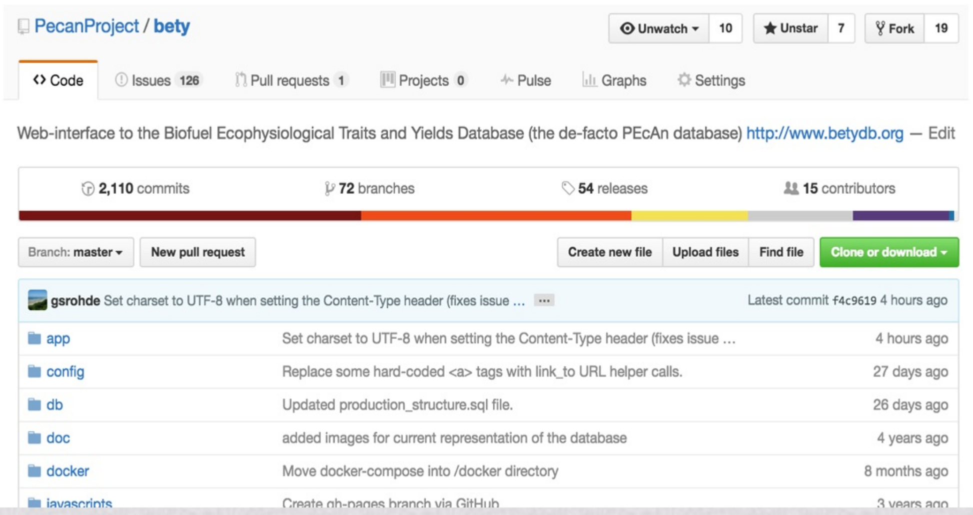
BETYdb Documentation: Volume 3: Technical Guide

Implementation and deployment of the BETYdb database and web applications.


PecanProject / bety

Unwatch 10 | Unstar 7 | Fork 19

Code | Issues 126 | Pull requests 1 | Projects 0 | Pulse | Graphs | Settings

Web-interface to the Biofuel Ecophysiological Traits and Yields Database (the de-facto PEcAn database) http://www.betydb.org — Edit

2,110 commits | 72 branches | 54 releases | 15 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

gsrohde Set charset to UTF-8 when setting the Content-Type header (fixes issue … Latest commit f4c9619 4 hours ago

app	Set charset to UTF-8 when setting the Content-Type header (fixes issue …	4 hours ago
config	Replace some hard-coded <a> tags with link_to URL helper calls.	27 days ago
db	Updated production_structure.sql file.	26 days ago
doc	added images for current representation of the database	4 years ago
docker	Move docker-compose into /docker directory	8 months ago
javascripts	Create gh-pages branch via GitHub	3 years ago

Energy Biosciences Institute

BETYdb | Biofuel Ecophysiological Traits and Yields Database

Table of Contents

About BETYdb

Database Description and User's Guide

This wiki describes the purpose, design, and use of the Biofuel Ecophysiological Traits and Yields database (BETYdb). BETYdb is a database of plant trait and yield data that supports research, forecasting, and decision making associated with the development and production of cellulosic biofuel crops. While the content of BETYdb is agronomic, the structure of the database itself is general and can therefore be used more generally for ecosystem studies.

Note that this document does not cover the suite of tables used by PEcAn. These are covered in the PEcAn documentation.

Objectives

A major motivation of the biofuel industry is to reduce greenhouse gas emissions by providing ecologically and economically sustainable sources of fuel thereby reducing dependence on fossil fuel. The goals of this database are to provide a clearinghouse of existing research on potential biofuel crops; to provide a source of data on plant ecophysiological traits and yields; and to present ecosystem-scale re-analysis and forecasts that can support the agronomic, ecological, policy, and economic aspects of the biofuel industry. This database will facilitate the scientific advances and assessments that the transition to biofuels will require.

The objectives of this database are to allow other users access to data that has been collected from previously published and ongoing research in a consistent format, and to provide a streamlined interface that allows users to enter their own data. These objectives will support specific research and collaboration, advance agricultural practices, and inform policy decisions. Specifically, BETYdb supports the following uses, allowing users to:

1. Carry out statistical analyses to explore the relationships between traits
2. Identify differences among species and functional groups
3. Access BETYdb from simulation models to look up values for traits and parameters
4. Identify gaps in knowledge about biofuel crop traits and model parameters to aid rational planning of research activities

BETYdb provides a central clearinghouse of biofuel crop physiological traits and yields in a consistently organized framework that simplifies the use of these data for further analysis and interpretation. Scientific applications include the development, assessment, and prediction of crop yields and ecosystem services in biofuel agroecosystems. The database directly supports parameterization and validation of ecological, agronomic, engineering, and economic models. The initial target end-users of BETYdb version 1.0 are users within EBI who aim to support sustainable biofuel production through statistical analysis and ecological modeling. By streamlining the process of data summary, we hope to inspire new scientific perspectives on biofuel crop ecology that are based on a comprehensive evaluation of available knowledge.

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. Data in tables having an an access_level column and in rows where the access_level value is 1 or 2 are not covered by this license but may be available for use with consent.

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

Scope

The database contains trait, yield, and ecosystem service data. Because all plants have the potential to be used as biofuel feedstock, BETYdb supports data from all plant species. In practice, the species included in the database reflect available data and the past and present research interests of contributors. Trait and yield data are provided at the level of species, with cultivar and clone information provided where available.

The yield data not only includes end-of-season harvestable yield, but also includes measurements made over the course of the growing season. These yield data are useful in the assessment of historically observed crop yields, and they can also be used in the validation of plant models. Yield data includes peak biomass, harvestable biomass, and the biomass of the crop throughout the growing season.

The trait data represent phenotypic traits; these are measurable characteristics of an organism. The primary objective of the trait data is to allow researchers to model second generation biofuel crops such as miscanthus and switchgrass. In addition, these data enable evaluation of new plant species as potential biofuel crops. Ecosystem service data reflect ecosystem-level observations, and these data are included in the traits table.

Content

BETYdb includes data obtained through extensive literature review of target species in addition to data collected from the Energy Farm at the University of Illinois, and by our collaborators. The BETYdb database contains trait and yield data for a wide range of plant species so that it is possible to estimate the distribution of plant traits for broad phylogenetic groups and plant functional types.

BETYdb contains data from intensive efforts to find data for specific species of interest as well as from previous plant trait and yield syntheses and other databases. Most of the data currently in the database is from plant genera that are the focus of our current and previous research. These species include perennial grasses, such as miscanthus (*Miscanthus sinensis*) switchgrass (*Panicum virgatum*), and sugarcane (*Saccharyn* spp.). BETYdb also includes short-rotation woody species, including poplar (*Populus* spp.) and willow (*Salix* spp.) and a group of species that are being evaluated at the energy farm as novel woody crops. In addition to these herbaceous species, we are collecting data from a species in an experimental low-input, high diversity prairie.

An annotated, interactive schema can be accessed on the website by selecting "Docs --> Schema".[1]

Design

BETYdb is a relational database that comprehensively documents available trait and yield data from diverse plant species (Figure 1). The underlying structure of BETYdb is designed to support meta-analysis and ecological modeling. A key feature is the PFT (plant functional type) table which allows a user to group species for analysis. On top of the database, we have created a web-portal that targets a larger range of end users, including scientists, agronomists, foresters, and those in the biofuel industry.

Data Entry

The Data Entry Workflow provides a complete description of the data entry process. BETYdb’s web interface has been developed to facilitate accurate and efficient data entry. This interface provides logical workflow to guide the user through comprehensively documenting data along with species, site information, and experimental methods. This workflow is outlined in the BETYdb Data Entry Workflow document. Data entry requires a login with `Create` permissions; this can be obtained by contacting David LeBauer.

Software

The BETYdb was originally developed in MySQL and later converted to PostgreSQL. It uses Ruby on Rails for its web portal and is hosted on a RedHat Linux Server (ebi-forecast.igb.illinois.edu). BETYdb is a relational database designed in a generic way to facilitate easy implementation of additional traits and parameters.

List of Tables in the BETY Database

An up-to-date list of the tables in BETYdb along with their descriptions and diagrams of their interrelationships may be found at https://www.betydb.org/schemas. [2]

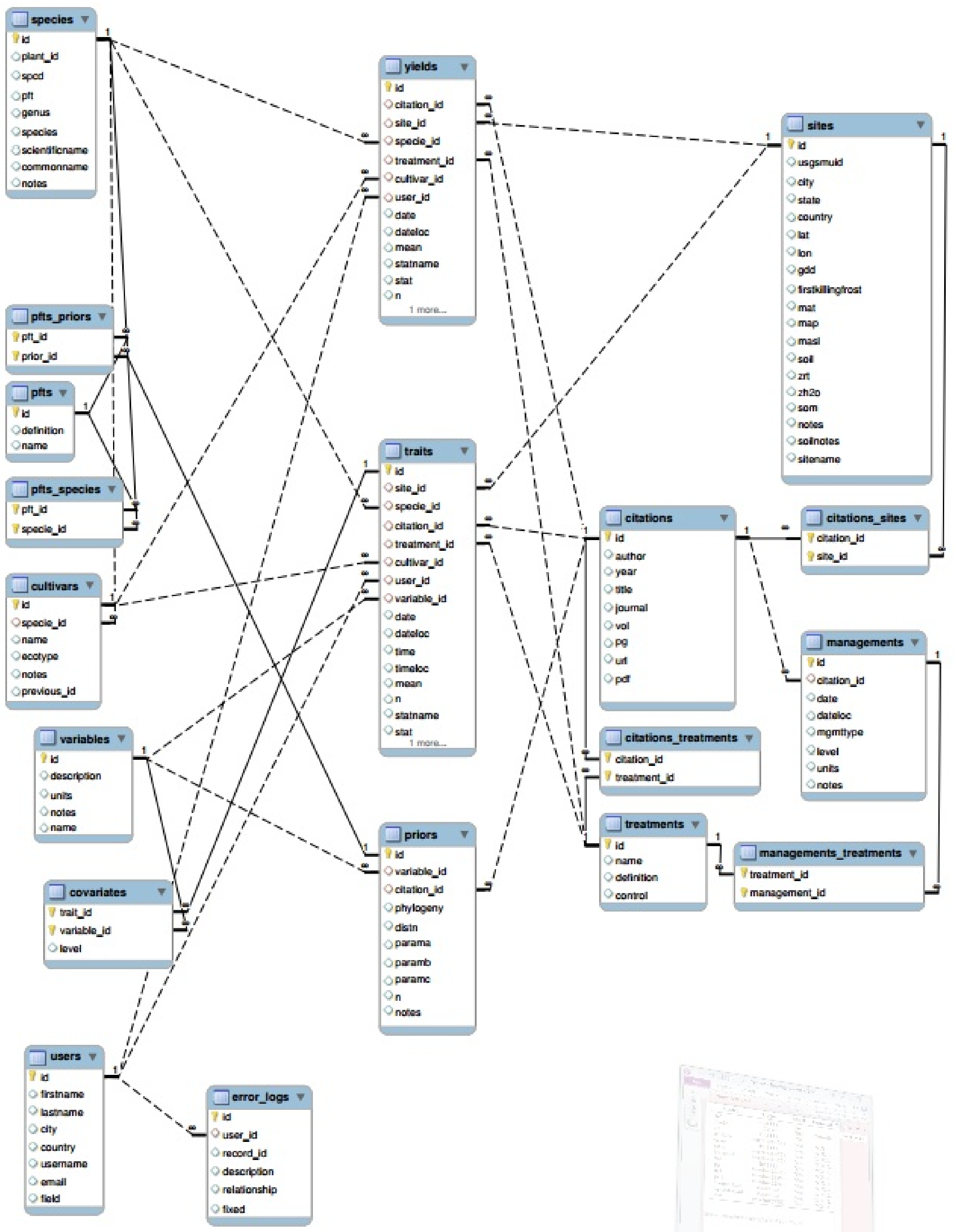
[1] Not all of the columns intended as foreign keys are marked as such in the SQL schema. Thus some lines (and even some tables) may be missing from the schema diagram.

