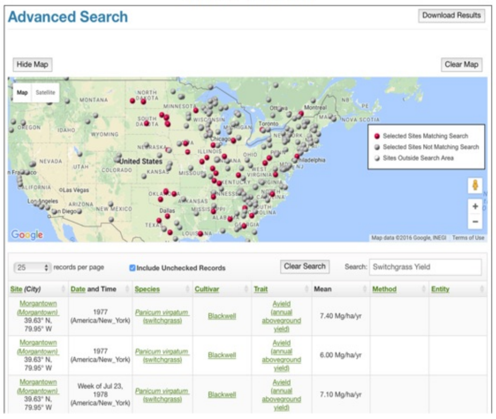
# BETYdb Documentation: Volume 1: Data Access

Basic Search

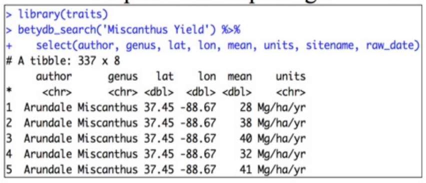

Advanced Search
Download Results
Hide Map
Clear Map
Map
Satellite
Selected Sites Matching Search
Selected Sites Not Matching Search
Sites Outside Search Area
Map data ©2016 Google, INEGI
Terms of Use
records per page
Include Unchecked Records
Clear Search

25 records per page

Search: Switchgrass Yield

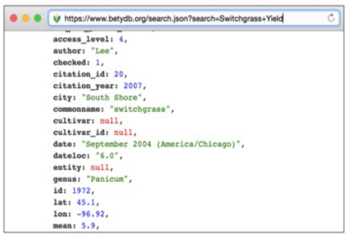
| Site (City) | Date and Time | Species | Cultivar | Trait | Mean | Method | Entity |
|---|---|---|---|---|---|---|---|
| Morgantown (Morgantown) 39.63° N, 79.95° W | 1977 (America/New_York) | Panicum virgatum (switchgrass) | Blackwell | Ayield (annual aboveground yield) | 7.40 Mg/ha/yr | | |
| Morgantown (Morgantown) 39.63° N, 79.95° W | 1977 (America/New_York) | Panicum virgatum (switchgrass) | Blackwell | Ayield (annual aboveground yield) | 6.00 Mg/ha/yr | | |
| Morgantown (Morgantown) 39.63° N, 79.95° W | Week of Jul 23, 1978 (America/New_York) | Panicum virgatum (switchgrass) | Blackwell | Ayield (annual aboveground yield) | 7.10 Mg/ha/yr | | |

rOpenSci traits package



```
> library(traits)
> betydb_search('Miscanthus Yield') %>%
+    select(author, genus, lat, lon, mean, units, sitename, raw_date)
# A tibble: 337 x 8
     author      genus   lat    lon  mean    units
*     <chr>      <chr> <dbl>  <dbl> <dbl>    <chr>
1  Arundale Miscanthus 37.45 -88.67    28 Mg/ha/yr
2  Arundale Miscanthus 37.45 -88.67    38 Mg/ha/yr
3  Arundale Miscanthus 37.45 -88.67    40 Mg/ha/yr
4  Arundale Miscanthus 37.45 -88.67    32 Mg/ha/yr
5  Arundale Miscanthus 37.45 -88.67    41 Mg/ha/yr
```

URL based API



https://www.betydb.org/search.json?search=Switchgrass+Yield

```
access_level: 4,
author: "Lee",
checked: 1,
citation_id: 20,
citation_year: 2007,
city: "South Shore",
commonname: "switchgrass",
cultivar: null,
cultivar_id: null,
date: "September 2004 (America/Chicago)",
dateloc: "6.0",
entity: null,
genus: "Panicum",
id: 1972,
lat: 45.1,
lon: -96.92,
mean: 5.9,
```

Energy Biosciences Institute BETYdb | Biofuel Ecophysiological Traits and Yields Database

# Table of Contents

# Guide to accessing data from BETYdb

## What is covered here:

There are many ways to access data in BETYdb. This document is written for people who want to get data from BETYdb through either 1) the web browser interface search box or 2) the more powerful URL-based methods of submitting queries programmatically. These URL-based queries are the basis of the rOpensci 'traits' package that allows users to retrieve data in R.

## What is not covered:

While these methods allow users to access these data, some users may wish to use BETYdb actively in their research. If you want to contribute to BETYdb, see the Data Entry documentation. That document teaches users how they can curate their own data using BETYdb—either via the primary instance hosted at https://www.betydb.org or as a distributed node on the BETYdb network that enables data sharing.

Many people who contribute data—and others who want to submit more complex queries—can benefit from accessing a PostgreSQL server. Users who want to install a BETYdb PostgreSQL database (just ask!) are able to enter, use, and share data within and across a network of synced instances of BETYdb (see the BETYdb Technical Documentation topic "Distributed BETYdb"). This network shares data that has not necessarily been independently reviewed beyond the correction of extreme outliers. In practice, data sets in active use by researchers have been evaluated for meaning by researchers. Expert knowledge is encoded in the `priors` table, in the PEcAn meta-analysis models, and in the Ecosystem models parameterized by the meta-analysis results. Users with access to a BETYdb PostgreSQL database can access data through R packages—in particular `PEcAn.DB`, which is designed for use in PEcAn—or through any programming language a scientist is likely to use. When not using PEcAn, I recommend the R package `dplyr` unless you prefer constructing joins directly in SQL.

## Data sharing

Data is made available for analysis after it is submitted and reviewed by a database administrator. These data are suitable for basic scientific research and modeling. All reviewed data are made publicly available after publication to users of BETYdb who are conducting primary research.

If you want data that has not been reviewed, please request access. There are many records that are not publicly available through the web API, but that can be accessed by installing a copy of the BETYdb PostgreSQL database.

## Access restrictions

All public data in BETYdb is made available under the Open Data Commons Attribution License (ODC-By) v1.0. You are free to share, create, and adapt its contents. For tables having an `access_level` column, data in rows where the column value is 1 or 2 is not covered by this license, but may be available for use with consent.

## Quality flags

In tables and views containing trait and yield data, the `checked` column value indicates if data has been checked (1), has *not* been checked (0), or has been identified as incorrect (-1). Checked data has been independently reviewed. By default, only *checked* data is provided through the web browser interface and API. As of BETYdb 4.4 (Oct 14 2015), users can opt-in to viewing and downloading unchecked data either by way of a web browser or via the API. Unchecked data may also be loaded into a local PostgreSQL database simply by running the script `./load.bety.sh`.

## Citation

Please cite the source of data as:

> LeBauer, David; Dietze, Michael; Kooper, Rob; Long, Steven; Mulrooney, Patrick; Rohde, Gareth Scott; Wang, Dan; (2010): Biofuel Ecophysiological Traits and Yields Database (BETYdb); Energy Biosciences Institute, University of Illinois at Urbana-Champaign. http://dx.doi.org/10.13012/J8H41PB9

In publications, please also include a copy of the data that you used and information about the original publication. The *original publication* data corresponds to the citation author, year, and title in database queries.

It is important to archive the data used in a particular analysis so that someone can reproduce your results and check your assumptions. Keep in mind, though, that we are continually collecting new data and checking existing data: BETYdb is not a static source of data.

## A brief overview of some key topics

**Data access**

Data is made available for analysis after it is submitted and reviewed by a database administrator. These data are suitable for basic scientific research and modeling. All reviewed data are made publicly available after publication to users of BETYdb who are conducting primary research. Access to these raw data is provided to users based on affiliation and contribution of data.

**Search box**

On the Welcome page of BETYdb there is a search option for trait and yield data. This tool allows users to search the entire database for specific sites, species, and traits.

**Enhanced search capabilities: the Advanded Search page**

The result of a search using the Welcome page search box is displayed on its own page, called the Advanced Search page. This page has additional search features, including map-based searching.

Following a search using the search box, one may click the "Search Using Map" button. The results of the search are then shown in Google Maps, that is, each site represented in the search result will be highlighted on the map. Moreover, a search can be further refined by site location by clicking locations of interest on the map.

Another feature of the Advanced Search page is the "Download Results" button; clicking it will send a CSV file of the most recent search result to your computer.

**General CSV downloads**

Data from any database table can be downloaded as a CSV file.

For example[1], to download all of the Switchgrass (Panicum virgatum) yield data, open the BETYdb Welcome page at https://www.betydb.org, select Species under Data menu. Type "Panicum virgatum" in the search box and click on the "Show" icon in the Actions column of the row for Panicum virgatum. You should see the URL `https://www.betydb.org/species/938` in the address box after the View Species page is displayed. The trailing number in the URL —938—is the id number of the species Panicum virgatum. You can now use this number to search for yields for this species.

To do so, enter the URL for the species listing page, https://www.betydb.org/yields, into the browser address box, append ".csv" to indicate that you want results in CSV format, and append the query string "?specie_id=938" to indicate that you want only the results for Panicum virgatum. Your browser should then download a CSV file containing all yield records (subject to your user permissions) for Panicum virgatum.

This is just one of a number of ways of downloading data from BETYdb. See the API for URL-based Queries for additional information.

### API keys

Using an API key allows access to data without having to enter a login. For example, assuming "Kq2xEWiRCyj90Sq7EFVn2A0LxJb58UTbscJl4fJP" is a valid API key[2], you could visit the URL https://www.betydb.org/yields.csv?specie_id=938&key=Kq2xEWiRCyj90Sq7EFVn2A0LxJb58UTbscJl4fJP" directly (that is, without first logging in) to download a CSV file of Panicum virgatum yield data.

### Installing BETYdb (the complete database)

See Installing your own version of BETYdb.

