script, *grasp*, is provided in *bin* and should automatically be installed to your *PATH*. It contains functions to set up your database config and to initialize the grasp database easily, making the initial steps trivial.

To set up your database configuration, run:

```
grasp config --init
```

This will prompt you for your database config options and create a file at *~/.grasp* with those options saved.

You can now initialize the grasp database:

```
grasp init study_file grasp_file
```

The study file is available in this repository (grasp2_studies.txt.gz) It is just a copy of the official GRASP List of Studies converted to text and with an additional index that provides a numeric index for the non pubmed indexed studies.

Both files can be gzipped or bzipped.

The grasp file is the raw unzipped file from the project page: GRASP2fullDataset

The database takes about 90 minutes to build on a desktop machine and uses about 3GB of space. The majority of the build time is spent parsing dates, but because the dates are encoded in the SNP table, and the formatting varies, this step is required.

## 1.2 Usage

The code is based on SQLAlchemy, so you should read their ORM Query tutorial to know how to use this well.

It is important to note that the point of this software is to make bulk data access from the GRASP DB easy, SQLAlchemy makes this very easy indeed. However, to do complex comparisons, SQLAlchemy is very slow. As such, the best way to use this software is to use SQLAlchemy functions to bulk retrieve study lists, and then to directly get a pandas dataframe of SNPs from those lists.

Tables are defined in *grasp.tables* Database setup functions are in *grasp.db* Query tools for easy data manipulation are in *grasp.query*.

### 1.2.1 Tables

This module provides 6 tables:

Study, Phenotype, PhenoCats, Platform, Population, and SNP (as well as several association tables)

### 1.2.2 Querying

The functions in *grasp.query* are very helpful in automating common queries.

The simplest way to get a dataframe from SQLAlchemy is like this:

```
df = pandas.read_sql(session.query(SNP).statement)
```

Note that if you use this exact query, the dataframe will be too big to be useful. To get a much more useful dataframe:

```
studies = grasp.query.get_studies(pheno_cats='t2d', primary_pop='European')
df = grasp.query.get_snps(studies)
```

It is important to note that there are **three** ways of getting phenotype information: - The Phenotype table, which lists the primary phenotype for every study - The PhenoCats table, which lists the GRASP curated phenotype categories,

each Study has several of these.

- The phenotype_desc column in the SNP table, this is a poorly curated column directly from the full dataset, it roughly corresponds to the information in the Phenotype table, but the correspondance is not exact due to an abundance of typos and slightly differently typed information.

## 1.3 Example Workflow

```
from grasp import db
from grasp import tables as t
from grasp import query as q
s, e = db.get_session()

# Print a list of all phenotypes (also use with populations, but not with SNPs (too␣
↪many to display))
s.query(t.Phenotype).all()

# Filter the list
s.query(t.Phenotype).filter(t.Phenotype.phenotype.like('%diabetes%').all()

# Get a dictionary of studies to review
eur_t2d = get_studies(only_disc_pop='eur', primary_phenotype='Type II Diabetes␣
↪Mellitus', dictionary=True)

# Filter those by using eur.pop() to remove unwanted studies, and then get the SNPs␣
↪as a dataframe
eur_snps_df = get_snps(eur, pandas=True)

# Do the same thing for the african population
afr_t2d = get_studies(only_disc_pop='afr', primary_phenotype='Type II Diabetes␣
↪Mellitus', dictionary=True)
afr.pop('Use of diverse electronic medical record systems to identify genetic risk␣
↪for type 2 diabetes within a genome-wide association study.')
afr_snps_df = get_snps(afr, pandas=True)

# Collapse the matrices (take median of pvalue) and filter by resulting pvalue
eur_snps_df = q.collapse_dataframe(eur_snps_df, mechanism='median', pvalue_filter=5e-
↪8)
afr_snps_df = q.collapse_dataframe(afr_snps_df, mechanism='median', pvalue_filter=5e-
↪8)
```

```
# The new dataframes are indexed by 'chr:pos'

# Plot the overlapping SNPs
snps = q.intersect_overlapping_series(eur_snps_df.pval_median, afr_snps_df.pval_
↪median)
snps.plot()
```

## 1.4 ToDo

- Add more functions to grasp script, including lookup by position or range of positions

CHAPTER 2

# GRASP Console Script

A Simple GRASP (grasp.nhlbi.nih.gov) API based on SQLAlchemy and Pandas

| | |
|---|---|
| Author | Michael D Dacre <mike.dacre@gmail.com> |
| Organization | Stanford University |
| License | MIT License, use as you wish |
| Created | 2016-10-08 |
| Version | 0.4.0b1 |