[2] More comprehensive documentation of the schema may be found at https://www.betydb.org/db_docs/index.html. The software used to produce this documentation, SchemeSpy, unfortunately does not document PostgreSQL check constraints. Also note that row counts in this document are not, in general, completely up-to-date. The complete, definitive documentation of the schema is the PostgreSQL code used to produce it, which may be found at https://github.com/PecanProject/bety/blob/master/db/production_structure.sql.

Some background information about intended constraints may be found in the spreadsheet at https://docs.google.com/spreadsheets/d/1fJgaOSR0egq5azYPCP0VRIWw1AazND0OCduyjONH9Wk/edit?pli=1#gid=956483089 and in a PDF document viewable and downloadable at https://www.overleaf.com/2086241dwjyrd. These two documents are not necessarily up-to-date, and not all of the constraints mentioned in them have been implemented. In some instances, constraints on new data have been imposed at the application level but have not yet been imposed on the database itself because of violations in existing data.

BETYdb Tables

Schema: Enhanced Entity-Relationship Model



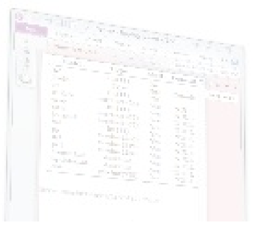


Figure 1: Abbreviated schema for BETYdb, focusing on tables used to store plant trait and yield data. This figure excludes other tables used for PEcAn workflow system provenance and data management and for synchronizing independent instances of BETYdb across many servers. A complete and up-to-date interactive schema is published at https://www.betydb.org/schemas.

Tables

BETYdb is designed as a relational database, somewhat normalized as shown in the structure diagram Figure 1. Each table has a primary key field, `id`, which serves as surrogate key, a unique identifier for each row in the table. Most tables have a natural key defined as well, by which rows can be uniquely identified by real-world attributes. In addition, most tables have a `created_at` and an `updated_at` column to record row-insertion and update timestamps, and the traits and yields tables each have a `user_id` field to record the user who originally entered the data.

A complete list of tables along with short descriptions is provided in Table 2, and a comprehensive description of the contents of each table is provided below. **Note: An up-to-date list of the tables in BETYdb along with their descriptions and diagrams of their interrelationships may be found at https://www.betydb.org/schemas.**

![Alt text] (figures/ug table 2.png "Table 2")

Table and field naming conventions

Each table is given a name that describes the information that it contains. For example, the table containing trait data is called `traits`, the table containing yield data is `yields`, and so on. Each table also has a *primary key*; the primary key is always `id`, and the primary key of a specific table might be identified as `yields.id`. One table can reference another table using a *foreign key*; the foreign key is given a name using the singular form of the foreign table, an underscore, and `id`, e.g. `trait_id` or `yield_id`.

In some cases, two tables can have multiple references to one another, known as a ’many to many’ or ’m:n’ relationship. For example, one citation may contain data from many sites; at the same time, data from a single site may be included in multiple citations. Such relationships use join tables (also known as "association tables" or "junction tables"). Join tables (e.g. Table 4, Table 5, Table 10, Table 12, Table 13) combine the names of the two tables being related. For example, the table used to link `citations` and `sites` is named `citations_sites`. These join tables have two foreign keys (`citation_id` and `site_id` in this example) which together uniquely identify a row of the table (and thus constitute a *candidate key*). (For various implementational reasons, these tables also have a surrogate key named `id`, but in general such a key is extraneous.)

While foreign key columns are identified implicitly by the naming convention whereby such columns end with the suffix `_id`, foreign keys can be made explicit by imposing a *foreign-key constraint* at the database level. Such a constraint identifies the table and column which the foreign key refers to and in addition guaranties that a row with the required value exists. Thus, if there is a foreign-key constraint saying that the column `yields.citation_id` refers to `citations.id`, then if there is a row in the yields table where `cititation_id = 9`, there must also be a row in the citations table where `id = 9`. Explicit foreign keys show up in the schema documentation as an entry in the *References* column of the table listing and as a line between tables in the schema diagrams.

Data Tables

The two data tables, **traits** and **yields**, contain the primary data of interest; all of the other tables provide information associated with these data points. These two tables are structurally very similar as can be seen in Table 17 and Table 20.

traits

The **traits** table contains trait data (Table 17). Traits are measurable phenotypes that are influenced by a plants genotype and environment. Most trait records presently in BETYdb describe tissue chemistry, photosynthetic parameters, and carbon allocation by plants.

yields

The **yields** table includes aboveground biomass in units of Mg per ha (Table 20). Biomass harvested in the fall and winter generally represents what a farmer would harvest, whereas spring and summer harvests are generally from small samples used to monitor the progress of a crop over the course of the growing season. Managements associated with Yields can be used to determine the age of a crop, the fertilization history, harvest history, and other useful information.

Auxillary Tables

sites

Each site is described in the **sites** table (Table 15). A site can have multiple studies and multiple treatments. Sites are identified and should be used as the unit of spatial replication; treatments are used to identify independent units within a site, and these can be compared to other studies at the same site with shared management. "Studies" are not identified explicitly, but independent studies can be identified via shared management entries at the same site.

treatments

The **treatments** table provides a categorical identifier of a study's experimental treatments, if any (Table 18).

Any specific information such as rate of fertilizer application should be recorded in the managements table. A treatment name is used as a categorical (rather than continuous) variable, and the name relates directly to the nomenclature used in the original citation. The treatment name does not have to indicate the level of treatment used in a particular treatment—if required for analysis, this information is recorded as a management.

Each study includes a control treatment; when there is no experimental manipulation, the treatment is considered "observational" and listed as "control". In studies that compare plant traits or yields across different genotypes, site locations, or other factors that are built in to the database, each record is associated with a separate cultivar or site so these are not considered treatments.

For ambiguous cases, the control treatment is assigned to the treatment that best approximates the background condition of the system in its non-experimental state; for this reason, a treatment that approximates conventional agronomic practice may be labeled "control".

managements

The **managements** table provides information on management types, including planting time and methods, stand age, fertilization, irrigation, herbicides, pesticides, as well as harvest method, time and frequency.

The **managements** and **treatments** tables are linked through the `managements_treatments` table (Table 10).

Managements are distinct from treatments in that a management is used to describe the agronomic or experimental intervention that occurs at a specific time and may have a quantity whereas *treatment* is a categorical identifier of an experimental group. Managements include actions that are done to a plant or ecosystem—for example the planting density or rate of fertilizer application.

In other words, managements are the way a treatment becomes quantified. Each treatment can be associated with multiple managements. The combination of managements associated with a particular treatment will distinguish it from other treatments. Each management may be associated with one or more treatments. For example, in a fertilization experiment, planting, irrigation, and herbicide managements would be applied to all plots but the fertilization will be specific to a treatment. For a multi-year experiment, there may be multiple entries for the same type of management, reflecting, for example, repeated applications of herbicide or fertilizer.

covariates

The **covariates** table is used to record one or more covariates associated with each trait record (Table 6). Covariates generally indicate the environmental or experimental conditions under which a measurement was made. The definition of specific covariates can be found in the **variables** table (Table 19). Covariates are required for many of the traits because without covariate information, the trait data will have limited value.

The most frequently used covariate is the temperature at which some respiration rate or photosynthetic parameter was measured. For example, photosynthesis measurements are often recorded along with irradiance, temperature, and relative humidity.

Other covariates include the size or age of the plant or plant part being measured. For example, root respiration is usually measured on fine roots, and if the authors define fine root as < 2mm, the covariate `root_diameter_max` has a value of 2.

pfts

The plant functional type (PFT) table **pfts** is used to group plants for statistical modeling and analysis. Each row in **pfts** contains a PFT that is linked to a set of species in the **species** table. This relationship requires the lookup table **pfts_species** (Table 13). Alternatively, a PFT may be linked to a set of cultivars in the **cultivars** table via the **cultivars_pfts** lookup table. (A PFT can not comprise both cultivars and species.) Furthermore, each PFT can be associated with a set of trait prior probability distributions in the **priors** table (Table 14). This relationship requires the lookup table **pfts_priors** (Table 12).

In many cases, it is appropriate to use a pre-defined default PFT (for example `tempdecid` is temperate deciduous trees). In other cases, a user can define a new PFT to query a specific set of priors or subset of species. For example, there is a PFT for each of the functional types found at the EBI Farm prairie. Such project-specific PFTs can be named using the binomial scheme *projectname.pftname*—for example, `ebifarm.c4grass` instead of simply `c4grass`.

variables

The **variables** table includes definitions of different variables used in the traits, covariates, and priors tables (Table 19). Each variable has a `name` field and is associated with a standardized value for `units`. The `description` field provides additional information or context about the variable.