---

> 1. This example is revisited in the section Original API, where the same information is retrieved in XML format by issuing GET requests to URLs. ↩

> 2. This is the guest user's API key at the time of writing. ↩

# The Advanced Search Box

Most tables in BETYdb have search boxes; for example, https://www.betydb.org/citations and https://www.betydb.org/sites. We describe below how to query these tables and download the results as CSV, JSON, or XML. The Advanced Search box is the easiest way to download summary datasets designed to have enough information (location, time, species, citations) to be useful for a wide range of use cases.

## Using the Search Box

On the Welcome page of BETYdb there is a search option for trait and yield data (Figure 1). This tool allows users to search the entire collection of trait and yield data for specific sites, citations, species, and traits.

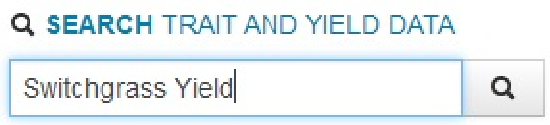
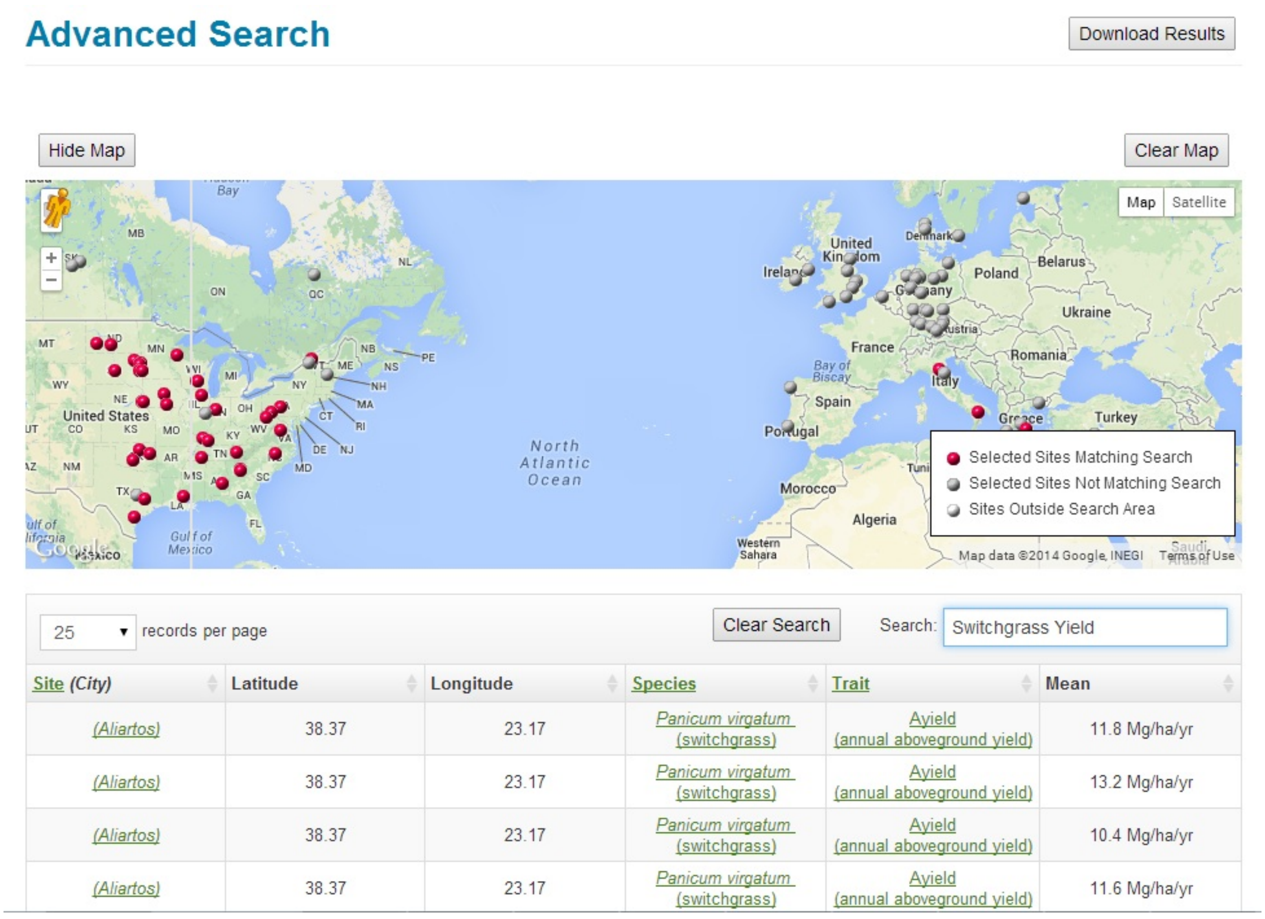


*Figure 1: BETYdb Advanced Search box—searching for switchgrass yield data*

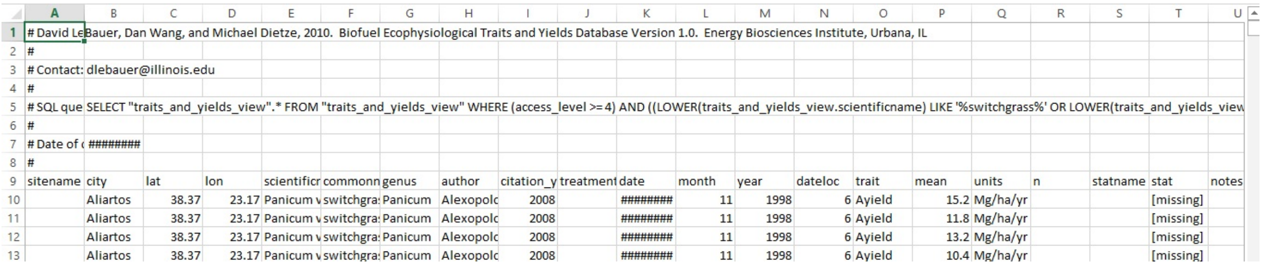
The results page provides a map interface (Figure 2) and the option to download a file containing search results.



| Site (City) | Latitude | Longitude | Species | Trait | Mean |
|---|---|---|---|---|---|
| (Aliartos) | 38.37 | 23.17 | Panicum virgatum (switchgrass) | Ayield (annual aboveground yield) | 11.8 Mg/ha/yr |
| (Aliartos) | 38.37 | 23.17 | Panicum virgatum (switchgrass) | Ayield (annual aboveground yield) | 13.2 Mg/ha/yr |
| (Aliartos) | 38.37 | 23.17 | Panicum virgatum (switchgrass) | Ayield (annual aboveground yield) | 10.4 Mg/ha/yr |
| (Aliartos) | 38.37 | 23.17 | Panicum virgatum (switchgrass) | Ayield (annual aboveground yield) | 11.6 Mg/ha/yr |

*Figure 2: Results of searching for switchgrass yields, including an interactive map showing sites for which data is available*

The downloaded file (Figure 3) is in CSV format. This file provides meta-data and provenance information, including: the SQL query used to extract the data, the date and time the query was made, the citation source of each result row, and a citation for BETYdb itself (*to be updated on publication*).



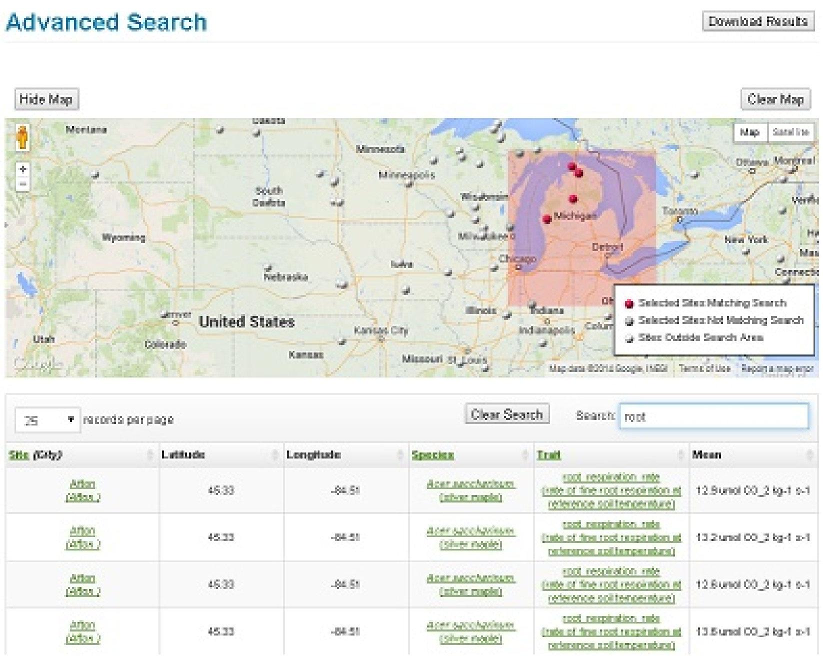
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | # David LeBauer, Dan Wang, and Michael Dietze, 2010. Biofuel Ecophysiological Traits and Yields Database Version 1.0. Energy Biosciences Institute, Urbana, IL | | | | | | | | | | | | | | | | | | | | |
| 2 | # | | | | | | | | | | | | | | | | | | | | |
| 3 | # Contact: | dlebauer@illinois.edu | | | | | | | | | | | | | | | | | | | |
| 4 | # | | | | | | | | | | | | | | | | | | | | |
| 5 | # SQL que | SELECT "traits_and_yields_view".* FROM "traits_and_yields_view" WHERE (access_level >= 4) AND ((LOWER(traits_and_yields_view.scientificname) LIKE '%switchgrass%' OR LOWER(traits_and_yields_view | | | | | | | | | | | | | | | | | | | |
| 6 | # | | | | | | | | | | | | | | | | | | | | |
| 7 | # Date of ( | ######### | | | | | | | | | | | | | | | | | | | |
| 8 | # | | | | | | | | | | | | | | | | | | | | |
| 9 | sitename | city | lat | lon | scientificr | commonn | genus | author | citation_y | treatment | date | month | year | dateloc | trait | mean | units | n | statname | stat | notes |
| 10 | | Aliartos | 38.37 | 23.17 | Panicum v | switchgra: | Panicum | Alexopolc | 2008 | | ######### | 11 | 1998 | 6 | Ayield | 15.2 | Mg/ha/yr | | | [missing] | |
| 11 | | Aliartos | 38.37 | 23.17 | Panicum v | switchgra: | Panicum | Alexopolc | 2008 | | ######### | 11 | 1998 | 6 | Ayield | 11.8 | Mg/ha/yr | | | [missing] | |
| 12 | | Aliartos | 38.37 | 23.17 | Panicum v | switchgra: | Panicum | Alexopolc | 2008 | | ######### | 11 | 1998 | 6 | Ayield | 13.2 | Mg/ha/yr | | | [missing] | |
| 13 | | Aliartos | 38.37 | 23.17 | Panicum v | switchgra: | Panicum | Alexopolc | 2008 | | ######### | 11 | 1998 | 6 | Ayield | 10.4 | Mg/ha/yr | | | [missing] | |