Last modified: 2016-10-18 00:38

This is the front-end to a python grasp api, intended to allow easy database creation and simple querying. For most of the functions of this module, you will need to call the module directly.

```
usage: grasp [-h] {search,conf,info,init} ...
```

**Sub-commands:**

**search (s, lookup)** Query database for variants by location or id

Query for SNPs in the database. By default returns a tab-delimeted list of SNPs with the following columns: 'id', 'snpid', 'study_snpid', 'chrom', 'pos', 'phenotype', 'pval'

The –extra flag adds these columns: 'InGene', 'InMiRNA', 'inLincRNa', 'LSSNP'

The –study-info flag adds these columns: 'study_id (PMID)', 'title'

The –db-snp flag uses the myvariant API to pull additional data from db_snp.

```
usage: grasp search [-h] [--extra] [--study-info] [--db-snp] [--pandas] [-o]
                    [--path]
                    query
```

**Positional arguments:**

| | |
|---|---|
| **query** | rsID, chrom:loc or chrom:start-end |

**Options:**

| | |
|---|---|
| **--extra** | Add some extra columns to output |
| **--study-info** | Include study title and PMID |
| **--db-snp** | Add dbSNP info to output |
| **--pandas** | Write output as a pandas dataframe |
| **-o, --out** | File to write to, default STDOUT. |

**--path** PATH to write files to

**conf (config)** Manage local config

```
usage: grasp conf [-h]
                  [--db {sqlite,postgresql,mysql} | --get-path | --set-path␣
↪PATH | --init]
```

**Options:**

| | |
|---|---|
| **--db** | Set the current database platform.<br>Possible choices: sqlite, postgresql, mysql |
| **--get-path** | Change the sqlite file path |
| **--set-path** | Change the sqlite file path |
| **--init** | Initialize the config with default settings. Will ERASE your old config! |

**info** Display database info

Write data summaries (also found on the wiki) to a file or the console.

Choices: all: Will write everything to separate rst files, ignores all other flags except '–path' phenotypes: All primary phenotypes. phenotype_categories: All phenotype categories. populations: All primary populations. population_flags: All population flags. snp_columns: All SNP columns. study_columns: All Study columns.

```
usage: grasp info [-h] [-o] [--path]
                  {study_columns,phenotypes,population_flags,phenotype_
↪categories,all,snp_columns,populations}
```

**Positional arguments:**

| | |
|---|---|
| **display** | Choice of item to display, if all, results are written to independant rst files, and optional args are ignored<br>Possible choices: study_columns, phenotypes, population_flags, phenotype_categories, all, snp_columns, populations |

**Options:**

| | |
|---|---|
| **-o, --out** | File to write to, default STDOUT. |
| **--path** | PATH to write files to |

**init** Initialize the database

```
usage: grasp init [-h] [-n] study_file grasp_file
```

**Positional arguments:**

| | |
|---|---|
| **study_file** | GRASP study file from: github.com/MikeDacre/grasp/blob/master/grasp2_studies.txt |
| **grasp_file** | GRASP tab delimeted file |

**Options:**

| | |
|---|---|
| **-n, --no-progress** | Do not display a progress bar |

CHAPTER 3

# Library (API Documentation)

This code is intended to be primarily used as a library, and works best when used in an interactive python session (e.g. with jupyter) alongside pandas. Many of the query functions in this library returns pandas dataframes.

Below is a complete documentation of the API for this library. The functions in *grasp.query* will be the most interesting for most users wanting to do common db queries.

Tables are defined in *grasp.tables*, functions for connecting to and building the database are in *grasp.db*. *grasp.info* contains simple documentation for all of the tables and phenotypes (used to build this documentation).

*grasp.config* handles the static database configuration at *~/.grasp*, and *grasp.ref* is used to define module wide static objects, like dictionaries and the *PopFlags* class.

## 3.1 grasp.query

A mix of functions to make querying the database and analyzing the results faster.

**Primary query functions:**

***get_studies()*:** Allows querying the Study table by a combination of population and phenotype variables.

***get_snps()*:** Take a study list (possibly from *get_studies*) and return a SNP list or dataframe.

**Helpful additional functions:**

***intersecting_phenos()*:** Return a list of phenotypes or phenotype categories present in all queried populations.

***write_study_dict()*:** Write the dictionary returned from *get_studies(dictionary=True)* to a tab delimited file with extra data from the database.