Join Tables

Join tables are required when each row in one table may be related to many rows in another table, and vice-versa; this is called a 'many-to-many' relationship.

citations_sites

Because a single study may use multiple sites and multiple studies may use the same site, these relationships are tracked in the **citation_sites** table (Table 4).

citations_treatments

Because a single study may include multiple treatments and each treatment may be associated with multiple citations, these relationships are recorded in the **citations_treatments** table (Table 5).

cultivars_pfts

The **cultivars_pfts** table allows a many-to-many relationship between the **pfts** and **cultivars** tables. A PFT that is related to a set of cultivars may not also be related to one or more species (except indirectly, by virtue of its associated cultivars belonging to particular species). A database-level constraint ensures this.

managements_treatments

It is clear that one treatment may have many managements, e.g. tillage, planting, fertilization. It is also important to note that any managements applied to a control plot should, by definition, be associated with all of the treatments in an experiment; this is why the many-to-many association table **managements_treatments** is required.

pfts_priors

The **pfts_priors** table allows a many-to-many relationship between the **pfts** and **priors** tables (Table 12). This allows each pft to be associated with multiple priors and each prior to be associated with multiple pfts.

pfts_species

The **pfts_species** table allows a many-to-many relationship between the **pfts** and **species** tables (Table 13). A PFT that is related to a set of species may not also be related to one or more cultivars (except perhaps indirectly, by virtue of the associated species having certain cultivars). A database-level constraint ensures this.

Installing BETYdb

Installing BETYdb Postgres database

Installing a Production Copy of BETYdb on ebi-forecast or pecandev

These instructions are specifically tailored to the task of adding new instances of BETYdb to a CentOS 5 machine using the deployment scheme we already currently have in place. Thus, they assume the following are installed:

1. Ruby version 2.1.5.
2. Apache 2.2.
3. PostgreSQL 9.3.
4. PostGIS 2.1.3.
5. Git 1.8.2.1.
6. R 3.1.0

In most or all cases later versions of these will work, and in many cases earlier versions may work as well.

IMPORTANT! In what follows, we use the following placeholder strings to represent names that will vary with the installation:

- **`<betyapp>` — the name of the Rails root directory for the BETY app instance**
- **`<betyappuser>` — the operating system account name for the user that will own this BETY app instance; generally, `<betyappuser>` should equal `<betyapp>`, but we distinguish them here for clarity.**
- **`<betydb>` — the name of the database this instance of the BETY app will use**
- **`<dbuser>` — the owner of database `<betydb>`**
- **`<dbpw>` — the password for database user `<dbuser>`**
- **`<bety_url>` — the path portion of the URL at which this BETY app instance will be deployed; this *usually* (but not always) matches `<betyapp>`.**

Step 1: Log in to the Deployment Machine.

You will need to have root access or sudo permission to add a new user and edit the HTTPD configuration files.

Step 2: Add a new user as the owner of the BETYdb instance

It is recommended to run each Rails app under its own user account. By convention, the user account will have the same name as the app. Use the following command to create the user:[1]

```
CREATE_MAIL_SPOOL=no sudo useradd -M <betyappuser>
```

Step 3: Clone a new BETYdb instance.

```
cd /usr/local
sudo -u <betyappuser> -H git clone https://github.com/PecanProject/bety.git <betyapp>
```

(Note that -H should only be used if you created a home directory for `<betyappuser>`.)

Step 4: Log in as the app's user (`<betyappuser>`) and CD to the root directory of the new BETYdb instance.

You can log in by running

```
sudo -u <betyappuser> -H bash -l
```

Then do

```
cd <betyapp>
```

Step 5: Create a Database Configuration File

To do this from the command line, just run[2]

```
cat > config/database.yml << EOF
production:
  adapter: postgis
  encoding: utf-8
  reconnect: false
  database: <betydb>
  pool: 5
  username: <dbuser>
  password: <dbpw>
EOF
```

Step 6: Create the New Database

Just run

```
createdb -U <dbuser> <betydb>
```

You will be prompted for `<dbuser>` 's password.

Step 7: Load the Database Schema and Essential Data

Use the script `script/update-betydb.sh` to load the database schema. `update-betydb.sh` is just a wrapper around the script `load.bety.sh` , which doesn't exist until you download it. To do so, run `update-betydb.sh` without options:

```
./script/update-betydb.sh
```

Then re-run it with the -c, -e, -m, -r, -g, and -d options as follows:

```
./script/update-betydb.sh -c -e -m <localdb id number> -r 0 -g -d <betydb>
```

(Here, "localdb id number" is some integer that is unique to each database. See https://github.com/PecanProject/bety/wiki/Distributed-BETYdb for further information.)

This will create the tables, views, indices, constraints, and functions required for BETYdb. The tables will all be empty except for the following: `formats` , `machines` , `mimetypes` , `schema_migrations` , `spacial_ref_sys` , and `users` . A guestuser account will be added to the users table.[3]

Step 8: Run the Bundler to Install Ruby Gems

To install all Ruby Gems needed by the Rails application, run

```
bundle install --deployment --without development test javascript_testing debug
```

The deployment flag installs the Gems into the vendor subdirectory rather than to the global Ruby Gem location. This makes your deployed instances more independent of one another. The `--without` flag omits installing Gems that are only used in the test and developments environments. (They may be installed later if desired.) Note that the `--deployment` and `--without` options are remembered options.[4]

Step 9: Make a Site Key File

This will create the needed file:

```
cat > config/initializers/site_keys.rb << EOF
REST_AUTH_SITE_KEY          = 'some moderately long, unpredictable text'
REST_AUTH_DIGEST_STRETCHES = 10
EOF
```

Without this, you won't be able to log in to the BETYdb Rails app. You can run the command

```
bundle exec rake secret
```

in the Rails root directory to generate an arbitrary key. For the site key to do any good, it should be kept a secret. *Note that if you ever change the value of `REST_AUTH_SITE_KEY`, all of your user's passwords will be invalidated!*

Step 10: Configure Apache HTTP Server to Serve Your New Rails App Instance[5]

Add the following to your Apache HTTP Server configuration:

```
Alias /<bety_url> /usr/local/<betyapp>/public
<Location /<bety_url>>
    PassengerBaseURI /<bety_url>
    PassengerAppRoot /usr/local/<betyapp>
    # Also add the following as needed (see note below)
    # PassengerRuby [[path to ruby executable]]
</Location>
<Directory /usr/local/<betyapp>/public>
    Allow from all
    Options -Multiviews
    # Uncomment this if you're on Apache >= 2.4:
    # Require all granted
</Directory>
```

If you are using virtual hosting (we *are* on pecandev and ebi-forecast, and so we add this to a VirtualHost configuration in file `/etc/httpd/conf.d/servers.conf`), put this inside the VirtualHost block whose ServerName and/or ServerAlias values correspond to the URL at which you want to access the application ("pecandev.igb.illinois.edu" on pecandev and "ebi-forecast.igb.illinois.edu" on ebi-forecast). Otherwise, include the configuration code at the top level.

Note that the documentation at https://www.phusionpassenger.com/library/deploy/apache/deploy/ruby/ says to use `Require all granted` for Apache versions *greater* than 2.4, but it seems that this is required for version 2.4 itself. (As of this writing, both pecandev and ebi-forecast use version 2.2).

Note: As of this writing, the path to the Ruby executables that should be used are

```
/usr/local/rvm/wrappers/ruby-2.1.5@betydb_rails3/ruby
```

on pecandev, and

```
/usr/local/ruby-2.1.5/bin/ruby
```

on ebi-forecast. Check the value of PassengerDefaultRuby in `mod_passenger.conf`. If the path doesn't match the appropriate setting, you must either override it with a PassengerRuby setting or update it (being sure to override the new default if necessary for any Rails applications that still need to use the old setting).

Where should the PassengerRuby directive be added? If you are using a virtual host and all of the Rails applications served by that host will use this Ruby version, you may add the PassengerRuby directive directly inside the VirtualHost block. Otherwise, restrict it to a particular location by placing it in the Location block for the app instance you are adding, as shown above.

Step 11: Restart the Apache Server

On pecandev, this works:

```
sudo apachectl restart
```

At this point you should be able to view the site at `https://ebi-forecast.igb.illinois.edu/<bety_url>` or `http://pecandev.igb.illinois.edu/<bety_url>`, depending on which machine you deployed to.

Step 12: Create an Administrative Account

Go to the login page (`https://ebi-forecast.igb.illinois.edu/<bety_url>` or `http://pecandev.igb.illinois.edu/<bety_url>`) and click the "Register for BETYdb" button. Fill out at least the required fields (Login, Email, and the two password fields—do *not* change the access level settings!), type the captcha text, and click "Sign Up". You should see the "Thanks for signing up!" message.

Once you have created a user, give that user full access privileges. To do this, use psql:

```
psql -U <dbuser> <betydb>
```

Once psql has started, if the login of the user you wish to alter is "betydb-admin", run

```
UPDATE users SET access_level = 1, page_access_level = 1 WHERE login = 'betydb-admin';
```

Step 13: Set the Guest User Account Password

The Guest User account password will not be set correctly unless you used 'thisisnotasecret' as the site key in step 6, and you shouldn't use this as a site key on a production server. So you need to reset the guestuser password.

Log in as the administrative user you created in step 12 and go to the Users list (menu item `Data/Users`). Search for "guestuser" and click the edit button for that user. Check the "change password" checkbox and then enter "guestuser" in both password fields; then click the "Update" button.

Now log out and try the "Log in as Guest" button.

1

[1] The relevant Phusion Passenger documentation is at https://www.phusionpassenger.com/library/walkthroughs/deploy/ruby/ownserver/nginx/oss/rubygems_norvm/deploy_app.html. You may wish to give the account a home directory—for example, to be able to log in as that user with an ssh key. In this case, leave out the `-M` option.

[2] This sets up a bare-bones `database.yml` file for the production environment only. You may wish to add sections for the development and test environments. See the template file `config/database.yml.template` for a model. It may be convenient in certain cases to use the same database for both development and production. ***The test database, however, should always be different!***

[3] All table, views, constraints, indices, and functions for the BETY database are defined in the file db/production_structure.sql. We could have loaded the database schema by loading this file:

```
bundle exec rake db:structure:load RAILS_ENV=production DB_STRUCTURE=db/production_structure.sql
```

(The `RAILS_ENV` variable tells which block in `config/database.yml` to consult when looking up the database name, and the `DB_STRUCTURE` variable tells which file contains the database schema definition.) This would have created an entirely empty database, however, and would not set the initial indices for newly-generated ids so as to be compatible with the database synchonization scripts.