*Figure 3: The download result file from a search for switchgrass yields, displayed in Excel. This shows the meta-data the the first few query results out of several hundred.*

## Step-by-step instructions

Using the search box to search trait and yield data is very simple: Type the site (city or site name), species (scientific or common name), cultivar, citation (author and/or year), or trait (variable name or description) into the search box and the results will show contents of BETYdb that match the search. The number of records per page can be changed to accord with the viewer's preference and the search results can be downloaded in the Excel-compatible CSV format.

The *search map* may be used in conjunction with search terms to restrict search results to a particular geographical area—or even a specific site—by clicking on a map. Clicking on a particular site will restrict results to that site. Clicking in the vicinity of a group of sites but not on a particular site will restrict the search to the region around the point clicked. Alternatively, if a search using search terms is done without clicking on the map, all sites associated with the returned results are highlighted on the map. Then, to zero in on results for a particular geographic area, click on or near highlighted locations on the map (Figure 4).



| Site (City) | Latitude | Longitude | Species | Trait | Mean |
|---|---|---|---|---|---|
| Afton (Afton ) | 45.33 | -84.51 | Acer saccharinum (silver maple) | root_respiration_rate (rate of fine root respiration at reference soil temperature) | 12.8 umol CO_2 kg-1 s-1 |
| Afton (Afton ) | 45.33 | -84.51 | Acer saccharinum (silver maple) | root_respiration_rate (rate of fine root respiration at reference soil temperature) | 13.2 umol CO_2 kg-1 s-1 |
| Afton (Afton ) | 45.33 | -84.51 | Acer saccharinum (silver maple) | root_respiration_rate (rate of fine root respiration at reference soil temperature) | 12.6 umol CO_2 kg-1 s-1 |
| Afton (Afton ) | 45.33 | -84.51 | Acer saccharinum (silver maple) | root_respiration_rate (rate of fine root respiration at reference soil temperature) | 13.6 umol CO_2 kg-1 s-1 |

*Figure 4: Refining the search area*

# Details

## Basic search: Searching from the Home Page

1. Go to the home page betydb.org.

2. Enter one or more terms in the search box. The search will succeed if (and only if) each term in the search box matches one of the following columns of the `traits_and_yields_view`: `sitename`, `city`, `scientificname`, `cultivar`, `commonname`, `author`, `trait`, `trait_description`, `entity`, or `citation_year`. Thus, the more terms entered, the more restrictive the search. Note that a search term *matches* a column value if it occurs anywhere within the text of that column value and that matches are case-insensitive. Thus, the term "grass" would match a row with commonname column value "switchgrass" or "tussock cottongrass", a row with site name "Grassland Soil & Water Research Laboratory", or a row with trait description "for grasses, stem internode length". There is no way to restrict which of the ten match columns a given term is matched against except perhaps by making the term itself more specific; in the example just given, typing "cottongrass" rather than just "grass" would make it more likely that only the match against the `commonname` column would succeed.
3. Click the search icon next to the search box (or press the ENTER key). This will take one to the search results/"Advanced Search" page and show the result of the search.

## Advanced search: Searching from the Advanced Search page

1. From the home page, you can get to the advanced search page simply by typing search terms into the search box and clicking on the search icon. (Or just click the search icon without entering any terms. Or enter the URL for the advanced search page directly: https://www.betydb.org/search)
2. The search will return results for both traits and yields.
3. Enter one or more terms in the main search box. As with basic searches, each term must match one of the ten searched columns.
4. Search results are updated as you type, but you may press the `Enter` key to do a formal search form submission; this will reload the page and update the URL in the browser address box to include all of the parameters of your search. You may want to do this, for example, if you want to copy and store a URL that can be used to replicate the search.
5. Check the `Include Unchecked Records` box if you want to include results that haven't yet passed a quality-assurance check.
6. You can adjust the number of results show per page by adjusting the number in the `results per page` dropdown box.

7. By default, search results are sorted by site name. They can be resorted by date, species name, cultivar name, trait variable name, method name, or entity name by clicking the link in the corresponding column heading. (Click the same column heading twice to reverse the sort order.)

8. As mentioned above, you can filter search results geographically using the search map. To display the search map, click `Search Using Map`.

## Example

1. Go to https://www.betydb.org.

2. **Task 1 (simple search)**: Search for sugarcane yields — Enter the search string "sugarcane Ayield" in the search box.

3. You will see a new page with a table of results showing data about sugarcane yields.

4. **Task 2 (advanced search)**: Limit results to a specific site — We just want to see the results for the 'Zentsuji' site. Add a space and the word 'Zentsuji' to the search box.

5. You will see only the sugarcane yields for the 'Zentsuji' site.

6. **Task 3 (advanced search)**: Limit results to a specific trait — We only want to see the results concerning stemN instead of annual yield. Replace the word 'Ayield' with the word 'stemN' in the search box.

7. The resulting table rows all concern stem nitrogen data for sugarcane at the Zentsuji site.

# The BETYdb Web API

The BETYdb URL-based API allows users to query data.

The BETYdb Application Programming Interface (API) is a means of obtaining search or query results in CSV, JSON, or XML format from URL-based queries entered into a Web browser or called from a scripting language such as Bash, R, Matlab, or Python. All the tables in BETYdb are RESTful, which allows you to use GET, POST, PUT, or DELETE methods without interacting with the web browser-based GUI interface. The primary advantage of this approach is the ability to submit complex queries that access information and tables without having to do so through the GUI interface: text-based queries can be recorded with more precision than instructions for interacting with the GUI interface and can be constructed automatically within a scripting language.

## Versions

- **original**: Works but is a little bit clunky and limited.
- **beta**: Is more standardized than the original; supports *inserting* data as well as requesting data (see Inserting New Traits Via the API in the BETYdb Data Entry Workflow documentation).
- **v1** (to be released): Will work the same as beta, but with additional features.

## API keys

Using an API key allows access to data without having to log in. If you do not already have an API key, you may create one by logging in to the BETYdb Web site, selecting Users from the Data menu, and clicking the "key" button in the Action column of the row containing your name. (Unless you are an adminstrator, you will only see one row—the one containing your name.) A new key will then be displayed in the "Apikey" column. You may also use this same method to change your API key if your existing one should become compromised.

To use an API key, simply append `?key=<your key>` to the end of the URL. If the URL already contains a query string, append `&key=<your key>` instead.

Your API key is unique to you and will have the same data-access permissions as those associated with your account.

## For administrators

API keys are automatically generated for new users, but users who signed up on previous releases may not have an API key in the database. To create an API key for every existing user who doesn't already have one, run the command

```
rake bety:make_keys
```

in the Rails root directory. (You may need to prepend `bundle exec` to this command, especially if you are not using a Ruby version manager.)

This command will not change existing keys. If for some reason you *want* to change the keys of all existing users, you must first log in to the BETYdb database and update the users table, setting the `apikey` column value for every user to `null`. After this, run the `rake` command to give all users a new API key.

# Original API

Jump to examples to get you started.

## Simple API queries

The simpler of the available API calls provide ways to extract useful information from a single database table in various formats. CSV (comma-separated values) format is of particular interest, as it is a format that is easy to view and use in any spreadsheet software and in R. Other possible formats are JSON and XML.

In the discussion that follows, most URLs given as examples are clickable; the results of the various API calls represented by these URLs will then be viewable in a browser.

In most of these calls, we will write the URLs to request results in XML format rather than in CSV format. This is mainly for convenience: requests for XML format will usually display the result in the browser whereas requests for CSV format will download a CSV file. If, however, you want to see what the result would be in either of the other two formats, this is easy to do: After clicking the link and seeing the XML result in the browser, go into the browser's address box and manually edit the URL there, replacing the .xml extension with .json or .csv as desired.

### A simple example

Data can be requested as a CSV, XML, or JSON file, and data from previously published syntheses can be requested without logging in. In the simplest case, we can use a single key-value pair to filter the table, restricting the result to the set of rows having a given value in a given column. For example, to download all of the trait data for the switchgrass species Panicum virgatum, we must first look up the id number of the Panicum virgatum species. We can do this by querying the species table by issuing a call to this URL[1]

https://www.betydb.org/species.xml?scientificname=Panicum+virgatum

and then examining the "id" value of the one species in the result. Once we have the id, we can use it in a query to the traits table:

https://www.betydb.org/traits.xml?specie_id=938 [2]

The reason we must use this two-stage process is that the traits table doesn't contain the species name information directly.