**Lookup functions:** *lookup_rsid()*, *lookup_location()* and *lookup_studies()* allow the querying of the database for specific SNPs and can return customized information on them.

**MyVariant:**

***get_variant_info()*:** Use myvariant to get variant info for a list of SNPs.

**DataFrame Manipulation:**

***collapse_dataframe()*:** Collapse a dataframe (such as that returned by *get_snps()*) to include only a single entry per SNP (collapsing multiple studies into one).

***intersect_overlapping_series()*:** Merge two sets of pvalues (such as those from *collapse_dataframe()*) into a single merged dataframe with the original index and one column for each pvalue. Good for plotting.

### 3.1.1 get_studies

grasp.query.**get_studies**(*primary_phenotype=None*, *pheno_cats=None*, *pheno_cats_alias=None*, *primary_pop=None*, *has_pop=None*, *only_pop=None*, *has_disc_pop=None*, *has_rep_pop=None*, *only_disc_pop=None*, *only_rep_pop=None*, *query=False*, *count=False*, *dictionary=False*, *pandas=False*)

Return a list of studies filtered by phenotype and population.

There are two ways to query both phenotype and population.

**Phenotype:** GRASP provides a 'primary phenotype' for each study, which are fairly poorly curated. They also provide a list of phenotype categories, which are well curated. The problem with the categories is that there are multiple per study and some are to general to be useful. If using categories be sure to post filter the study list.

Note: I have made a list of aliases for the phenotype categories to make them easier to type. Use pheno_cats_alias for that.

**Population:** Each study has a primary population (list available with 'get_populations') but some studies also have other populations in the cohort. GRASP indexes all population counts, so those can be used to query also. To query these use *has_* or *only_* (exclusive) parameters, you can query either discovery populations or replication populations. Note that you cannot provide both *has_* and *only_* parameters for the same population type.

For doing population specific analyses most of the time you will want the excl_disc_pop query.

**Argument Description:** Phenotype Arguments are 'primary_phenotype', 'pheno_cats', and 'pheno_cats_alias'.

Only provide one of pheno_cats or pheno_cats_alias

Population Arguments are *primary_pop*, *has_pop*, *only_pop*, *has_disc_pop*, *has_rep_pop*, *only_disc_pop*, *only_rep_pop*.

*primary_pop* is a simple argument, the others use bitwise flags for lookup.

*has_pop* and *only_pop* simpply combine both the discovery and replication population lookups.

The easiest way to use the *has_* and *only_* parameters is with the PopFlag object. It uses *py-flags*. For example:

```
pops = PopFlag.eur | PopFlag.afr
```

In addition you can provide a list of strings corresponding to PopFlag attributes.

Note: the *only_* parameters work as ANDs, not ORs. So only_disc_pop='eur|afr' will return those studies that have BOTH european and african discovery populations, but no other discovery populations. On the other hand, *has_* works as an OR, and will return any study with any of the specified populations.

**Parameters**

- **primary_phenotype** – Phenotype of interest, string or list of strings.
- **pheno_cats** – Phenotype category of interest.
- **pheno_cats_alias** – Phenotype category of interest.
- **primary_pop** – Query the primary population, string or list of strings.
- **has_pop** – Return all studies with these populations
- **only_pop** – Return all studies with these populations

- **has_disc_pop** – Return all studies with these discovery populations
- **has_rep_pop** – Return all studies with these replication populations
- **only_disc_pop** – Return all studies with ONLY these discovery populations
- **only_rep_pop** – Return all studies with ONLY these replication populations
- **query** – Return the query instead of the list of study objects.
- **count** – Return a count of the number of studies.
- **dictionary** – Return a dictionary of title->id for filtering.
- **pandas** – Return a dataframe of study information instead of the list.

**Returns** A list of study objects, a query, or a dataframe.

### 3.1.2 get_snps

grasp.query.**get_snps**(*studies*, *pandas=True*)

Return a list of SNPs in a single population in a single phenotype.

**Studies** A list of studies.

**Pandas** Return a dataframe instead of a list of SNP objects.

**Returns** Either a DataFrame or list of SNP objects.

### 3.1.3 lookup_rsid

grasp.query.**lookup_rsid**(*rsid*, *study=False*, *columns=None*, *pandas=False*)

Query database by rsID.