For the developement environment's database, this all probably does not matter; if you wanted to also set up a development database, then, assuming you have a block for the development environment section of `config/database.yml` that uses the database name `betydb_dev` with the same username and password, you would first run

```
createdb -U <dbuser> betydb_dev
```

and then run

```
bundle exec rake db:structure:load RAILS_ENV=development DB_STRUCTURE=db/production_structure.sql
```

The development environment is usually the default, so the `RAILS_ENV` setting is most likely not strictly necessary. ***Note that we still pull the database schema definition from `db/production_structure.sql` even though we are setting up a development database!***

[4]

[4] If you set up the development and test database specifications in step 5, this means you probably actually want to be able to use these environments. In this case you should omit the `--without` flag.

Some users may want to do *some* testing but may have trouble installing Capybara Webkit. If you don't care about using Capabara Webkit for testing, you can add the `--without javascript_testing` option to your bundle install command to skip installing it. Similarly, add `--without debug` to skip installation of the Selenium web driver, or skip both with `--without javascript_testing debug`. Again, this is a "remembered" option: subsequent "bundle install" commands will automatically re-use the `--without` option unless you remove it from the bundler configuration. But note that if you specify the `--without` option a second time, the list of groups to skip will overwrite any previous remembered list; you must specify the complete list of groups you wish to skip each time unless the list is the same as it was the last time you ran `bundle install`.

[5] Instructions here are for sub-URL deployments. These are deployments where the Rails root is reached via a URL of the form

```
http(s)://hostname/directoryname
```

If you want to serve your app at `http(s)://<hostname>`, and if your `DocumentRoot` equals `/usr/local/<betyapp>/public`, then omit the Alias directive and the Location block.

Setting up a RedHat or CentOS Server

Overview

BETYdb runs on RedHat, CentOS, Ubuntu and OSX; instructions for installing on these systems can be found in the PEcAn documentation.

The original BETYdb (betydb.org) runs on a Red Hat Enterprise Linux version 5.8 Server. To simulate this environment, we have set up a CentOS 5.8 server at pecandev.igb.illinois.edu for testing.

This documentation is aimed at installing BETYdb on a production RedHat, CentOS or similar operating system.

These instructions have been tested and refined on our production (ebi-forecast.igb.illinois.edu) and development (pecandev.igb.illinois.edu) servers. See the "Installing BETY" section of the PEcAn wiki for more generic installation instructions.

If you have any questions about installing BETYdb ... please submit an issue or send an email.

Create an netinstall of the CentOS ISO

Boot from CD and Install

- Following instructions here: http://www.if-not-true-then-false.com/2010/centos-netinstall-network-installation/
- Download this iso: http://vault.centos.org/5.8/isos/x86_64/CentOS-5.8-x86_64-netinstall.iso
    - it is the "netinstall" version, small enough to fit on a CD, but requires internet to install
    - burn to CD
    - boot from CD
- ftp server: `vault.centos.org`
- directory: `/centos/5.8/os/x86_64`

Configuration

1. Add new user

```
adduser johndoe
```

2. add user to root

   ```
   sudo su
   emacs /etc/sudoers
   ```

3. add the line

   ```
   johndoe  ALL=(ALL)  ALL
   ```

Add new repository

instructions here: http://www.rackspace.com/knowledge_center/article/installing-rhel-epel-repo-on-centos-5x-or-6x

```
wget http://dl.fedoraproject.org/pub/epel/5/x86_64/epel-release-5-4.noarch.rpm
sudo rpm -Uvh epel-release-5*.rpm
```

... move to PEcAn wiki page for build environment

https://github.com/PecanProject/pecan/wiki/Installing-PEcAn#installing-bety

Remember to install R 3.0:

```
wget http://cran.us.r-project.org/src/base/R-3/R-3.0.1.tar.gz
tar xzf R-3.0.1.tar.gz
cd R-3.0.1
./configure
make
sudo make install
```

Site data installation

```
cd /usr/local/ebi

rm -rf sites
curl -o sites.tgz http://isda.ncsa.illinois.edu/~kooper/EBI/sites.tgz
tar zxf sites.tgz
sed -i -e "s#/home/kooper/projects/EBI#${PWD}#" sites/*/ED_MET_DRIVER_HEADER
rm sites.tgz

rm -rf inputs
wget http://isda.ncsa.illinois.edu/~kooper/EBI/inputs.tgz
tar zxf inputs.tgz
rm inputs.tgz
```

Database Creation

See the PEcAn wiki for additional information, e.g. on the scripts (load.bety.sh is in the PEcAn repo), running the rails and/or php front-ends

```
# install database (code assumes password is bety)
sudo -u postgres createuser -d -l -P -R -S bety
sudo -u postgres createdb -O bety bety
sudo -u postgres CREATE=YES REMOTESITE=0 scripts/load.bety.sh
REMOTESITE=1 scripts/load.bety.sh
```

Open up `updatedb.sh` and remove the two lines

```
#change to home
cd
```

If these are left in, the script will attempt to put the site data in ~/sites instead of /usr/local/ebi/sites.

```
#fetch and updates the bety database
./updatedb.sh
```

Ruby installation

The version of ruby available through yum is too low, so we have to use rvm

```
user$ \curl -L https://get.rvm.io | sudo bash -s stable
rvm install 1.9
rvm use 1.9

yum upgrade rubygem

yum install mysql-devel.x86_64
yum install ImageMagick-devel.x86_64
yum install rubygem-rails
yum install httpd-devel
wget http://www.sqlite.org/2013/sqlite-autoconf-3071700.tar.gz
tar -xzf sqlite-autoconf-3071700.tar.gz
cd sqlite-autoconf-3071700.tar.gz
./configure
make
make install
```

If you plan to run the JavaScript-based rspec tests, you will also need to install qt:

```
wget http://download.qt-project.org/official_releases/qt/4.8/4.8.6/qt-everywhere-opens
ource-src-4.8.6.tar.gz
tar -xf qt-everywhere-opensource-src-4.8.6.tar.gz
cd qt-everywhere-opensource-src-4.8.6
./configure --prefix=/usr/local/qt-4.8.6
make
make install

export PATH=/usr/local/qt-4.8.6/bin:$PATH
export PATH=/usr/local/ruby-1.8.3/bin:$PATH
# Before running the "gem" command, type "gem environment" and make sure the installat
ion directory matches
# the installation directory used by the BetyDB Rail application.  If you are using RV
M, it should suffice
# simply to switch the the root directory of the application, e.g. "cd /usr/local/ebi"
.
gem install capybara-webkit
```

Then all the ruby gems bety needs.

```
cd /usr/local/bety
gem install bundler
bundle install
# (If you didn't install qt and capybara-webkit, you will need to run "bundle install
--without test_js" instead.)
```

Configuration for Bety:

```
cd /usr/local/ebi/bety
```

```
# create folders for upload folders
mkdir paperclip/files paperclip/file_names
chmod 777 paperclip/files paperclip/file_names

# create folder for log files
mkdir log
touch log/production.log
chmod 0666 log/production.log
touch log/test.log
chmod 0666 log/test.log

cat > config/database.yml << EOF
production:
  adapter: mysql2
  encoding: latin1
  reconnect: false
  database: bety
  pool: 5
  username: bety
  password: bety

test:
  adapter: mysql2
  encoding: latin1
  reconnect: false
  database: test
  pool: 5
  username: bety
  password: bety
EOF


# setup per-instance configuration
cp config/application.yml.template config/application.yml
# Be sure to edit the sample values in application.yml to provide values appropriate t
o your server.
# In particular, the actual value for rest_auth_site_key needs to be replaced with one
 matching the DB you are using.



# configure apache
ln -s /usr/local/ebi/bety/public /var/www/bety

cat > /etc/apache2/conf.d/bety << EOF
RailsEnv production
RailsBaseURI /bety
<Directory /var/www/bety>
   Options FollowSymLinks
   AllowOverride None
   Order allow,deny
   Allow from all
```

```
</Directory>
EOF
```

You may have to change your DocumentRoot in /etc/httpd/conf/httpd.conf from "/var/www/html" to "/var/www" if you get the error message 'Passenger error #2 An error occurred while trying to access '/var/www/html/bety': Cannot resolve possible symlink '/var/www/html/bety': No such file or directory (2)'. Up next make apache2 and passenger play nicely:

```
rvmsudo passenger-install-apache2-module
```

If that fails, try

```
sudo -s
source `rvm gemdir`
```

Finally run the tests

```
cd /usr/local/eby/bety/
bundle exec rake db:test:prepare && bundle exec rake db:fixtures:load RAILS_ENV=test &
& bundle exec rspec spec/
```

[Note: If you are using RVM version 1.11 or later, you can omit the "bundle exec" portion of all rake and rspec commands. For example, the command above could just be typed as

```
   rake db:test:prepare && rake db:fixtures:load RAILS_ENV=test && rspec spec/
```

]

Installing Postgres

Instructions for Installing and Configuring PostgreSQL

These instructions are for installation on Ubuntu. For other Linux-like systems, the steps are similar. Mainly the file locations and the installation command (`apt-get` here) will be different.

1. Run `sudo apt-get install postgresql` .
2. Create user bety: Run `sudo -u postgres createuser -P bety` . Type `bety` when prompted for a password. This is the default user the the BetyDB Rails application connects to the database as.
3. [Optional:] If you wish to administer the database without switching to user postgres (so that you don't have to prefix all your commands with `sudo -u postgres`) add yourself as a superuser: `sudo -u postgres createuser -s <your_ubuntu_login_name>` . This will allow you to run all PostgreSQL commands as yourself without a password.
4. Edit the pg_hba.conf file to allow bety to log in using a password. On Ubuntu, this file is most likely located at `/etc/postgresql/9.3/main/pg_hba.conf` (assuming you are using version 9.3). In detail:

   ```
   sudo su # this allows accessing and editing the conf file
   cd /etc/postgresql/9.3/main
   cp pg_hba.conf pg_hba.conf-original # optional convenience if you need to start ov
   er
   emacs pg_hba.conf # use your favorite editor
   ```

   In this file, you will see a line like this `local all all peer` Assuming this is a fresh install of PostgreSQL, you can give password access to user bety by adding the line `local bety bety md5` immediately *before* this. Then save the file.