***Caveat!*** *There is no guarantee that these id numbers will not change!* Although it is unlikely that the id number for Panicum virgatum will change between the time we query the species table to look it up and the time we use it to query the traits table, it entirely possible that if we run the query https://www.betydb.org/traits.xml?specie_id=938 a year from now, the result will be for an entirely different species! For this and other reasons, it is worthwhile learning how to do cross-table queries.

# Complex API Queries

The API allows more complex queries and faster programmatic access by including multiple search terms in the query string of the URL. This feature is (for example) used to access BETYdb via the Ropensci `traits` package.

## A brief summary of the syntax of URL-based queries:

- The portion of the URL directly following https://www.betydb.org/ determines the primary table being queried and the format of the query result. For example, https://www.betydb.org/traits.json would obtain results from the traits table in JSON format and https://www.betydb.org/yields.xml would get results from the yields table in XML format.

- As a special case, to query the SQL view `traits_and_yields_view`, use "search" in place of the table name. For example, https://www.betydb.org/search.json returns the rows of this view in JSON format.

- The URL https://www.betydb.org/traits.csv will download the entire traits table (or rather, that portion of the table which the user is permitted to access) in CSV format. To restrict the results, we use a *query string*, which is the portion of the URL after the question mark. The query string consists of one or more clauses of the form `<key>=<value>` separated by `&` characters. For example, to obtain only the portion of the traits table associated with the site having id 99, the species having id 938, and the citation having id 45, we could write https://www.betydb.org/traits.csv?site_id=99&specie_id=938&citation_id=45. Note that restrictions are cummulative: each clause further restricts the result of applying the previous restriction. In general, there is no way of taking the disjunction of multiple clauses—that is, there is not way of "OR-ing" the clauses together (but see the next item).

- In queries on `traits_and_yields_view`, in addition to being able to use column names as keys, you can also use the special key `search`.[3] This matches against multiple columns at once, and the query succeeds if each word in the search string matches any portion of one of the column values in the list of searched columns. Moreover, these matches are not case-sensitive. For full details, see the appendix.

- URLs can not contain spaces. If the value you are querying against contains a space, substitute `+` or `%20` in the URL. Thus, above we had to write the query URL as https://www.betydb.org/species.csv?scientificname=Panicum+virgatum. We couldn't have written it as https://www.betydb.org/species.csv?scientificname=Panicum virgatum.

## Cross-table queries

It is possible to write queries involving multiple tables, but compared with the full expressive power of SQL, this ability is somewhat limited. Nevertheless, most common queries of interest should be possible. Here is a short outline of the scope and syntax of such queries:

- In any multiple-table query, there is always a "primary" table. This corresponds to the name directly following the hostname portion of the URL. For example, in a query of the form https://www.betydb.org/yields.csv?specie_id=938&..., "yields" is the primary table of the query.

- Other tables involved in the query must be related to the primary table in one of the following three ways: (1) The primary table has a foreign key column that refers to the other table. For example, the `citation_id` column of the yields table refers to the citations table, so we can include the citations table in any query on the yields table.[4] (2) The other table has a foreign key referring to the primary table. For example, we could involve the cultivars table in any query where species is the primary table because the cultivars table has a `specie_id` column. (3) There is a join table connecting the primary table to the other table. For example, because the `citations_sites` table joins the citations table with the sites table, a query with `citations` as the primary table is allowed to include the sites table. Visit https://www.betydb.org/schemas to view a list of the tables (including join tables) of BETYdb. Click on a table name to see a list of its columns.

- For each "other" table involved in the query, there must be a clause in the query string of the URL of the form `include[]=other`. The "other" portion of this clause is either (1) the name of the other table, or (2) the singular form of the name of the other table. Which of these two to use depends on whether a row of the primary table may be associated with multiple rows of the other table (case 1) or whether a row of the primary table is always associated with at most one row of the associated table (case 2). An easy rule of thumb here is to use the singular form if the primary table has a foreign key column referring to the other table and to use the bona fide (actual) table name

5

otherwise.[5] Note that for the purposes of this syntactical requirement (and contrary to reality), "specie" is the singular form of "species". Note you *must* include the square brackets after "include"!

- "include[]-ing" another table in a query allows the query to filter query results based on column values in that other table. But to do so, the "key" portion of the filter clause must be of the form `<other table name>.<column name>`. (Note that keys referring to column values in the *primary* table may use, but do not require, the table name qualifier.)
- "include[]-ing" another table in a query has another effect in the case of XML and JSON-formatted results: The associated row(s) from the included table are included with each entity of the primary table the query returns. For example, consider the query https://www.betydb.org/cultivars.xml?species.genus=Salix&include[]=specie. Not only will this return all cultivars for any species with genus Salix, it will return detailed species information for each cultivar returned. Contrast this with the CSV-format query https://www.betydb.org/cultivars.csv?species.genus=Salix&include[]=specie. This returns the same cultivars as before, but we only get the information contained in the cultivars table. In particular, while we will get the value of `specie_id` for each returned cultivar, we won't get the corresponding species name.

A few points need to be made about how cross-table queries work.

1. Suppose the primary table is in a many-to-one relationship with the *other* table. Then if we include a clause of the form `<other table name>.<column name>=value`, a row of the primary table is selected precisely when `<other table name>.<column name>=value` for at least one associated row from the other table. For example, in the query https://www.betydb.org/citations.xml?author=Clifton-Brown&include[]=sites&sites.city=Lisbon, the citations returned will be all and only those where the author is Clifton-Brown and where at least one of the associated sites is in the city Lisbon.
2. As mentioned above, in the case of XML and JSON-formatted results, full information is given about associated rows from the other table(s). In the example just given, for each citation of Clifton-Brown associated with some site(s) in Lisbon, we will see full information about those sites in the results. Other sites associated with the same citation but not in Lisbon, however, will not be shown.
3. You can add an `include[]=other` clause without any corresponding filter clauses. This will not filter out any results but *will* add in information from the other table (except when the return format is CSV). For example, the query https://www.betydb.org/citations.xml?author=Clifton-Brown&include[]=sites shows the same citations as the query https://www.betydb.org/citations.xml?author=Clifton-Brown, but it will, in addition, show full information about all of the sites associated with each of the selected citations.

4. Even if the singular form of the table name must be used in the `include[]=other` clause, the *actual* table name must be used in the `<other table name>.<column name>=value` filtering clause.

Some examples will make the query syntax clearer. In all of these examples, you can interchange `.json` with `.xml` or `.csv` depending on the format that you want.

## Examples

1. Return all trait data for "Panicum virgatum" without having to know its species id number:

   https://www.betydb.org/traits.xml?include[]=specie&species.scientificname=Panicum+virgatum

   This will return exactly the same traits as the query in simple example above that used the specie_id number (but with added species information; if CSV format is used instead, *exactly* the same result files are returned).

2. Note that the result of the previous query contains much redundancy: full information about species *Panicum virgatum* is included with each trait returned. The following query will only show the Panicum virgatum species information once:

   https://www.betydb.org/species.xml?scientificname=Panicum+virgatum&include[]=traits

   This returns a single result from the species table—the row information for Panicum virgatum—but nested inside this result is a list of all trait rows associated with that species.

   In CSV mode, this option is not available to us. CSV format only shows information extracted from the primary table. In CSV mode, an "include[]=other_table" clause is useless unless you are going to use the included table to filter the returned results by also including a clause of the form "other_table.column_name=column_value".

3. Return all citations in JSON format (replace .json with .xml or .csv for those formats):

   https://www.betydb.org/citations.json

4. Return all yield data for the genus *Miscanthus*:

   https://www.betydb.org/yields.xml?include[]=specie&species.genus=Miscanthus

   If we're only interested in Miscanthus having species name "sinensis", we can do:

   https://www.betydb.org/yields.xml?include[]=specie&species.genus=Miscanthus&species.species=sinensis

As in the first example, this query will show the same species information multiple times—once for each yield.

Once again, we may remedy this by making *species* the primary table:

https://www.betydb.org/species.xml?genus=Miscanthus&include[]=yields
https://www.betydb.org/species.xml?genus=Miscanthus&species=sinensis&include[]=yields

These each return a list of Miscanthus species where each species item itself contains a list of yields for that species. Since species and yields are in a one-to-many relationship, each species is only listed once.

5. Return all yield data associated with the citation Heaton 2008:

   https://www.betydb.org/citations.xml?include[]=yields&author=Heaton&year=2008

   To get this information in CSV format, we have to use yields as the primary table:

   https://www.betydb.org/yields.csv?include[]=citation&citations.author=Heaton&citations.year=2008

   Note that we use the singular form "citation" in the include clause (since there is one citation per yield), but the qualified column names "citations.author" and "citations.year" must always use the actual database table name.

6. Find species associated with the *salix* plant functional type:

   https://www.betydb.org/pfts.xml?include[]=specie&pfts.name=salix

   You can also do the query like so:

   https://www.betydb.org/species.xml?include[]=pfts&pfts.name=salix

   The results of these two queries contain essentially the same information. And to emphasize that we only need to interchange the primary and included table names to get from one query to the other, we have used the fully-qualified column name "pfts.name" in both (though just "name" would suffice in the first).

   The form of the results of the two queries is rather different. It is instructive to compare them. The first form is better for use with XML and JSON. It doesn't repeat the same PFT information over and over again inside each species listing. But the second form is the only one we can use with CSV format if we want to get any species information at all.