**Parameters**

- **rsID** (*str*) – An rsID or list of rsIDs
- **study** (*bool*) – Include study info in the output.
- **columns** (*list*) – A list of columns to include in the query. Default is all. List must be made up of column objects, e.g. [t.SNP.snpid, t.Study.id]
- **pandas** (*bool*) – Return a dataframe instead of a list of SNPs

**Returns** List of SNP objects

**Return type** list

### 3.1.4 lookup_location

grasp.query.**lookup_location**(*chrom*, *position*, *study=False*, *columns=None*, *pandas=False*)

Query database by location.

**Parameters**

- **chrom** (*str*) – The chromosome as an int or string (e.g. 1 or chr1)
- **position** (*int*) – Either one location, a list of locations, or a range of locations, range can be expressed as a tuple of two ints, a range object, or a string of 'int-int'
- **study** (*bool*) – Include study info in the output.

- **columns** (*list*) – A list of columns to include in the query. Default is all. List must be made up of column objects, e.g. [t.SNP.snpid, t.Study.id]
- **pandas** (*bool*) – Return a dataframe instead of a list of SNPs

**Returns** List of SNP objects

**Return type** list

### 3.1.5 lookup_studies

grasp.query.**lookup_studies**(*title=None*, *study_id=None*, *columns=None*, *pandas=False*)

Find all studies matching either title or id.

**Parameters**

- **title** (str) – The study title, string or list of strings.
- **study_id** (int) – The row ID, usually the PMID, int or list of ints.
- **columns** (*list*) – A list of columns to include in the query. Default is all. List must be made up of column objects, e.g. [t.SNP.snpid, t.Study.id]
- **pandas** (*bool*) – Return a dataframe instead of a list of SNPs

**Returns** All matching studies as either a dataframe or a list

**Return type** DataFrame or list

### 3.1.6 get_variant_info

grasp.query.**get_variant_info**(*snp_list*, *fields='dbsnp'*, *pandas=True*)

Get variant info for a list of SNPs.

**Parameters**

- **snp_list** – A list of SNP objects or SNP rsIDs
- **fields** – Choose fields to display from: docs.myvariant.info/en/latest/doc/data.html#available-fields Good choices are 'dbsnp', 'clinvar', or 'gwassnps' Can also use 'grasp' to get a different version of this info.
- **pandas** – Return a dataframe instead of dictionary.

**Returns** A dictionary or a dataframe.

### 3.1.7 get_collapse_dataframe

grasp.query.**collapse_dataframe**(*df*, *mechanism='median'*, *pvalue_filter=None*, *protected_columns=None*)

Collapse a dataframe by chrom:location from get_snps.

Will use the mechanism defined by 'mechanism' to collapse a dataframe to one indexed by 'chrom:location' with pvalue and count only.

This function is agnostic to all dataframe columns other than:

```
['chrom', 'pos', 'snpid', 'pval']
```

All other columns are collapsed into a comma separated list, a string. 'chrom' and 'pos' are merged to become the new colon-separated index, snpid is maintained, and pval is merged using the function in 'mechanism'.

**Parameters**

- **df** – A pandas dataframe, must have 'chrom', 'pos', 'snpid', and 'pval' columns.
- **mechanism** – A numpy statistical function to use to collapse the pvalue, median or mean are the common ones.
- **pvalue_filter** – After collapsing the dataframe, filter to only include pvalues less than this cutoff.
- **protected_columns** – A list of column names that will be maintained as is, although all duplicates will be dropped (randomly). Only makes sense for columns that are identical for all studies of the same SNP.

**Returns**

**Indexed by chr:pos, contains flattened pvalue column, and** all original columns as a comma-separated list. Additionally contains a count and stddev (of pvalues) column. stddev is nan if count is 1.

**Return type** DataFrame

### 3.1.8 intersect_overlapping_series

grasp.query.**intersect_overlapping_series**(*series1*, *series2*, *names=None*, *stats=True*, *plot=None*, *name=None*)

Plot all SNPs that overlap between two pvalue series.

**Parameters**

- **series{1,2}** (*Series*) – A pandas series object
- **names** (*list*) – A list of two names to use for the resultant dataframes
- **stats** (*bool*) – Print some stats on the intersection
- **plot** (str) – Plot the resulting intersection, path to save figure to (must end in .pdf/.png). Numpy and Matplotlib are required.
- **name** (str) – A name for the plot, optional.

**Returns** with the two series as columns

**Return type** DataFrame

### 3.1.9 write_study_dict

grasp.query.**write_study_dict**(*study_dict*, *outfile*)

Write a study dictionary from *get_studies(dictionary=True)* to file.

Looks up studies in the Study table first.