5. In order for the configuration to take effect, you must reload the config files: `sudo /etc/init.d/postgresql reload`
6. You can test that user bety exists and can log in by running the following:

   ```
   sudo -u postgres createdb -O bety bety
   psql -U bety
   ```

Manually dumping and installing BETYdb

In this example, we are dumping BETY from "betyhost" to "myserver"

```
ssh betyhost
pg_dump -U postgres bety > bety_YYYYMMDD.sql
rsync bety_YYYYMMDD.sql myserver:
```

Create Copy of BETY

```
ssh myserver
createdb -U postgres bety_copy
```

Enable PostGIS

```
psql -U postgres
postgres=# \c bety_copy
postgres=# CREATE EXTENSION POSTGIS;
```

Also see [[Creating a New PostGIS Enabled Database Template | Automated-Tests#creating-a-new-postgis-enabled-database-template]]

Import database

```
psql -U postgres bety_copy < bety_YYYYMMDD.sql
```

Installing the BETYdb Rails Application

[Note: This guide is aimed at Rails developers and testers. If you are a Pecan developer, you may want to use the notes in the Pecan Wiki instead of or in addition to the notes below.]

Prerequisites

1. Git
2. Ruby 2.1.5 (Anything later than version 1.9.3 will probably work, but 2.1.5 is the officially supported version.) If you are doing Rails development or if you are using Ruby for outside of BETYdb, you may want to install RVM so that you can easily switch between Rails versions and Gem sets.
3. PostgreSQL with the PostGIS extension (see Installing and Configuring PostgreSQL for information on installing and configuring PostgreSQL)
4. Apache web server (optional; developers in particular can simply use the built-in Rails server)

In addition, the scripts below assume you have a working Bash shell. (Windows users might be able to use Cygwin or some other some other port of Linux tools.)

Installing the Rails Application

Installing the Rails code and Ruby Gems

Run these commands to get the Rails code and the Ruby Gems that it uses:

```
# This can be any place you have write permissions for, probably something under your
home directory:
INSTALLATION_DIRECTORY=~/projects

# install bety
cd $INSTALLATION_DIRECTORY

# Developers who will be submitting Git pull requests should make a fork of bety.git o
n GitHub and then
# replace the URL below with the address of their own copy:
git clone https://github.com/PecanProject/bety.git

# install gems
cd bety
gem install bundler # not needed if you already have bundler
bundle install # Use the --without option to avoid installing certain groups of Gems;
see below.
exit
```

[Note: If you can't or don't wish to install the capybara-webkit gem, you can comment it out in the Gemfile before running bundle install.

Note: If you receive "checking for pg_config... no" and the associated errors then you may need update build.pg using the "bundle config" command. For example, to update the bundle executable with the location of the pg_config command you can run: *bundle config build.pg --with-pg-config=/usr/pgsql-9.4/bin/pg_config* This assumes your pg_config is located at */usr/pgsql-9.4/bin/*. Update this path as necessary for your local PostgreSQL/Postgis install]

Minimizing Gem Installation

Certain Ruby Gems are difficult or time-consuming to install on certain platforms, and if they are not essential to your work, you may wish to avoid installing them. (If this isn't a concern, you may skip this section.)

If you look at the Gemfile in the root directory of the BETYdb Rails code, you will see the certain Gems are specified within *group* blocks; this means they are intended to be used only in certain contexts. If you don't intend to use BETYdb within those contexts, you may safely use the `--without` option to `bundle install` to exclude the Gems used only in those contexts.

As an example, the passenger Gem is used only in the production environment. Therefore, it is in a `production` group within the Gemfile. If you run

```
bundle install --without=production
```

the bundler will skip installation of passenger.

Moreover, this is a "remembered" option: the next time you run `bundle install`, it will remember not to install production-only Gems even if you haven't specified the `--without` option. Furthermore, this "remembered option" is also respected by WEBrick, the default Rails server, so it won't complain that you didn't install the passenger Gem.

As another example, the `capybara-webkit` Gem is difficult and time-consuming to install on some platforms, and unless you are running the RSpec tests, you can do without it. (In fact, even if you *are* running RSpec tests, most of the tests don't use `capybara-webkit`, and for those that do, you can either skip them or tell them to use `selenium-webdriver` instead.)

`capybara-webkit` is in a group called `javascript_testing`, so to avoid installing it, run

```
bundle install --without=javascript_testing
```

To see what the remembered "without" options are, run

```
bundle config
```

You can also use `bundle config` to specify directly what groups Bundler should skip. For example, to tell Bundler to ignore all groups except the production group, pass a colon-separated list containing all of the *other* groups to `bundle config --local without`:

```
bundle config --local without development:test:javascript_testing:debug
```

To revert to installing everything when you run `bundle install`, remove the `without` setting from the configuration with

```
bundle config --delete without
```

Configuring Rails

Configure the BETYdb Rails application using the following commands:

```
cd $INSTALLATION_DIRECTORY/bety

# setup bety database configuration
cat > config/database.yml << EOF
development:
  adapter: postgis
  encoding: utf-8
  reconnect: false
  database: bety
  pool: 5
  username: bety
  password: bety
EOF

# Optional: Override some of the default configuration settings given in config/defaul
ts.yml.
cp config/application.yml.template config/application.yml
# Read the comments in this file and set the variable values you are interested in; de
lete the other settings.
```

Installing the Database

Note to join the distributed network of databases, see the chapter "Distributed BETYdb"

In the `script` directory of the bety Rails installation, find and run the update-betydb.sh script:

```
./update-betydb.sh
```

This script is a wrapper script for the script `load.bety.sh` from the Pecan project. The latter can be downloaded by running `update-betydb.sh` without options. Use the `-h` option for more information.

Updating / Syncing the database

See instructions [[Updating-BETY]]

Starting the BETYdb Rails Web Application

1. cd to the bety directory, the directory you cloned the Rails code to.

2. Run `rails s` .
3. You should now be able to visit the web application at http://localhost:3000.
4. To log in, use `Login: carya` , `Password: illinois`

Updating BETYdb When New Versions are Released

Updating BETY database

A new system is in place which will allow you to update the BETY database without losing any local changes (this is still BETA though). See the section Distributed BETYdb wiki for details on updating a local database in a way that retains any local changes.

If you are a BETYdb Ruby-on-Rails developer, you probably don't care about losing changes to your copy of the BETY database. You probably just want a reasonably up-to-date copy that you can use to test code changes with. If so, you may use the script `update-betydb.sh` in the `script` directory. This is just a wrapper for `load.bety.sh` script that makes it easy to download that script (without having to download all of Pecan) and easy to run it with the options you probably want. Step for doing this are:

1. Run `script/update-bety.sh` without options to download a copy of `load.bety.sh` .
2. Run `script/update-bety.sh -i` to write a stock configuration file. This file will contain the following settings:

   ```
    export DATABASE=bety   # update database "bety"; change this name as needed
    export CREATE=YES      # completely overwrite the database you are updating with
   new content
    export FIXSEQUENCE=YES # needed for the CREATE option
    export USERS=YES       # create a stock set of users of various permission levels
    to use when testing
   ```

DO NOT RUN THIS IF YOU HAVE DATA IN YOUR DATABASE!!!!!!!!!!!

The CREATE=YES **WILL** destroy all data in your database

1. Run `script/update-bety.sh` again with no options. This will read the config file you just created and use the settings to update your database copy.

Update the Rails app and schema

If you have an instance of BETY and you might want to update it to the latest version at certain points for the following reasons.

- security updates

- new functionality
- importing data and remote server has newer version

To update BETY you can use the following steps to update your system (this is assuming the VM, if you installed BETY in another location please change the path accordingly). This requires the development version of ruby.

```
sudo -s
# change to BETY
cd /usr/local/bety
# update BETY to latest version
git pull
# install all required gems
bundle install --without test
# update database
RAILS_ENV="production" rake db:migrate
# restart BETY
touch tmp/restart.txt
```

At this point your database should have migrated to the latest version and the BETY application should have restarted.

Load a fresh version of database

See the section of *Updating BETY database* above that pertains to BETYdb Ruby-on-Rails developers.

Distributed instances of BETYdb

Syncing Databases

Exporting Data

The dump.bety.sh script does the following:

- dumps all data except:
  - traits or yields where access_level < 3 or checked = -1
  - runs and workflows contents (After implementing benchmarking, we will need to dump reference runs.)
- anonymizes users
  - sets user1 to carya/illinois with access_level =1, page_access_level = 1,
  - other users get access_level = 3, page_access_level = 4.
  - Users id's are preserved in order to allow admin to identify issues that are uncovered in anonymized version of db.

Output:

- schema as .sql file, named after the most recent value in migrations table, to ensure that the data can be loaded (schema versions must match in order to sync)
- each table exported as a csv file, and tar-zipped. (psql function '\copy')

Importing Data

You can sync a local instance of the BETYdb database with other instances. The load.bety.sh script will import data from other servers.

```
MYSITE=X REMOTESITE=Y load.bety.sh
```

This will set the number range based on the `MYSITE` variable

> (1,000,000,000 *MYSITE) to ((1,000,000,000* (MYSITE + 1)) - 1)

Primary Key Allocations

Assigning a unique set of primary key values to each instance of BETYdb allows each distributed system to create new records that can later be shared, and to import new records from other databases.

Any changes to existing records should be done on the server that owns that record.

Institution	server	url	id	
Energy Biosciences Institute / University of Illinois	ebi-forecast.igb.illinois.edu	https://betydb.org	0	1
Boston University	psql-pecan.bu.edu	https://psql-pecan.bu.edu/bety	1	1 2
Brookhaven National Lab	modex.test.bnl.gov	http://modex.test.bnl.gov/bety	2	2 3
Purdue	bety.bio.purdue.edu	http://bety.bio.purdue.edu/	3	3 4
Virginia Tech			4	4 5
University of Wisconsin	tree.aos.wisc.edu	http://tree.aos.wisc.edu:6480/bety	5	5 6
TERRA Ref	141.142.209.94	http://terraref.ncsa.illinois.edu/bety	6	6 7
TERRA test	141.142.209.95	http://terraref.ncsa.illinois.edu/bety-test	7	7 8
TERRA MEPP UIUC	ebi-forecast.igb.illinois.edu	http://ebi-forecast.igb.illinois.edu/mepp-bety	8	8 9
TERRA TAMU		*.tamu.edu/tamu-bety	9	9 1
Ghent			10	1 - 1
Development / Virtual Machine	localhost	https://localhost:6480/bety	99	9 a

Feedback Tab

New users and 'feedback tab' submissions are sent to all users with Admin privileges (gh-56).