7. Return all citations with their associated sites:

   https://www.betydb.org/citations.xml?include[]=sites

(Note that we use the plural form "sites" in the include clause since citations and sites are in a many to many relationship.)

8. Return all citations with their associated sites and yields:

   https://www.betydb.org/citations.xml?include[]=sites&include[]=yields

   *Note that this will take considerable time since the information for all yields rows will be displayed.* Note also that the result of this query will not tell us directly which site is associated with which yield (although the yield result will have a site_id column, which will tell us indirectly).

9. Return all citations by author Lewandowski in the journal *European Journal of Agronomy* along with their associated sites and yields.

   https://www.betydb.org/citations.xml?journal=European+Journal+of+Agronomy&author=Lewandowski&include[]=sites&include[]=yields

10. Return citation 1 in json format:

    https://www.betydb.org/citations/1.json

    Alternatively, we can do:

    https://www.betydb.org/citations.json?id=1

    This second result is slightly different: It returns a JSON array with a single item instead of the item itself.

11. Return citation 1 in json format with it’s associated sites

    https://www.betydb.org/citations/1.json?include[]=sites

12. Return citation 1 in json format with it’s associated sites and yields

    https://www.betydb.org/citations/1.json?include[]=sites&include[]=yields

Regarding the last three examples, although the syntax for returning data associated with a single entity (citation 1 in these examples) is very convenient, in general it is best not to rely on fixed id numbers for entities. "The citation in *Agronomy Journal* with author Adler" is a more reliable locator than "The citation with id 1".

## Avoiding Cross-Table Queries: Using the Traits and Yields View

The database view `traits_and_yields_view` provides summary information about traits and yields: It combines information from the traits and the yields tables, and it includes information from six associated tables—namely, the *sites*, *species*, *citations*, *treatments*, *variables*, and *users* tables. Whereas the traits and the yields tables contain only pointers to these tables (foreign keys in the form of id numbers), the `traits_and_yields_view` directly contains information associated with a trait or yield such as the name of the species of the plant whose trait was measured, the name of the measured variable, and the author and year of the citation associated with the trait or yield. This means that in many cases, a query of the `traits_and_yields_view` may be used to extract information that otherwise would require a complex cross-table query with traits or yields as the primary table.

Moreover, by using the special `search=` key in place of column names, one can avoid the strict matching that is used when column names are used as keys. For example,

https://www.betydb.org/search.xml?search=cottongrass

will return all traits and yields for which the common name of the species includes the word "cottongrass". To get the same rows using `commonname` as the key, we would have to do three separate searches:

https://www.betydb.org/search.xml?commonname=tussock+cottongrass

https://www.betydb.org/search.xml?commonname=white+cottongrass

https://www.betydb.org/search.xml?commonname=tall+cottongrass

This is because when we use a column name as the search key, the value must match the column value exactly (including the letter case and the number of spaces between words).[6]

## Including Unchecked Data in the Traits and Yields View

By default, searches on `traits_and_yields_view` return only checked data. The Web interface now has a checkbox labelled "include unchecked records" that the user may check to override this default. To do the same thing through the API, the user must include a clause of the form `include_unchecked=true` in the query string. (Any of "TRUE", "yes", "YES", "y", "Y", "1", "t", "T" may be used in place of "true" here.)

## A Few Notes about Searching the Traits and Yields View

1. The base URL to use is https://www.betydb.org/search

2. You can use *column name* keys only when the result format is XML or JSON. If the result format is CSV or HTML, any query-string clause of the form `<columnname>=<value>` is ignored.

3. The query-string clauses of the form `search=<value>` and `include_unchecked=true` may be used with all result formats.

4. If multiple `search=<value>` clauses are included, only last one is used. **Don't do this!** To search on multiple terms, separate them in the query string by `+` signs. For example, https://www.betydb.org/search.xml?search=LTER+cottongrass&include_unchecked=true will likely find traits and yields results for cottongrass species at LTER (Long-term Ecological Research) sites, including those that have not been checked.

5. As noted in the Web Interface Search section, each word in the value of a `search=<value>` clause must match one of the columns `sitename`, `city`, `scientificname`, `commonname`, `cultivar`, `author`, `trait`, `trait_description`, `citation_year`, or `entity` in order for a given row to be included in the search results. Recall that a word in the value of the search string *matches* a column value if it is textually included in it.

6. Remember that values for column name keys are treated very differently from the way the value of the `search` key is treated. In particular, the value is treated as one unit. Consider the difference between these two examples: In the search

   https://www.betydb.org/search.xml?commonname=white+willow

   the search string "white willow" is treated as a unit, and this search will return only rows pertaining to the species with commonname "white willow". But in the search

   https://www.betydb.org/search.xml?search=white+willow

   the words "white" and "willow" are considered separately. So this search will in addition return (for example) rows for white ash at a site called Willow Creek (US-WCr) because "white" matches the species name and "willow" matches the site name.

---

> 1. In all of the examples in this section, we have written the query URLs as though they are to be run in a web browser after having logged in to the BETYdb web site. To run these queries from the command line or in a script, you must either pass login creditials with the HTTP call or include a valid API key in the URL (see the section on API keys). ↩

Using the command-line program `curl` as an example, we could, instead of visiting https://www.betydb.org/species.xml?scientificname=Panicum+virgatum in a Web browser, issue the `curl` command

```
curl -u <your login>:<your password> "https://www.betydb.org/species.xml?scientificnam
e=Panicum+virgatum"
```

or, using an API key instead of the `-u` option,

```
curl "https://www.betydb.org/species.xml?scientificname=Panicum+virgatum?key=yourAPIke
y"
```

> 2. Note the name of the key `specie_id` (which should rightfully be called `species_id`). This naming is an unfortunate artifact of Rails default schema for naming foreign keys. ↩

> 3. When searching in HTML format (the normal format when using the search page in the browser) column name parameters are ignored; only the value of the "search" parameters is significant. For a full list of which columns can be search in which contexts, and of all the search parameters that may be used, consult the appendix. ↩

> 4. In reality, it is not `citation_id`'s being a foreign key referencing the `id` column of the `citations` table which is the determining factor here. Rather, it is a Rails construct that determines whether one table is associated with another. In this example, it is a line in the Yield model—`belongs_to :citation`—that connects the citations table with the yields table. Similarly, it is the fact that the `Specie` model has the `has_many :cultivars` that allows us to bring in columns from the cultivars table when querying the species table. In general, the Rails relations correspond to the SQL ones, but this isn't always the case. For a full listing of Rails associations, see the appendix. ↩

> 5. The "real" rule is: use the singular form if the Rails model for the table has a line of the form `belongs_to <singular form of other table>`, and use the actual table name if the Rails model has a line of the form `has_many <other table>`. See the preceding footnote. ↩

> 6. The newer beta api offers less strict methods of matching by column values. ↩

# The Beta API

This page provides a general description of how to access data via the (new) beta version of the BETYdb API. For a list of URLs of API endpoints, visit https://www.betydb.org/api/docs.

# API endpoints for table listings

The paths for these endpoints are generally of the form `/api/beta/<table name>[.<extension>]`. Most tables of interest are exposed. The extension determines the format of the returned result and may be either "json" or "xml". The extension is optional, and if not given, the result is returned in JSON format.

## Format of table listing requests

A GET request may be sent to any of the paths listed on the `/api/docs` page (see above). (The path should be preceded by the HTTP protocol (http or https) and the hostname of course.) The only *required* parameter is *key* which is the user's API key.[1] (To view your API key, log in to the BETYdb Web interface, go to the users list page, and look in the *Apikey* column.)

#### Example

To get all[2] of the trait information from the traits table on the betydb.org site using `curl`, do

```
curl "https://www.betydb.org/api/beta/traits?key=<your api key>"
```

In examples from now on, we will put simply `key=` in the query string instead of `key=<your api key>`. This way, if you are logged in to www.betydb.org, the example can be run simply by pasting the URL shown into your browser's address box.[1]

## Implicit restriction of results

### Limit and Offset

Be default, the size of returned results is limited to 200 rows. To get around this limit, or to impose a smaller limit, use the `limit` parameter.

#### Example

To get a list of *all* citations, do

```
curl "https://www.betydb.org/api/beta/citations?key=&limit=none"
```

The `limit` parameter may be combined with an `offset` parameter to "shift the window" of returned results.

#### Example

The following three requests should[3] return three distinct groups of results of 10 rows each:

```
curl "https://www.betydb.org/api/beta/species?key=&limit=10"
curl "https://www.betydb.org/api/beta/species?key=&limit=10&offset=10"
curl "https://www.betydb.org/api/beta/species?key=&limit=10&offset=20"
```

### Restricted data

Unless the account associated with the specified API key has administrative access, certain rows are filtered out of the traits, yields, and users tables. In particular, non-administrators visiting the path `/api/beta/users` will only see the single row corresponding to their own account.

## Filtering results with exact matching

In order to filter the returned results, parameters specifying required column values can be added to the query string.

#### Example

To find all species in the family Orchidaceae, add `Family=Orchidaceae` to the query string:

```
curl "https://www.betydb.org/api/beta/species?key=&Family=Orchidaceae&limit=none"
```

Exact matching may be used on any column of the table being queried. It works best with integer and textual data types.

### Exact Matching Caveats

1. Textual matchings are case- and whitespace-sensitive.

   **Example**

   To find full information about sweet orange, you could do the API call

```
http://bety-dev:3000/api/beta/species?key=&limit=250&scientificname=Citrus+×+sinen
sis
```

(Note here that the spaces in the name must be URL-encoded as a `+` symbol when used in the query string.) But if you do

```
http://bety-dev:3000/api/beta/species?key=&limit=250&scientificname=Citrus+×sinens
is
```

instead (with no space between the `×` and the species name), you won't get the result you want.