**Parameters**

- **study_dict** (*dict*) – A dictionary of title=>id from the Study table.
- **outfile** – A valid path to a file with write permission.

**Outputs:** A tab delimited file of ID, Title, Author, Journal, Discovery Population, Replication Population, SNP_Count

**Returns** None

## 3.2 grasp.tables

GRASP table descriptions in SQLAlchemy ORM.

These tables do not exist in the GRASP data, which is a single flat file. By separating the data into these tables querying is much more efficient.

This submodule should only be used for querying.

### 3.2.1 SNP

**class** `grasp.tables.`**`SNP`**(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`

An SQLAlchemy Talble for GRASP SNPs.

Study and phenotype information are pushed to other tables to minimize table size and make querying easier.

**Table Name:** snps

**Columns:** Described in the columns attribute

**`int`**
The ID number of the SNP, usually the NHLBIkey

**`str`**
SNP loction expressed as 'chr:pos'

**`hvgs_ids`**
A list of HGVS IDs for this SNP

**`columns`**
A dictionary of all columns 'column_name'=>('type', 'desc')

**`columns = OrderedDict([('id', ('BigInteger', 'NHLBIkey')), ('snpid', ('String', 'SNPid')), ('chrom', ('String', 'chr')), ('p`**
A description of all columns in this table.

**`display_columns`**(*display_as='table'*, *write=False*)
Return all columns in the table nicely formatted.

**Display choices:** table: A formatted grid-like table tab: A tab delimited non-formatted version of table list: A string list of column names

**Parameters**

- **`display_as`** – {table,tab,list}
- **`write`** – If true, print output to console, otherwise return string.

**Returns** A formatted string or None

**`get_columns`**(*return_as='list'*)
Return all columns in the table nicely formatted.

**Display choices:** list: A python list of column names dictionary: A python dictionary of name=>desc long_dict: A python dictionary of name=>(type, desc)

**Parameters** **return_as** – {table,tab,list,dictionary,long_dict,id_dict}

**Returns** A list or dictionary

**get_variant_info**(*fields='dbsnp'*, *pandas=True*)
Use the myvariant API to get info about this SNP.

Note that this service can be very slow. It will be faster to query multiple SNPs.

**Parameters**

- **fields** – Choose fields to display from: docs.myvariant.info/en/latest/doc/data.html#available-fields Good choices are 'dbsnp', 'clinvar', or 'gwassnps' Can also use 'grasp' to get a different version of this info.
- **pandas** – Return a dataframe instead of dictionary.

**Returns** A dictionary or a dataframe.

**hvgs_ids**
The HVGS ID from myvariant.

**snp_loc**
Return a simple string containing the SNP location.

### 3.2.2 Study

**class** grasp.tables.**Study**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

An SQLAlchemy table to store study information.

This table provides easy ways to query for SNPs by study information, including population and phenotype.

Note: *disc_pop_flag* and *rep_pop_flag* are integer representations of a bitwise flag describing population, defined in ref.PopFlag. To see the string representation of this property, lookup *disc_pops* or *rep_pops*.

**Table Name:** studies

**Columns:** Described in the columns attribute.

**int**
The integer ID number, usually the PMID, unless not indexed.

**str**
Summary data on this study.

**len**
The number of individuals in this study.

**disc_pops**
A string displaying the number of discovery poplations.

**rep_pops**
A string displaying the number of replication poplations.

**columns**
A dictionary of all columns 'column_name'=>('type', 'desc')

**population_information**
A multi-line string describing the populations in this study.

**columns = OrderedDict([('id', ('Integer', 'id')), ('pmid', ('String', 'PubmedID')), ('title', ('String', 'Study')), ('journal',**
A description of all columns in this table.

**disc_pops**
Convert disc_pop_flag to PopFlag.

**display_columns**(*display_as='table'*, *write=False*)
Return all columns in the table nicely formatted.

**Display choices:** table: A formatted grid-like table tab: A tab delimited non-formatted version of table list: A string list of column names

**Parameters**

- **display_as** – {table,tab,list}
- **write** – If true, print output to console, otherwise return string.

**Returns** A formatted string or None

**get_columns**(*return_as='list'*)
Return all columns in the table nicely formatted.

**Display choices:** list: A python list of column names dictionary: A python dictionary of name=>desc long_dict: A python dictionary of name=>(type, desc)

**Parameters** **return_as** – {table,tab,list,dictionary,long_dict,id_dict}

**Returns** A list or dictionary

**pops**
Convert rep_pop_flag to PopFlag.

**population_information**
Display a summary of population data.

**rep_pops**
Convert rep_pop_flag to PopFlag.

### 3.2.3 Phenotype

**class** grasp.tables.**Phenotype**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

An SQLAlchemy table to store the primary phenotype.