Planned features

- GitHub issue 368 will implement the capacity to specify institution or project-specific design elements (e.g. colors, header, title, text, attribution on the front page).
- to edit data from records owned by another server, must edit on that server (or risk loosing this on next update)

Issues

- to get latest data, have to query each server (or can get full dump from one server, but this is as out of date as the last sync)

BETYdb Development

This is a Draft

Introduction to Ruby-on-Rails

Introduction to MVC

Source Code Map

Misc. Information

Providing model output for download

Access to download model output is in app/views/maps/locations_yields.html.erb

Related Issues / Commits:

https://github.com/PecanProject/bety/commit/7b7d56fdf4c577fa14d65fcf81c677f5a4bf0633

Ruby on Rails Application Overview

Ruby-on-Rails: Developing, Upgrading, and Deploying

Development and Testing

Testing is an integral part of releasing a new version of the BETYdb Rails app. Developers should test prospective code on their development machines prior to submitting a pull request, and code managers should re-test the code before accepting a pull request. See [[running the automated tests | automated-tests]] for complete testing instructions.

Deploying a new version:

For up-to-date instructions on making a new release, see https://dlebauer.gitbooks.io/betydb-documentation/content/management/making_a_new_release.html

At the end of each sprint (or set of sprints, or when ready to deploy a new version), the version should be tagged, and a "release" should be created. See the [[Release Notes Template | Release-Notes-Template]] page for a sample draft of release notes.

(We use the terms "deploy" and "upgrade" roughly synonymously, but "deploy" connotes what code manager does when providing a new version of BETYdb to, say, the production server, and "upgrade" connotes what developers maintaining their own copies of BETYdb do to keep those copies up-to-date. Making a new "release" is part of the deployment process but is not part of the upgrade process for individual users.

The instructions below are written mainly with the code manager in mind but are easily adapted to the needs of developers or maintainers of private copies of BETYdb.)

Deploying or upgrading to a new version of the BETYdb Rails app

Here is an outline "script" for deploying a new version of BETYdb to the production server:

```
cd /usr/local/ebi # or to the Rails root of the copy you are upgrading

# Check the git status to be sure the copy is "clean".
# Generally there shouldn't be any modified files and you should be
# on the master branch.
#
git status

# Get the latest code from Github.
#
git pull

# This is usually not necessary, but if there has been code checked in to the
# master branch since the version to be released, you will have to back up to it.
# Note that we tag releases AFTER deploying them, so we can't use the release tag
# as the git reference.
#
# git checkout <git reference to version being deployed>

# Check the gem bundle.
#
bundle check

# If the the bundle contains uninstalled Gems, run this:
#
bundle install

# Check for pending migrations.
#
RAILS_ENV=production bundle exec rake db:migrate:status
# The RAILS_ENV setting can be omitted by individual developers who only want
# to migrate their development databases.

# If there ARE pending migrations, run them (see below).

# Tell the server there is new code; you may want to run the automated tests
# BEFORE restarting the Rails server.
#
touch tmp/restart.txt
```

[Note: If you are using RVM version 1.11 or later, you can omit the "bundle exec" portion of all rake commands. For example, the command above could be typed as just

```
RAILS_ENV=production rake db:version
```

]

If you do not wish to install the test pieces you can run `bundle install --without test` .

[Note: If you can't or don't wish to install the capybara-webkit gem, you can comment it out in the Gemfile. It is only needed for testing the RSpec tests and it is only needed for a few of them. To avoid running the tests that require it, run rspec with the "--tag ~js" option.]

At this point, the site can be tested, both through the browser and by running the [[automated tests]]. [To do: write hints for running automated tests on production servers]

reference: [protocol for pull requests, testing etc. were discussed in #48]

Running Migrations

If new code requires database migrations, then the database should be dumped, and then the migrations should be run. As noted above, you can find out whether there are migrations pending by comparing the result of

```
RAILS_ENV=production bundle exec rake db:version
```

with the latest migration shown by

```
ls db/migrate
```

If migrations are required, dump the database and run the migrations:

```
pg_dump ebi_production > [some suitable directory]/ebi_production.psql
# (Generally, after the dump file has been created, I rename it to include the timesta
mp
# information in the file name: ebi_production_YYYYMMDDhhmm.psql.  This way multiple d
ump
# files can be stored in the same directory.)
RAILS_ENV=production bundle exec rake db:migrate
```

It is especially important to dump the database if any of the pending migrations will delete data!

An up-to-date copy of db/production_structure.sql should be generated and checked in:

```
RAILS_ENV=production bundle exec rake db:structure:dump
git add db/production_structure.sql
git commit
git push
```

It is recommended to run this (only!) on the production version, since it is the structure of the production database that we want to document. (That said, it is true that the pecandev and beta deployments of the BetyDB database should have precisely the same structure. But in case they do not, we want to capture what is actually be used live.)

Note that we no longer use the `schema.rb` file (see https://github.com/PecanProject/bety/issues/44). The `structure.sql` files allow for complete documentation of the database structure, including features that (by default) are not expressible in the `schema.rb` file. **production_structure.sql is the canonical specification of the complete database schema, the schema which the Rails code is meant to be run against.**

Versioning and Tagging

Protocol for defining and releasing versions

- **During Sprint**
  1. Merge pull requests into the master branch of PecanProject/bety as necessary (in order to avoid conflicts, preferably within one working day) .
  2. [to-do: clarify how we handle pre-releases and why they are necessary] Create a pre-release. This should include a list of key expected features to be implemented during the sprint.
  3. If critical bug fix is required on production server:
     i. Create a branch off of the currently-deployed master version. (If subsequent critical bug fixes are later needed, they can also go on this branch.)
     ii. Apply the bug fix to the branch.
     iii. Test on your development machine.
     iv. Push the branch to PecanProject/bety.
     v. If time permits, pull the branch to pecandev, switch to the branch, do any necessary gem and database updates (see below) and test.
     vi. Repeat the previous step on the production server's beta site.
     vii. Repeat the previous step on the production server's live site. The live server will now be on a branch until the next sprint release.
     viii. If there were any database structure changes, regenerate the production_schema.sql file and commit it.
     ix. Tag the currently-deployed version. [to-do: decide on a schema for versioning branch releases]
     x. Push the tag (and the new production_schema.sql file if it changed) to the repository.

- **Before Sprint Review**
  1. Merge any remaining pull requests into master.
  2. Deploy and test (see below) on `pecandev.igb.illinois.edu:/usr/local/ebi`
  3. Deploy and test on `ebi-forecast.igb.illinois.edu:/usr/local/beta` (for the sprint-review demo).
- **After Sprint Review**
  1. Revise pre-release based on features implemented during sprint.
  2. If any code changes were made after the sprint review, redeploy and test on pecandev and on ebi-forecast's beta site.
  3. Deploy and test the latest master version on `ebi-forecast.igb.illinois.edu:/usr/local/ebi`
  4. Tag the published release.
  5. Ask David to
     - post via the BETYdatabase account on Twitter
     - add doi to release

Version Numbering

We loosely follow semantic versioning.

- Any tag of the form betydb_x.x or betydb_x.x.x refers to a version that has been tested and deployed.
- Changes in the first or second digit of the version number mark some significant change. For example, the change to major version number 2.x marks the change to a new user interface.

Commenting in the Rails Models

Example of a properly commented citation model (/app/models/citations.rb): https://gist.github.com/e68fea1baa070e68b984

And a properly commented covariates model (/app/models/covariates.rb): https://gist.github.com/5d0d96d7be1b1fd7b47c

Automated Tests

We use RSpec with Capybara for testing the BETYdb Rails application. If you are involved in changing code, you should run the entire test suite before submitting a pull request. If you discover a bug, you may wish to *write* a failing test that demonstrates the bug, one that will pass once the bug is fixed.

Running the RSpec tests on BETYdb.

Preparing the test database.

[Note: The instructions below assume that the current Rails environment is `development` (the default). If you have set it to something else--for example, by running `export RAILS_ENV=production` (you might do this, for example, if you only run BETYdb in production mode and didn't bother to set up a development environment or a development database)--modify these instructions accordingly.]

1. If you haven't yet done so, add the PostGIS extension to the default template database, `template1`. The simplest way to do this is to add the PostGIS extension to the `template1` database template (`psql -d template1 -c 'CREATE EXTENSION postgis'`). This is needed because the test database is dropped and recreated every time you run the "prepare" task, and you want it to be created with the PostGIS extension so that tables that rely on it can be recreated. Alternatively, create a new template database that includes the PostGIS extension. (See below for instructions.)
2. Go to the root directory of the copy of BETYdb that you are testing.
3. Ensure Rails has a test database configuration: Open the file `config/database.yml`. It should have a section that looks something like

```
test:
  adapter: postgis
  encoding: unicode
  reconnect: false
  database: test
  pool: 5
  username: bety
  password: bety
```

If it doesn't, you can copy the section for `development` and then change the heading and the database specification to `test`. If you wish to use a template database other than the default (`template1`), also add a line of the form

```
    template: [your template name]
  `
```

4. Create the test database by running

   ```
   bundle exec rake db:test:prepare
   ```

   [Note: If you are using RVM version 1.11 or later, you can omit the "bundle exec" portion of all rake and rspec commands. For example, the command above could just be typed as

   ```
   rake db:test:prepare
   ```

   ] This will check for any pending migrations in the development database. If there are none, it will re-create the `db/development_structure.sql` file, create the database for the testing environment ("test" if you use the configuration listed above), and then create the tables, views, functions, etc. mentioned in the *structure* file. *Note that we no longer use the `schema.rb` file* (see https://github.com/PecanProject/bety/issues/44). The `config.active_record.schema_format` setting has been changed from `:ruby` to `:sql` to permit complete documentation of the database structure, including features that (by default) are not expressible in the `schema.rb` file. **production_structure.sql is the canonical specification of the complete database schema for the last-released version of BETYdb; this is the schema which the latest release of the Rails code is meant to be run against.**

5. If there are pending migrations, run them ("`bundle exec rake db:migrate`") and repeat the previous step. *Note: I found that I ran into permissions problems when attempting this step. To get around this, I started psql as a superuser and then ran "ALTER USER bety WITH SUPERUSER;" to make user `bety` a superuser as well. This shouldn't be a security worry if you are just running tests on your own copy of BETYdb.]
6. Populate the database from the fixtures:

   ```
   RAILS_ENV=test bundle exec rake db:fixtures:load
   ```

   (The fixtures are YAML files under `test/fixtures`.)