2. Certain symbols that look very much alike are not interchangeable.

   In the previous example, if you had used the letter "x" instead of the times symbol "×", the API call wouldn't have returned the desired result. (You could have, however, used "%C3%97" in place of "×": "%C3%97" is the URL encoding of "×".)

3. Date/Time may not match as expected.

   To match a column value having data type "timestamp", you must exactly match the PostgreSQL textual representation of that timestamp. In general, this is not the same as the representation displayed by the query result. For example, a query that filters by creation time using the key-value string `created_at=2010-10-22+13:59:26` may return results where the "created_at" value is `2010-10-22T08:59:26-05:00`. This is because the "created_at" column has type "timestamp without time zone" and because Rails stores timestamps in UTC but displays them in local time showing the timezone offset. Moreover, whereas SQL separates the date and time parts of a timestamp with a space, Rails displays a "T" between the two parts.

4. Numerical types match as numbers.

   For integers, this means that could (somewhat perversely) match an id number of 26536 using the key-value pair `id=+++%2B26536+++` in the query string. Here, "%2B" is the URL-encoded version of the plus sign, and the "+" symbols are URL-encodings of spaces.

   For floating point numbers, some rounding is done when matching values, but not much. For example, the key-value string "stat=0.18299999999999999" may return a trait having stat value 0.183 whereas the string "stat=0.1829999999999" may not.

## Matching using regular expressions

To turn on regular expression matching, use the symbol "~" as the first symbol of the value in any key-value pair.

**Example**

To find all sites with the string "University" in their sitename, do the API call

```
curl "https://www.betydb.org/api/beta/sites?sitename=~University&key="
```

Note that regular expressions are not anchored by default. To return only those sites whose sitename begins with "University", use the regular-expression anchor symbol "^" at the beginning of the string:

```
curl "https://www.betydb.org/api/beta/sites?sitename=~%5EUniversity&key="
```

(Here, "%5E" is the URL-encoding of "^".)

We can use regular expression matching to alleviate some of the problems with exact matching mentioned above. For example, to find all trait rows that were updated in the middle part of March 2015, we could do an API call like

```
curl "https://www.betydb.org/api/beta/traits?key=&limit=250&updated_at=~2015-03-1
```

This will find all traits updated from March 10 through March 19, 2015 UTC.

## Format of Returned Results

The beta API suports two response formats: JSON and XML. This is a very brief description of the information contained in the response and how that information is presented in each format.

## General outline of information returned.

Every response is divided into two or more of the following main sections:

- metadata

  The metadata in the response contains the following three items:

  - URI: The URI requested in the GET request.
  - timestamp: The date and time of the request.

    - count: The number of rows of data represented in the response or `null` if there was an error in the request.

- errors

  The error or list of errors associated with the request. If no errors occurred, the item will be omitted.

- warnings

  Any warnings associated with the request. This item is omitted if there are no warnings.

- data

  This is main item of interest in a successful request. (It is omitted if the request is unsuccessful.) It contains all the rows of the table associated with the URI path that are not filtered out by request parameters (subject to access restrictions and the default size limit).

## Content of returned data: JSON format

The value of the "data" property will be a JSON array consisting of zero or more JSON objects, each of which represents one row of the database table corresponding to the API endpoint in the request. Each of these objects has a single property whose name is usually the singular form of the table being represented. For example, in a request to the API path `/api/beta/traits` , the property name will be "trait". The value of this property is itself a JSON object, one having a property for each column of the represented database table. The data portion of a typical trait request will look something like this:

```
"data": [
    {
        "trait": {
            "id": 36854,
            "site_id": 593,
            "specie_id": 19066,
            "citation_id": 375,
            "cultivar_id": null,
            "treatment_id": 1199,
            "date": "2002-07-21T19:00:00-05:00",
            "dateloc": "5.0",
            ...
            ...
        },
        "trait": {
            "id": 22320,
            "site_id": 76,
            "specie_id": 938,
            "citation_id": 18,
            "cultivar_id": 10,
            "treatment_id": 743,
            "date": "2006-04-25T19:00:00-05:00",
            "dateloc": "5.5",
            ...
            ...
        },
        ...
        ...
    }
]
```

### Additional properties

In addition to a property for each column of the represented database table, the data JSON objects will also contain the following:

- A property for certain associated tables that are in a many-to-many or many-to-one relationship with this table.

  For example, the value of each *site* property returned in a request to path `/api/beta/sites` will be an object containing the properties "number of associated citations", "number of associated sitegroups", "number of associated yields", etc.

  Note that properties corresponding to join tables are generally omitted. For example, a "site" object will have a property "number of associated citations" but not "number of associated citations_sites".

- A `view_url` property.

This property shows the "show" URL for the Web page that displays the item in HTML format. This page will usually contain an "Edit" link for editing the item.

**Representation of column values**

For the property values corresponding to table columns, there are at least two cases where the form in which they are represented in the JSON response is slightly different from the form returned by a direct query of the database using a client such as psql.

1. timestamps

   The form returned by the API is `YYYY-MM-DDTHH:MM:SS[+|-]HH:MM` whereas psql will display a timestamp in the form `YYYY-MM-DD HH:MM:SS.nnnnnn`. That is, the API (a) separates the date and the time with the character "T" where psql uses a space, (b) shows a timezone offset where psql simply returns a time with no offset (assumed to be UTC), and (c) does not show fractional seconds.

2. geometries

   The API displays the sites.geometry column in human-readable form (for example, "POINT (-68.1069 44.8708 265.0)") whereas psql does not.

## Content of returned data: XML format

The information content of the data in an XML response is exactly the same as that of a JSON response. In general, where JSON has a property name and a property value, XML will have an XML element with a corresponding name and content corresponding to the value. But note the following:

1. The root element of the XML response is named "hash" (this may change).
2. For JSON properties having non-textual simple values, the corresponding XML element will have a "type" attribute stating the type of the value.

   For example, the start tag for the element representing the "id" column is `<id type="integer">`. The start tag for the "created_at" column is `<created-at type="datetime">`. Note that the types shown are Ruby types, not SQL types.

3. For JSON properties whose values are JSON arrays, the corresponding XML element will have an attribute `type="array"`.
4. Underscores in property names become hyphens in XML element names.
5. Both NULLs and empty string values in the database are represented in XML by empty elements having the attribute `nil="true"`.[4]

6. The XML response has an element whose name matches the database table name.

   For example, a request to the path `/api/beta/sites.xml` returns a result whose data portion looks like

   ```
   <data>
    <sites type="array">
      <site>
        <id type="integer">...
        ...
        ...
      </site>
      <site>
        <id type="integer">...
        ...
        ...
      </site>
      ...
      ...
      ...
    </sites>
   </data>
   ```

## API endpoints for individual items

The paths for these endpoints are generally of the form

```
/api/beta/<table name>/<id number>[.<extension>]
```

Again, most tables of interest are exposed, and the extension (if given) determines the format of the returned result (JSON by default).

### Parameters for individual-item API endpoints

The only required parameter—indeed, the only *recognized* parameter—is the "key" parameter specifying the user's API key.

### Format of Returned Results

In general, a request of the form

```
curl "http[s]://<hostname>/api/beta/<table name>/nnn.json?key=<your api key>
```

will return nearly the same result as the request

```
curl "http[s]://<hostname>/api/beta/<table name>.json?key=<your api key>&id=nnn
```

Here are the main differences:

1. There will be no *count* property in the metadata. If the request is successful, the response will contain data about exactly one item.
2. The value of the "data" property will be a single JSON object rather than a JSON array.
3. If the table being queried is in a many-to-one relationship to some other table, full information will be included about the referred-to item. For example, https://www.betydb.org/api/beta/cultivars.json?key=&id=55 will show the id of the species associated with cultivar 55, but https://www.betydb.org/api/beta/cultivars/55.json?key= will show full information about the species.
4. For one-to-many or many-to-many associations, the *individual item* form of the request will return a list of associated items, not just a count of the number of associated items.

These differences carry over *mutatis mutandis* to XML-format requests.

---

> 1. If you enter these API URLs in a browser after logging into the host site, you don't actually need to use a valid API key. The `key` parameter must still be present, but you don't need to use a corresponding value. Thus, to see a list of traits, you could just enter https://www.betydb.org/api/beta/traits?key= into your browser's address box after logging in to www.betydb.org. This will save much typing as you try out examples. ↩

> 2. Actually, this retrieves only the first 200 rows, 200 being the default size limit of the response. ↩

To get more than 200 results, set an explicit limit using the `limit` parameter; for example,

```
curl "https://www.betydb.org/api/beta/traits?key=&limit=550"
```

Note that *it is especially important to quote the URL here* as otherwise the `&` symbol will be interpreted by the shell instead of as part of the URL.

If you want to ensure that *all* rows are returned, use the special limit value "all" (or equivalently, "none", that is, *no limit*):

```
curl "https://www.betydb.org/api/beta/traits?key=&limit=all"
```

> 3. Use of offset may not give consistent results. To quote from the PostgreSQL documentation: ↩

> The query optimizer takes LIMIT into account when generating query plans, so you are very likely to get different plans (yielding different row orders) depending on what you give for LIMIT and OFFSET. Thus, using different LIMIT/OFFSET values to select different subsets of a query result will give inconsistent results unless you enforce a predictable result ordering with ORDER BY.

The beta API does not (yet) support specifying the order of results.

> 4. This is one case in which the XML response contains less information that the JSON response. Whereas JSON displays `""` and `null` respectively for empty strings and NULLs, XML shows `nil="true"` in both cases. ↩

# rOpenSci traits package

The rOpenSci traits package is an R client for various sources of species trait data. The traits package provides functions that interface with the BETYdb API.