**Table Name:** phenos

**Columns:** phenotype: The string phenotype from the GRASP DB, unique. alias: A short representation of the phenotype, not unique. studies: A link to the studies table.

**int**
The ID number.

**str**
The name of the phenotype.

### 3.2.4 PhenoCats

**class** grasp.tables.**Phenotype**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

An SQLAlchemy table to store the primary phenotype.

**Table Name:** phenos

**Columns:** phenotype: The string phenotype from the GRASP DB, unique. alias: A short representation of the phenotype, not unique. studies: A link to the studies table.

**int**
The ID number.

**str**
The name of the phenotype.

### 3.2.5 Population

**class** grasp.tables.**Population**(*population*)
Bases: sqlalchemy.ext.declarative.api.Base

An SQLAlchemy table to store the platform information.

**Table Name:** populations

**Columns:** population: The name of the population. studies: A link to all studies in this population. snps: A link to all SNPs in this populations.

**int**
Population ID number

**str**
The name of the population

### 3.2.6 Platform

**class** grasp.tables.**Platform**(*platform*)
Bases: sqlalchemy.ext.declarative.api.Base

An SQLAlchemy table to store the platform information.

**Table Name:** platforms

**Columns:** platform: The name of the platform from GRASP. studies: A link to all studies using this platform.

**int**
The ID number of this platform

**str**
The name of the platform

## 3.3 grasp.db

Functions for managing the GRASP database.

*get_session()* is used everywhere in the module to create a connection to the database. *initialize_database()* is used to build the database from the GRASP file. It takes about an hour 90 minutes to run and will overwrite any existing database.

grasp.db.**get_session**(*echo=False*)
Return a session and engine, uses config file.

**Parameters** **echo** – Echo all SQL to the console.

**Returns**

**A SQLAlchemy session and engine object corresponding** to the grasp database for use in querying.

**Return type** session, engine

grasp.db.**initialize_database**(*study_file*, *grasp_file*, *commit_every=250000*, *progress=False*)
Create the database quickly.

**Study_file** Tab delimited GRASP study file, available here: github.com/MikeDacre/grasp/blob/master/grasp_studies.txt

**Grasp_file** Tab delimited GRASP file.

**Commit_every** How many rows to go through before commiting to disk.

**Progress** Display a progress bar (db length hard coded).

## 3.4 grasp.config

Manage a persistent configuration for the database.

grasp.config.**config = <configparser.ConfigParser object>**
A globally accessible ConfigParger object, initialized with CONFIG_FILE.

grasp.config.**CONFIG_FILE = '/home/dacre/.grasp'**
The PATH to the config file.

grasp.config.**init_config**(*db_type*, *db_file=''*, *db_host=''*, *db_user=''*, *db_pass=''*)
Create an initial config file.

**Parameters**

- **db_type** – 'sqlite/mysql/postgresql'
- **db_file** – PATH to sqlite database file
- **db_host** – Hostname for mysql or postgresql server
- **db_user** – Username for mysql or postgresql server
- **db_pass** – Password for mysql or postgresql server (not secure)

**Returns** NoneType

**Return type** None

grasp.config.**init_config_interactive**()
Interact with the user to create a new config.

Uses readline autocompletion to make setup easier.

grasp.config.**write_config**()
Write the current config to CONFIG_FILE.

## 3.5 grasp.info

Little functions to pretty print column lists and category info.

get_{phenotypes,phenotype_categories,popululations} all display a dump of the whole database.

get_population_flags displays available flags from PopFlag.

display_{study,snp}_columns displays a list of available columns in those two tables as a formatted string.

get_{study,snp}_columns return a list of available columns in those two tables as python objects.

grasp.info.**display_snp_columns**(*display_as='table'*, *write=False*)
Return all columns in the SNP table as a string.

**Display choices:** table: A formatted grid-like table tab: A tab delimited non-formatted version of table list: A string list of column names

**Parameters**

- **display_as** – {table,tab,list}
- **write** – If true, print output to console, otherwise return string.

**Returns** A formatted string or None

grasp.info.**display_study_columns**(*display_as='table'*, *write=False*)
Return all columns in the Study table as a string.

**Display choices:** table: A formatted grid-like table tab: A tab delimited non-formatted version of table list: A string list of column names

**Parameters**

- **display_as** – {table,tab,list}
- **write** – If true, print output to console, otherwise return string.