Running the tests

- The simplest way to run the tests is to simply run

  ```
  bundle exec rspec
  ```

  (or just `rspec` if you are using RVM > 1.11) from the root directory of the copy BETYdb that you are testing. This will run all the tests under the "spec" directory.

- If you did not install capybara-webkit, you can run all the tests that do not need webkit support with the command

  ```
  bundle exec rspec --tag ~js
  ```

  This will skip all the tests that have the tag `:js => true` and run all the others. (See the Troubleshooting section below for an alternative to capybara-webkit for running tests that need a JavaScript driver.) Conversely, to run *only* the tests requiring capybara-webkit, use the command

  ```
  RAILS_ENV=test_js bundle exec rspec --tag js
  ```

- ***Note that if you are logging in remotely to run the tests, you will need an X-server running in order to run the tests requiring capybara-webkit!*** If you see the error

  ```
  webkit_server: cannot connect to X server
  ```

  this probably means you connected without the -X option. If you see the error

  ```
  webkit_server: Fatal IO error: client killed
  ```

  this probably means the X-server is no longer running. In either case, re-log in with an X-enabled connection.

- To run a specific file of tests, run `bundle exec rspec path/to/testfile`, optionally using the --tag option to skip certain tests.

- You can run a specific test in a file by appending a line number:

  ```
  bundle exec rspec path/to/testfile:line_number_of_first_line_of_test
  ```

This command will appear under the "Failed examples" section of a test run (assuming the test failed); for example

```
bundle exec rspec ./spec/features/management_integration_spec.rb:48
```

- Some useful options to rspec are:
  1. --fail-fast: abort the run on first failure
  2. --format documentation (-fd for short): get nicely formatted output
  3. --backtrace (-b for short): get a full backtrace

Troubleshooting

Sometimes it is useful to carry out a features test manually as a web site user. To do this, start up the rails server in the *test* environment using the command `rails s -etest`. Many or most of the features tests are written in such a way that you can figure out exactly what actions to carry out in order to mimic the test.

Alternatively, use the the Selenium JavaScript driver in place of Capybara Webkit. The spec_helper file is set up to switch drivers automatically if you have set the environment variable RAILS_DEBUG=true. For example:

```
RAILS_DEBUG=true bundle exec rspec -t js
```

will run all your JavaScript-based tests using the Selenium JavaScript driver. *As with Capybara Webkit, if you are logging in to a UNIX machine remotely in order to run the tests, you must have X11-forwarding in effect* (use the -X option to your ssh command). Otherwise all of your Javascript-enabled tests will time out (but not for a full minute!) and then fail.

With the Selenium JavaScript driver in place, each test having `js: true` in its opening line will be run by opening a copy of Firefox and replicating all of the actions specified in the test. If you wish to insert a break point at any point in the test so that you can see the state of the browser at that point, add the line `binding.pry` at the point at which you want to suspend the test. When running the test, pressing `Ctrl-D` at the command line will signal the test to continue. [Note: `binding.pry` will cause an error if your JavaScript driver is not Selenium. So if you switch back to running your test under Capybara Webkit, remove or comment out your breakpoints.]

Note that even tests that don't normally use a JavaScript driver can be debugged with Selenium--simply change the opening line. For example, if you change the opening line of a test from

```
it 'should return "Editing Citation" ' do
```

to

```
it 'should return "Editing Citation" ', js: true do
```

the test will be run with a JavaScript driver (Selenium, if you set RAILS_DEBUG=true). Then you can add `binding.pry` breakpoint lines as needed and see the test in action.

Creating a new PostGIS-enabled database template

As mentioned above, you must have a PostGIS-enabled database template set up in order for the `rake db:test:prepare` command to work. The easiest way to do this is to add the PostGIS extension to the default database template, which is called `template`. This can be done with the command

```
psql -d template1 -c 'CREATE EXTENSION postgis'
```

But you may not want to do this, as this will cause *all* the databases you create to be PostGIS-enabled unless you explicitly specify a non-default template when you are creating them.

So here is how to create a new PostGIS-enabled database template:

1. Start psql and then run the following commands:
2. `CREATE DATABASE template_bety;`. (You can name the template anything you like as long as it's not the name of an existing database. Also, if you have made changes to template1 and want to ensure that template_bety is created from a "pristine" template, create it from template0:

   ```
   CREATE DATABASE template_bety TEMPLATE template0;
   ```

)
3. `\c template_bety`
4. `CREATE EXTENSION postgis;`
5. Change the owner of the newly-created table `spacial_ref_sys` to `bety` (or to the user whose username you specified in the "test" section of `config/database.yml`):

```
ALTER TABLE spatial_ref_sys OWNER TO bety;
```

This will avoid permissions problems when you try to load fixtures into your test database in the case where you haven't made user `bety` a superuser.

6. `UPDATE pg_database SET datistemplate = TRUE WHERE datname = 'template_bety';`
7. Finally, add the line

```
template: template_bety
```

to the test database section of your `database.yml` file.

Code Style

Whitespace conventions in the BETYdb Code

We want to avoid introducing extraneous whitespace into our source files. There are two kinds of extraneous whitespace that you generally won't see in your editor:

1. Extra whitespace at the end of a line ("trailing whitespace")
2. Tab characters

I call tab characters "extraneous whitespace" because if your editor shows tab characters as 4 spaces and my editor shows them as 8 spaces, I'm going to see a lot more whitespace than you are seeing, and the code alignment is going to look all wrong.

In general, please do not use tabs in any Ruby code. (But if there are tabs already in the file before you start editing it, it may be best to leave them alone. See below.)

Git has a built-in easy way to check your files for extraneous space before you commit them. Run the command:

```
git diff --check
```

If you've already staged the files you are about to commit, run

```
git diff --cached --check
```

These commands will find all trailing space and all tab characters that are preceded by a space characters. To catch other tab characters in your indentation, you will have to update your git configuration by running

```
git config --add core.whitespace tab-in-indent
```

Please do this! We don't want any tab characters at all in the source files! (One possible exception: It is OK to have tab characters inside of a string literal.)

You can also check for extraneous whitespace that you may have already committed. To check for extraneous whitespace in your last commit, run

```
git diff --check HEAD^
```

In general, to check for extraneous whitespace commit since commit xyzabc, run

```
git diff --check xyzabc
```

You can combine this with the `git merge-base` command to find all of the whitespace you introduced since branching off of upstream/master:

```
git diff --check $(git merge-base HEAD upstream/master)
```

(If you local copy of master mirrors upstream/master, you can just use "master" in place of "upstream/master" here.)

If you created your branch off your local master branch and haven't changed your local master branch since then, you can just run

```
git diff --check master
```

Please always run some version of the "git diff --check" command before you submit a pull request!

Fixing legacy formatting

Often when I'm coding, I'm tempted to "fix" the formatting of a file I'm working on. It's generally easier to understand the code I'm working on if it is nicely indented to show the logical structure. But there is a drawback to reformatting code: If I want to run "git diff" to find out about the history of a file, I'm going to see a lot of differences I don't care about if the file has been reformatted. And if I re-indent each line, commit the changes, and then run "git blame", it's going to look as though I am responsible for the latest change on each line.

So here are my recommendations:

- If you have to work extensively with a file, and reformatting it will help you to understand the code better, then go ahead and reformat it. But if you do this, please:
    - Look up and follow Rails code formatting guidelines (e.g., 2-space indentation levels, etc.)
    - Devote a single commit to just reformatting--don't include any "significant" code changes in that commit. This will make it easier to figure out the history of a file and what significant code changes were made to it.
- If you are only making minor changes to a file and don't need to reformat the file to figure out what you are doing, just leave the existing formatting alone. Any new code

you add, however, should follow good formatting conventions to the extent possible given that it may be surrounded by poorly-formatted code.

Complex Joins in the Web Interface

Several edit pages have sections for editing associations that use multi-select boxes with huge lists in them. These are being replaced by a search box the will display a list of of entities from the associated table and offer the user the opportunity toggle the "linked" status of each displayed entity--that is, unlink linked entities and link unlinked entities.

Here is a list of files that need to be revised. We'll call the primary table being edited "primary" and the associated table "associate".

- `primary_controller.rb`
- `_edit_primary_associate.html.erb`
- `routes.rb`

These files need to be added:

- `_associate_tbody.html.erb`
- `search_associate.js.erb`
- `edit.js.erb`
- `add_primary_associate.js.erb`
- `rem_primary_associate.js.erb`

[Note: The convention for Rails join tables is to alphabetize the names, and this convention was also used in naming some of the templates and corresponding actions. To maintain this convention, some of the file names above should be altered if `primary` follows `associate` alphabetically. In this case, the file names to use will be `edit_associate_primary.html.erb`, `add_associate_primary.js.erb`, and `rem_associate_primary.js.erb` in place of the names given above, and the names of actions and references to these files in the files themselves will have to be adjusted accordingly.]

Here are step-by-step instructions for making the required changes. The changes in place for editing sites associated with a citation may serve as a model (see the controller file `citations_controller.rb` and the template files in `app/views/citations`; in the case of the template files, these can simply be copied and a search and replace done to replace the primary and associated table names (use the singular form when doing this):

1. Revise the `_edit_primary_associate.html.erb` partial (copy e.g. `app/views/citations/_edit_citations_sites.html.erb` and replace `citation` and `site` with the appropriate strings).
2. Make the `_associate_tbody.html.erb` partial.
3. Alter the `routes.rb` file, adding a member route of the form `get :search_associate` to

the resources for `primary` .

4. Add a `search_associate.js.erb` JavaScript template.
5. Add a search (`search_associate`) action to the controller.
6. If needed, add a search scope to the model for `associate` (or write the action in step 5 in a way so that one isn't required).
7. At this point searching associates on the edit page should work. (If needed, add an instance variable for the associated collection to the `edit` action.)
8. Add an `edit.js.erb` Javascript template.
9. Alter the `edit` action of the controller so that it handles `js` format. (`render layout` should be `false` , and additional template variables may have to be defined in the action.) **Also, alter the update action to be ensure that if it renders the edit template on error, it has the variables it needs.**
10. At this point the "Show only related ..." that appears after doing a search should be functional.
11. In `routes.rb` , rename the `:edit_primary_associate` route to `:add_primary_associate` and change the method from `post` to `get` .
12. Rename the corresponding action in the controller and revise this action (use `CitationsController#add_citations_sites` as a model).
13. Add the `add_primary_associate.js.erb` JavaScript template.
14. At this point, linking an associated entity to the primary entity by clicking on the `+` sign should work.
15. In the controller, revise the `rem_primary_associate` action (use `CitationsController#rem_citations_sites` as a model).
16. Add the `rem_primary_associate.js.erb` JavaScript template.
17. At this point, unlinking an associated entity to the primary entity by clicking on the `X` should work. The "Update" link that appears next to the "Existing Sites Relationships" table caption after unlinking an associate should now work and should erase from view the entity you just unlinked.