These instructions are from the traits package documentation, which is released with a MIT-BSD license.

## Installation

Launch R

```
$ R
```

Install the stable version of the package from CRAN:

```
install.packages("traits")
```

Or development version from GitHub:

```
devtools::install_github("ropensci/traits")
```

## Load traits

```
library("traits")
```

## Example 1: Query trait data for Willow

Get trait data for Willow (*Salix* spp.)

```
(salix <- betydb_search("Salix Vcmax"))
#> Source: local data frame [14 x 31]
#> 
#>    access_level       author checked citation_id citation_year  city
#>           (int)        (chr)   (int)       (int)         (int) (chr)
#> 1             4       Merilo       1         430          2005 Saare
#> 2             4       Merilo       1         430          2005 Saare
#> 3             4       Merilo       1         430          2005 Saare
#> 4             4       Merilo       1         430          2005 Saare
#> 5             4 Wullschleger       1          51          1993    NA
#> 6             4       Merilo       1         430          2005 Saare
#> 7             4       Merilo       1         430          2005 Saare
#> 8             4       Merilo       1         430          2005 Saare
#> 9             4       Merilo       1         430          2005 Saare
#> 10            4       Merilo       1         430          2005 Saare
#> 11            4       Merilo       1         430          2005 Saare
#> 12            4       Merilo       1         430          2005 Saare
#> 13            4       Merilo       1         430          2005 Saare
#> 14            4         Wang       1         381          2010    NA
#> Variables not shown: commonname (chr), cultivar_id (int), date (chr),
#>   dateloc (chr), genus (chr), id (int), lat (dbl), lon (dbl), mean (chr),
#>   month (dbl), n (int), notes (chr), result_type (chr), scientificname
#>   (chr), site_id (int), sitename (chr), species_id (int), stat (chr),
#>   statname (chr), trait (chr), trait_description (chr), treatment (chr),
#>   treatment_id (int), units (chr), year (dbl)
# equivalent:
# (out <- betydb_search("willow"))
```

### Summarise data from the output `data.frame`

```
library("dplyr")
salix %>%
  group_by(scientificname, trait) %>%
  mutate(.mean = as.numeric(mean)) %>%
  summarise(mean = round(mean(.mean, na.rm = TRUE), 2),
            min = round(min(.mean, na.rm = TRUE), 2),
            max = round(max(.mean, na.rm = TRUE), 2),
            n = length(n))
#> Source: local data frame [4 x 6]
#> Groups: scientificname [?]
#> 
#>                    scientificname trait  mean   min   max     n
#>                             (chr) (chr) (dbl) (dbl) (dbl) (int)
#> 1                           Salix Vcmax 65.00 65.00 65.00     1
#> 2                 Salix dasyclados Vcmax 46.08 34.30 56.68     4
#> 3 Salix sachalinensis × miyabeana Vcmax 79.28 79.28 79.28     1
#> 4                  Salix viminalis Vcmax 43.04 19.99 61.29     8
```

BETYdb is the *Biofuel Ecophysiological Traits and Yields Database*. You can get many different types of data from this database, including trait data.

Function setup: All functions are prefixed with `betydb_`. Plural function names like `betydb_traits()` accept parameters and always give back a data.frame, while singular function names like `betydb_trait()` accept an ID and give back a list.

The idea with the functions with plural names is to search for either traits, species, etc., and with the singular function names to get data by one or more IDs.

## Example 2: Get yield data for Switchgrass (*Panicum virgatum*)

```
out <- betydb_search(query = "Switchgrass Yield")
```

Summarise data from the output `data.frame`

```
library("dplyr")
out %>%
  group_by(id) %>%
  summarise(mean_result = mean(as.numeric(mean), na.rm = TRUE)) %>%
  arrange(desc(mean_result))
```

```
## Source: local data frame [509 x 2]
##
##         id mean_result
## 1    1666       27.36
## 2   16845       27.00
## 3    1669       26.36
## 4   16518       26.00
## 5    1663       25.35
## 6   16742       25.00
## 7    1594       24.78
## 8    1674       22.71
## 9    1606       22.54
## 10   1665       22.46
## ..    ...         ...
```

## Advanced Queries

The tables above will return values of _tablename_id that can be used to query other tables

## Query a Single trait record by its id

```
betydb_trait(id = 10)
```

```
## $created_at
## NULL
##
## $description
## [1] "Leaf Percent Nitrogen"
##
## $id
## [1] 10
##
## $label
## NULL
##
## $max
## [1] "10"
##
## $min
## [1] "0.02"
##
## $name
## [1] "leafN"
##
## $notes
## [1] ""
##
## $standard_name
## NULL
##
## $standard_units
## NULL
##
## $units
## [1] "percent"
##
## $updated_at
## [1] "2011-06-06T09:40:42-05:00"
```

## Query a single Species

```
betydb_specie(id = 10)
```

```
## $AcceptedSymbol
## [1] "ACKA2"
##
## $commonname
## [1] "karroothorn"
##
## $created_at
## NULL
##
## $genus
## [1] "Acacia"
##
## $id
## [1] 10
##
## $notes
## [1] ""
##
## $scientificname
## [1] "Acacia karroo"
##
## $spcd
## NULL
##
## $species
## [1] "karroo"
##
## $updated_at
## [1] "2011-03-01T15:02:25-06:00"
```

## Query a single Citation

```
betydb_citation(10)
```

```
## $author
## [1] "Casler"
##
## $created_at
## NULL
##
## $doi
## [1] "10.2135/cropsci2003.2226"
##
## $id
## [1] 10
##
## $journal
## [1] "Crop Science"
##
## $pdf
## [1] "http://crop.scijournals.org/cgi/reprint/43/6/2226.pdf"
##
## $pg
## [1] "2226–2233"
##
## $title
## [1] "Cultivar X environment interactions in switchgrass"
##
## $updated_at
## NULL
##
## $url
## [1] "http://crop.scijournals.org/cgi/content/abstract/43/6/2226"
##
## $user_id
## NULL
##
## $vol
## [1] 43
##
## $year
## [1] 2003
```

## Query a single Site

```
betydb_site(id = 1)
```

```
## $city
## [1] "Aliartos"
##
## $country
## [1] "GR"
##
## $geometry
## [1] "POINT (23.17 38.37 114.0)"
##
## $greenhouse
## [1] FALSE
##
## $notes
## [1] ""
##
## $sitename
## [1] "Aliartos"
##
## $state
## [1] ""
```

# R dplyr Package

## R dplyr interface

Using dplyr requires having access to a PostgreSQL server running BETYdb or installing your own.

Comprehensive documentation for the `dplyr` interface to databases is provided in the dplyr vignette

# Query Miscanthus yield data

```
install.packages('dplyr')
library(dplyr)
d <- settings$database$bety[c("dbname", "password", "host", "user")]
bety <- src_postgres(host = d$host, user = d$user, password = d$password, dbname = d$d
bname)

species <- tbl(bety, 'species') %>%
  select(id, scientificname, genus) %>%
  filter(genus == "Miscanthus") %>%
  mutate(specie_id = id)

yields <-tbl(bety, 'yields') %>%
  select(date, mean, site_id, specie_id)

sites <- tbl(bety, 'sites') %>%
  select(id, sitename, city, country) %>%
  mutate(site_id = id)

mxgdata <- inner_join(species, yields, by = 'specie_id') %>%
  left_join(sites, by = 'site_id') %>%
  select(-ends_with(".x"), -ends_with(".y")) %>% # drops duplicate rows
  collect()
```

# Yield data with experimental treatments

Here we query Miscanthus and Switchgrass yield data along with planting, irrigation, and fertilization rates in order to update teh meta-analysis originally performed by Heaton et al (2004).

```
library(dplyr)
library(data.table)
## connection to database
d <- list(host = 'localhost',
          dbname = 'bety',
          user = 'bety',
          password = 'bety')

bety <- src_postgres(host = d$host, user = d$user, password = d$password, dbname = d$d
bname)

## query and join tables
species <- tbl(bety, 'species') %>%
  select(id, scientificname, genus) %>%
  rename(specie_id = id)

sites <- tbl(bety, sql(
  paste("select id as site_id, st_y(st_centroid(sites.geometry)) AS lat,",
        "st_x(st_centroid(sites.geometry)) AS lon,",
        " sitename, city, country from sites"))
  )

citations <- tbl(bety, 'citations') %>%
  select(citation_id = id, author, year, title)

yields <- tbl(bety, 'yields') %>%
  select(id, date, mean, n, statname, stat, site_id, specie_id, treatment_id, citation
_id, cultivar_id) %>%
  left_join(species, by = 'specie_id') %>%
  left_join(sites, by = 'site_id') %>%
  left_join(citations, by = 'citation_id')

managements_treatments <- tbl(bety, 'managements_treatments') %>%
  select(treatment_id, management_id)

treatments <- tbl(bety, 'treatments') %>%
  dplyr::mutate(treatment_id = id) %>%
  dplyr::select(treatment_id, name, definition, control)

managements <- tbl(bety, 'managements') %>%
  filter(mgmttype %in% c('fertilizer_N', 'fertilizer_N_rate', 'planting', 'irrigation'
)) %>%
  dplyr::mutate(management_id = id) %>%
  dplyr::select(management_id, date, mgmttype, level, units) %>%
  left_join(managements_treatments, by = 'management_id') %>%
  left_join(treatments, by = 'treatment_id')

nitrogen <- managements %>%
  filter(mgmttype == "fertilizer_N_rate") %>%
  select(treatment_id, nrate = level)

planting <- managements %>% filter(mgmttype == "planting") %>%
  select(treatment_id, planting_date = date)
```

```r
planting_rate <- managements %>% filter(mgmttype == "planting") %>%
  select(treatment_id, planting_date = date, planting_density = level)

irrigation <- managements %>%
  filter(mgmttype == 'irrigation')

irrigation_rate <- irrigation %>%
  filter(units == 'mm', !is.na(treatment_id)) %>%
  group_by(treatment_id, year = sql("extract(year from date)"), units) %>%
  summarise(irrig.mm = sum(level)) %>%
  group_by(treatment_id) %>%
  summarise(irrig.mm.y = mean(irrig.mm))

irrigation_boolean <- irrigation %>%
  collect %>%
  group_by(treatment_id) %>%
  mutate(irrig = as.logical(mean(level))) %>%
  select(treatment_id, irrig = irrig)

irrigation_all <- irrigation_boolean %>%
  full_join(irrigation_rate, copy = TRUE, by = 'treatment_id')

grass_yields <- yields %>%
  filter(genus %in% c('Miscanthus', 'Panicum')) %>%
  left_join(nitrogen, by = 'treatment_id') %>%
  #left_join(planting, by = 'treatment_id') %>%
  left_join(planting_rate, by = 'treatment_id') %>%
  left_join(irrigation_all, by = 'treatment_id', copy = TRUE) %>%
  collect %>%
  mutate(age = year(date)- year(planting_date),
         nrate = ifelse(is.na(nrate), 0, nrate),
         SE = ifelse(statname == "SE", stat, ifelse(statname == 'SD', stat / sqrt(n),
NA)),
         continent = ifelse(lon < -30, 'united_states', ifelse(lon < 75, 'europe', 'as
ia')))
```