**Returns** A formatted string or None

grasp.info.**get_phenotype_categories**(*list_only=False*, *dictionary=False*, *table=False*)
Return all phenotype categories from the PhenoCats table.

**List_only** Return a simple text list instead of a list of Phenotype objects.

**Dictionary** Return a dictionary of phenotype=>ID

**Table** Return a pretty table for printing.

grasp.info.**get_phenotypes**(*list_only=False*, *dictionary=False*, *table=False*)
Return all phenotypes from the Phenotype table.

**List_only** Return a simple text list instead of a list of Phenotype objects.

**Dictionary** Return a dictionary of phenotype=>ID

**Table** Return a pretty table for printing.

grasp.info.**get_population_flags**(*list_only=False*, *dictionary=False*, *table=False*)
Return all population flags available in the PopFlags class.

**List_only** Return a simple text list instead of a list of Phenotype objects.

**Dictionary** Return a dictionary of population=>ID

**Table** Return a pretty table for printing.

grasp.info.**get_populations**(*list_only=False*, *dictionary=False*, *table=False*)
Return all populatons from the Population table.

**List_only** Return a simple text list instead of a list of Phenotype objects.

**Dictionary** Return a dictionary of population=>ID

**Table** Return a pretty table for printing.

grasp.info.**get_snp_columns**(*return_as='list'*)
Return all columns in the SNP table.

**Display choices:** list: A python list of column names dictionary: A python dictionary of name=>desc long_dict: A python dictionary of name=>(type, desc)

**Parameters** **return_as** – {table,tab,list,dictionary,long_dict,id_dict}

**Returns** A list or dictionary

grasp.info.**get_study_columns**(*return_as='list'*)
Return all columns in the SNP table.

**Display choices:** list: A python list of column names dictionary: A python dictionary of name=>desc long_dict: A python dictionary of name=>(type, desc)

**Parameters** **return_as** – {table,tab,list,dictionary,long_dict,id_dict}

**Returns** A list or dictionary

## 3.6 grasp.ref

*ref.py* holds some simple lookups and the *PopFlags* classes that don't really go anywhere else. Holds reference objects for use elsewhere in the module.

**class** grasp.ref.**PopFlag**
Bases: flags.Flags

A simplified bitwise flag system for tracking populations.

Based on py-flags

**eur**
1 # European

**afr**
2 # African ancestry

**east_asian**
4 # East Asian

**south_asian**
8 # Indian/South Asian

**his**
16 # Hispanic

**native**
32 # Native

**micro**
64 # Micronesian

**arab**
128 # Arab/ME

**mix**
256 # Mixed

**uns**
512 # Unspec

**filipino**
1024 # Filipino

**indonesian**
2048 # Indonesian

**Example::** eur = PopFlag.eur afr = PopFlag(2) his_micro = PopFlag.from_simple_string('his|micro') four_pops = eur | afr four_pops |= his_micro assert four_pops == PopFlag.from_simple_string('his|micro|afr|eur') PopFlag.eur & four_pops > 0 # Returns True PopFlag.eur i== four_pops # Returns False PopFlag.arab & four_pops > 0 # Returns False

CHAPTER 4

# Table Columns

The two important tables with the majority of the data are Study and SNP. In addition, phenotype data is stored in Phenotype and PhenoCats, population data is in Population, and platforms are in Platform.

**Contents**



## 4.1 Study

To query studies, it is recommended to use the query.get_studies() function.

| Column | Description | Type |
|---|---|---|
| id | id | Integer |
| pmid | PubmedID | String |
| title | Study | String |
| journal | Journal | String |
| author | 1st_author | String |
| grasp_ver | GRASPversion? | Integer |
| noresults | No results flag | Boolean |
| results | #results | Integer |
| qtl | IsEqtl/meQTL/pQTL/gQTL/Metabolmics? | Boolean |
| snps | Link to all SNPs in this study | relationship |
| phenotype_id | ID of primary phenotype in Phenotype table | Integer |
| phenotype | A link to the primary phenotype in the Phenotype table | relationship |
| phenotype_cats | A link to all phenotype categories assigned in the PhenoCats table | relationship |
| datepub | DatePub | Date |

Continued on next page

Table 4.1 – continued from previous page

| Column | Description | Type |
|---|---|---|
| in_nhgri | In NHGRI GWAS catalog (8/26/14)? | Boolean |
| locations | Specific place(s) mentioned for samples | String |
| mf | Includes male/female only analyses in discovery and/or replication? | Boolean |
| mf_only | Exclusively male or female study? | Boolean |
| platforms | Link to platforms in the Platform table. Platform [SNPs passing QC] | relationship |
| snp_count | From "Platform [SNPs passing QC]" | String |
| imputed | From "Platform [SNPs passing QC]" | Boolean |
| population_id | Primary key of population table | Integer |
| population | GWAS description, link to table | relationship |
| total | Total Discovery + Replication sample size | Integer |
| total_disc | Total discovery samples | Integer |
| pop_flag | A bitwise flag that shows presence/absence of all populations (discovery and replication) | Integer |
| disc_pop_flag | A bitwise flag that shows presence/absence of discovery populations | Integer |
| european | European | Integer |
| african | African ancestry | Integer |
| east_asian | East Asian | Integer |
| south_asian | Indian/South Asian | Integer |
| hispanic | Hispanic | Integer |
| native | Native | Integer |
| micronesian | Micronesian | Integer |
| arab | Arab/ME | Integer |
| mixed | Mixed | Integer |
| unpecified | Unspec | Integer |
| filipino | Filipino | Integer |
| indonesian | Indonesian | Integer |
| total_rep | Total replication samples | Integer |
| rep_pop_flag | A bitwise flag that shows presence/absence of replication populations | Integer |
| rep_european | European.1 | Integer |
| rep_african | African ancestry.1 | Integer |
| rep_east_asian | East Asian.1 | Integer |
| rep_south_asian | Indian/South Asian.1 | Integer |
| rep_hispanic | Hispanic.1 | Integer |
| rep_native | Native.1 | Integer |
| rep_micronesian | Micronesian.1 | Integer |
| rep_arab | Arab/ME.1 | Integer |
| rep_mixed | Mixed.1 | Integer |
| rep_unpecified | Unspec.1 | Integer |
| rep_filipino | Filipino.1 | Integer |
| rep_indonesian | Indonesian.1 | Integer |
| sample_size | Initial Sample Size, string description of integer population counts above. | String |
| replication_size | Replication Sample Size, string description of integer population counts above. | String |

## 4.2 SNP

| Column | Description | Type |
|---|---|---|
| id | NHLBIkey | BigInteger |
| snpid | SNPid | String |

Continued on next page

Table 4.2 – continued from previous page

| Column | Description | Type |
|---|---|---|
| chrom | chr | String |
| pos | pos | Integer |
| pval | Pvalue | Float |
| NHLBIkey | NHLBIkey | String |
| HUPfield | HUPfield | String |
| LastCurationDate | LastCurationDate | Date |
| CreationDate | CreationDate | Date |
| population_id | Primary | Integer |
| population | Link | relationship |
| study_id | Primary | Integer |
| study | Link | relationship |
| study_snpid | SNPid | String |
| paper_loc | LocationWithinPaper | String |
| phenotype_desc | Phenotype | String |
| phenotype_cats | Link | relationship |
| InGene | InGene | String |
| NearestGene | NearestGene | String |
| InLincRNA | InLincRNA | String |
| InMiRNA | InMiRNA | String |
| InMiRNABS | InMiRNABS | String |
| dbSNPfxn | dbSNPfxn | String |
| dbSNPMAF | dbSNPMAF | String |
| dbSNPinfo | dbSNPalleles | String |
| dbSNPvalidation | dbSNPvalidation | String |
| dbSNPClinStatus | dbSNPClinStatus | String |
| ORegAnno | ORegAnno | String |
| ConservPredTFBS | ConservPredTFBS | String |
| HumanEnhancer | HumanEnhancer | String |
| RNAedit | RNAedit | String |
| PolyPhen2 | PolyPhen2 | String |
| SIFT | SIFT | String |
| LSSNP | LS | String |
| UniProt | UniProt | String |
| EqtlMethMetabStudy | EqtlMethMetabStudy | String |

## 4.3 Phenotype

All available phenotypes are available on the Phenotypes wiki page

- id
- phenotype
- studies (link to Study table)
- snps (link to SNP table)

## 4.4 PhenoCats

All phenotype categories are available on the Phenotype Categories wiki page

- id
- population
- alias
- studies (link to Study table)
- snps (link to SNP table)

## 4.5 Population

- id
- population
- studies (link to Study table)
- snps (link to SNP table)

All population entries are available on the Populations wiki page

## 4.6 Platform

- id
- platform
- studies (link to Study table)
- snps (link to SNP table)

## g

## R



## S



## U



## W