# PEcAn R package

## PEcAn.DB functions

Like using the R dplyr package, the PEcAn.DB package requires the user to have read access to a PostgreSQL server running BETYdb. Unlike the dplyr package, the PEcAn.DB package is less secure, more difficult to install. The primary reason to use this package is if you are using the PEcAn ecosystem modeling framework. The easiest way to install the BETYdb server and run these queries is to use the PEcAn virtual machine. This can be done on your computer or through Amazon Web Services.

Here is some of the basic functionality:

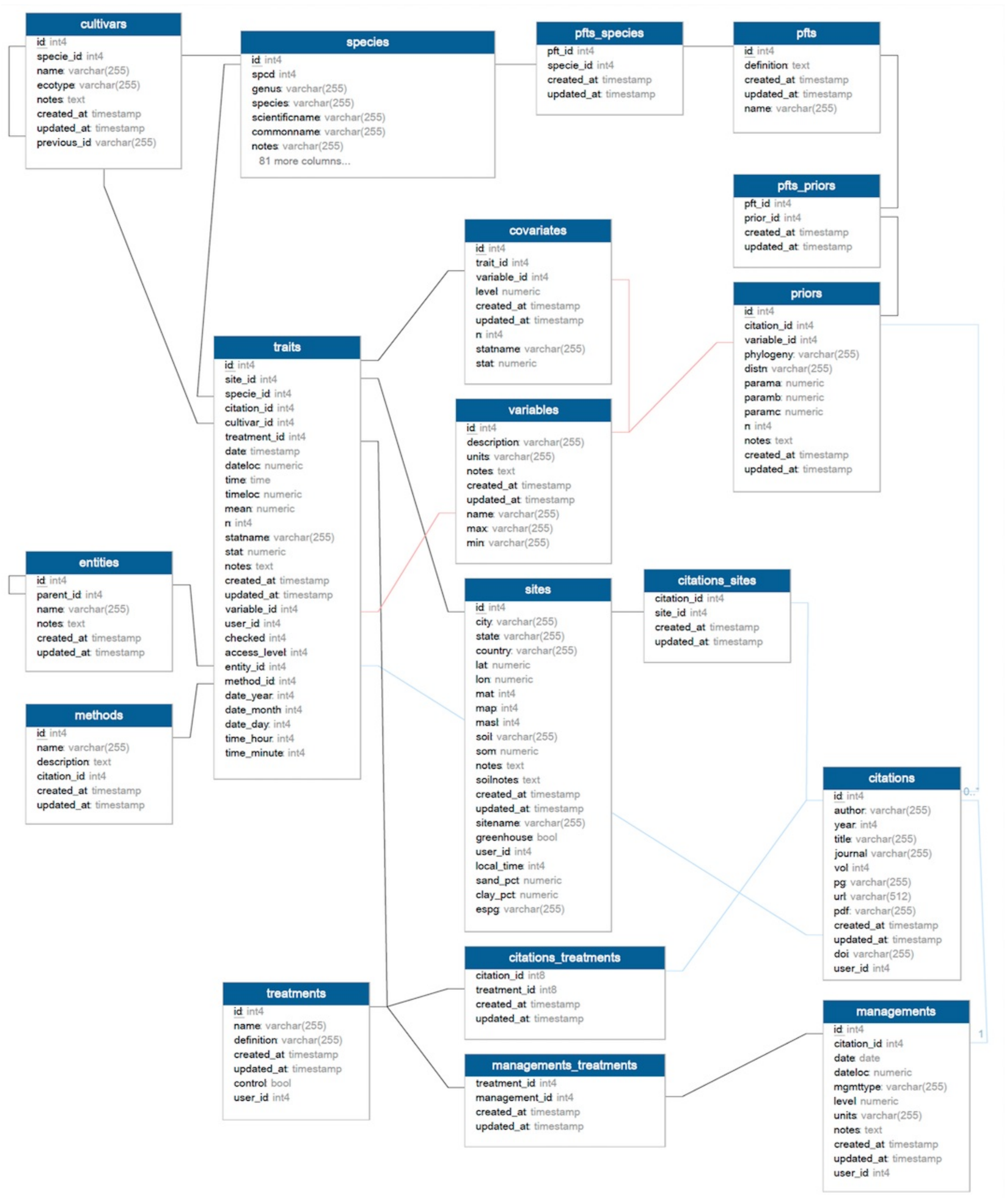
```
settings <- read.settings("pecan.xml")

# For database queries, this is equivalent to:
# settings <- list(database = list(bety = list(driver = "PostgreSQL", user = "bety", d
bname = "bety", password = "bety")))

library(PEcAn.DB)
require(RPostgreSQL)
dbcon <- db.open(settings$database$bety)

miscanthus <- db.query("select lat, lon, date, trait, units, mean from traits_and_yiel
ds_view where genus = 'Miscanthus';", con = dbcon)

salix_spp <- query.pft_species(pft = "salix", modeltype = "BIOCRO", con = dbcon)

salix_vcmax <- query.trait.data(trait = "Vcmax", spstr = vecpaste(salix_spp$id), con =
 dbcon)
```

# SQL queries

The BETYdb repository has a large number of example SQL queries if you search for the terms 'select' and 'join' in its issues.

SQL is the most flexible way of querying data from BETYdb. The full list of tables and schema on the website is up to date with the current database, but the tables below are stable.

These are the core tables required to query traits and yields:



We have created 'views' to make it easier to query data from multiple tables. For example, to lookup the name and location of a site or the names and units of variables. The following query joins multiple tables, and is based on the yieldsview table:

```
SELECT 'yields'::character(10) AS result_type,
    yields.id,
    yields.citation_id,
    yields.site_id,
    yields.treatment_id,
    sites.sitename,
    sites.city,
    st_y(st_centroid(sites.geometry)) AS lat,
    st_x(st_centroid(sites.geometry)) AS lon,
    species.scientificname,
    species.commonname,
    species.genus,
    species.id AS species_id,
    yields.cultivar_id,
    citations.author,
    citations.year AS citation_year,
    treatments.name AS treatment,
    yields.date,
    date_part('month'::text, yields.date) AS month,
    date_part('year'::text, yields.date) AS year,
    variables.name AS trait,
    variables.description AS trait_description,
    yields.mean,
    variables.units,
    yields.n,
    yields.statname,
    yields.stat,
    yields.notes,
   FROM ((((((yields
     LEFT JOIN sites ON ((yields.site_id = sites.id)))
     LEFT JOIN species ON ((yields.specie_id = species.id)))
     LEFT JOIN citations ON ((yields.citation_id = citations.id)))
     LEFT JOIN treatments ON ((yields.treatment_id = treatments.id)))
     LEFT JOIN variables ON (((variables.name)::text = 'Ayield'::text)))
     LEFT JOIN users ON ((yields.user_id = users.id)));
```

# Installing a local version of BETYdb

A standard ODB database provides access to the back-end PostgreSQL database server. This is hosted on a production server or locally.

The following scripts make it easy to install the latest version of BETYdb locally (on Linux or OSX), populated with all public data. Many users of PEcAn install a local copy of BETYdb, and BETYdb can sync across different servers. The PEcAn repository provides useful configuration files. Their use is described in the developer's guide to installing PEcAn. You can also run BETYdb on pre-configured PEcAn Virtual Machines using a VirtualBox or similar software, and can be run on a laptop, desktop, server, or cloud services such as Amazon cloud or NCSA OpenStack.

- PostgreSQL: https://github.com/PecanProject/pecan/blob/master/scripts/load.bety.sh
- BETYdb documentation for `update.bety.sh` :
  https://github.com/PecanProject/bety/wiki/Updating-BETY
- PEcAn Documentation for BETYdb installation
  https://github.com/PecanProject/pecan/wiki/Installing-PEcAn#installing-bety
- The Federated network of independently hosted instances of BETYdb
  https://github.com/PecanProject/bety/wiki/Distributed-BETYdb

# Appendix: Full Search Details

## List of searchable columns in each table

## Specification of result formats

### The CSV format

### The XML format

### The JSON format

### Regular expressions in API queries