Database Constraints

BETYdb contains database level constraints. Building constraints into the SQL schema provides a way of explicitly defining the meaning of the database model and its intended functionality.

Types of Constraints

1. **Value** constraints include:
   - range constraints on continuous variables
   - “enum” constraints on, for example, state or country designations; this is a form of normalization (“US” and “USA” should be folded into a common designation); forms utilized by SELECT controls should perhaps be favored
   - consistency constraints: for example (year, month, day) can’t be (2001, 2, 29); or city-(state)-country vs. latitude-longitude (this may be hard, but some level of checking may not be too difficult; for example, “select * from sites where country in ('US', 'United States', 'USA') and lon > 0;” shouldn’t return any rows)
2. **Foreign key** constraints
   - Prevents accidental deletion of meta-data records in lookup tables
   - Prevents entry of primary data without contextual information required to interpret the data
3. **Non-NULL** constraints
   - Constrains which fields can not be empty
4. **Uniqueness** constraints
   - These define what makes a row unique, by designating a 'natural key' - a combination of fields that make a row unique (distinct from primary keys that are used in cross-table joins).

All constraints are defined in the database schema, `db/production_structure.sql`

Value Constraints (including some NOT NULL constraints)

Global

- Text fields should not have leading or trailing white spaces. (Are there any fields for which this is not the case?)

This can be checked with

~~~
```

CHECK(TRIM(FROM <columnname>) = <columnname>)
```
~~~

Probably sequences of two or more consecutive whitespace characters should be forbidden as well except for various free-form textual columns such as `traits.notes`. This can be checked with

~~~
```

CHECK(REGEXP_REPLACE(TRIM(FROM <columnname>), ' +', ' ') = <columnname>)
```
~~~

For convenience, we should probably define a function so we can just do something like

~~~
```

CHECK(is_normalized(<columnname>))
```
~~~

.

covariates:

- Check that `level` is in the range corresponding to variable referenced by `variable_id`.
- Check that `n` is positive (or > 1 ?) if it is not NULL.
- Check that `statname` and `stat` are either both NULL or both non-NULL. (Alternatively, ensure that `statname` is non-NULL and that it equals the empty string if and only if `stat` is NULL.)
- Check that `statname` is one of "SD", "SE", "MSE", "95%CI", "LSD", "MSD" or possibly "". Consider creating an ENUM data type for this.

managements:

- mgmttype: Constrain to one of the values in the web interface's dropdown. (Is there any reason not to store these in the variables table, or in a separate lookup table? If we record units and range restrictions, this would be useful. On the other hand, if we continue to use a static list of management types, we should create a new ENUM type in the database to enumerate the allowed values.
- level: This should always be non-negative (except in the case that we want to use the special value -999 for mgmttypes where a level has no meaning; if so, we should also constrain level to be non-NULL).

- units: Should be constrained to a known set of values—in fact, on a per mgmttype basis; currently there are several varying designations for the same unit in a number of cases
- dateloc: Should be constrained to specific values. Since there is a value (9) designated as meaning "no data", this column should be constrained to be NOT NULL. We should perhaps constraint this column to have this value if `date` is NULL.
- All values of citation_id in managements should also be associated with treatment via citations_treatments table. Does thie mean: The management should be associated with (at least) one of the treatments associated with the citation specified by `citation_id` ?

species:

- Ensure scientificname LIKE CONCAT(genus, ‘ ‘, species, ‘%’)
- Ensure genus is capitalized (and consists of a single word?).

sites:

- lat (replaced by geometry)
- lon (replaced by geometry)
- som: 0 – 100
- mat: range: -50, 150
- masl: (replaced by geometry)
- map: Minimum is zero. Maximum = ?
- local_time: Range should be -12 to +12. This might more aptly be called timezone. A comment should clarify the meaning; I assume it should mean something like "the number of hours local standard time is ahead of GMT". Some kind of check might be possible to ensure consistence with the longitude.
- sand_pct, clay_pct: These both have range 0--100, and sand_pct + clay_pct should be <= 100.
- sitename: Unique and non-null (see below); also, ensure it does not have leading or trailing white space and no internal sequences of 2 or more consecutive spaces. This will make the uniqueness constraint more meaningful. (A similar white space constraint should apply to all textual keys in all tables.)

traits:

It isn’t clear what a natural key would be, but it would probably involve several foreign key columns. Perhaps (site_id, specie_id, cultivar_id, treatment_id, variable_id, and some combination of date and time fields. But it is important to have some sort of uniqueness constraint other than just the default unique-id constraint. For example, if the web-interface user accidentally presses the Create button on the New Trait page twice, two essentially equal trait rows will be created (they will differ only in the id and timestamp columns). See Uniqueness Constraints below!

- date, dateloc, time, timeloc, date_year, date_month, date_day, time_hour, time_minute: Check date and time fields consistency: For example, if dateloc is 91—97, date and date_year should both be NULL (but maybe old data doesn't adhere to this?). If date_year, date_month, or date_day is NULL, date should be NULL as well. Also, dateloc and timeloc should be constrained to certain meaningful values. (See comment above on managements.dateloc.)
- mean: Check mean is in the range corresponding to the variable referenced by variable_id.
- n, stat, statname: n should always be positive; if n = 1, statname should be NULL. statname should be one of a specified set of values. (See comments above on covariates.stat and covariates.statname.)
- specie_id and cultivar_id need to be consistent with one another.
- access_level: Range is 1--4.

treatments:

- name: Possibly standardize capitalization of names (easiest would be to have all words in all names not capitalized except for proper names and unit names where appropriate; this would convey the most information because (e.g.) author names would stand out from other words). This would need to be done manually to avoid converting proper names to lowercase. As stated below, names should be unique within a citation and site pair; standardizing capitalization will make this constraint more meaningful.
- definition: Treat captitalization similarly to that for names.
- control: There can be more than one control treatment per citation (currently there are). Below in the uniqueness section, it is stated that there can be only one control for a given citation *and site*.
- Since (as stated below) names should be unique within a citation and site pair, standardizing capitalization will make this constraint more meaningful.

users:

- login: Enforce any constraints required by the Rails interface.
- email: Constrain to valid email addresses.
- country: Constrain to valid country names.
- area: This currently isn't very meaningful. Perhaps this should be an ENUM. Alteratively, it could be constraint to be some category word followed by free-form text.

- access_level: Range is 1 - 4.
- page_access_level: Range is 1 - 4.
- postal_code: Ideally, this should be constrained according to the country. Since most users are (currently) from the U.S., we could at least constraint U.S. postal codes to "NNNNNN" or "NNNNNN-NNNN".

yields: [see also traits constraints]

- mean: mean should be in the range of plausible yield values.

Foreign Key Constraints

All foreign key constraints follow the form `table_id references tables`, following Ruby style conventions.

A Github Gist contains a list of foreign key constraints to be placed on BETYdb. The foreign keys are named using the form `fk_foreigntable_lookuptable_1` where the foreigntable has the foreign key.

Non Null Constraints

This is a list of fields that should not be allowed to be null. In all cases, the primary key should not be null. For many-to-many relationship tables, the foreign keys should be non-null.

- citations: author, year, title
- covariates: trait_id, variable_id
- cultivars: specie_id, name
- dbfiles: file_name, file_path, container_type, container_id, machine_id
- ensembles: workflow_id
- formats: dataformat
- formats_variables: ?
- inputs: name, access_level, format_id
- likelihoods: run_id, variable_id, input_id
- machines: hostname
- managements: date, management_type
- methods: name, description, citation_id
- models: model_name, model_path, revision, model_type
- pfts: definition, name
- posteriors: pft_id, format_id

- priors: phylogeny, variable_id, distn, parama, paramb
- runs: model_id, site_id, start_time, finish_time, outdir, outprefix, setting, parameter_list, started_at, ensemble_id (note: finished_at will not be available when record is created)
- sites: lat, lon, sitename, greenhouse
- species: genus, species, scientificname
- traits: specie_id, citation_id, treatment_id, mean, variable_id, checked, access_level
- treatments: name, control
- users: login, name, email, crypted_password, salt, access_level, page_access_level, apikey
- variables: namem, units
- workflows: folder, started_at, site_id, model_id, hostname, params, advanced_edit, start_date, end_date
- yields: specie_id, citation_id, treatment_id, mean, variable_id, checked, access_level

Uniqueness constraints

Uniqueness constraints are "natural keys", i.e. combinations of fields that should be unique within a table. Ideally, each table would have a natural key, but a table may have 0, 1, or many uniqueness constraints.

For many-to-many relationship tables, the foreign key pairs should be unique; these should be implemented but are not listed here for brevity.

- citations: author, year, title
- covariates: trait_id, variable_id
- cultivars: specie_id, name
- dbfiles: file_name, file_path, machine_id
- dbfiles: container_type, container_id
- formats_variables: ?
- formats: site_id, start_date, end_date, format_id
- likelihoods: run_id, variable_id, input_id
- machines: hostname
- managements: date, management_type
- methods: name, citation_id
- models: model_path
- pfts: name
- posteriors: pft_id
- priors: phylogeny, variable_id, distn, parama, paramb
- priors: phylogeny, variable_id, notes
- runs: (?) model_id, site_id, start_time, finish_time, parameter_list, ensemble_id

- sites: lat, lon, sitename
- species: scientificname (not genus, species because there may be multiple varieties)
- traits: site_id, specie_id, citation_id, cultivar_id, treatment_id, date, time, variable_id, entity_id, method_id, date_year, date_month, date_day, time_hour, time_minute
- treatments:
  - for a given citation, name should be unique;
  - for a given citation and site, there should be only one control
- users: (each of the following fields should be independently unique from other records)
  - login
  - email
  - crypted_password
  - salt
  - apikey
- variables: name
- workflows: site_id, model_id, params, advanced_edit, start_date, end_date
- yields: site_id, specie_id, citation_id, cultivar_id, treatment_id, date, entity_id, method_id, date_year, date_month, date_day

Technical Note

Database level constraints violates Ruby's "Active Record" approach.

The Rail Guide on Database Migrations suggests

> The Active Record way claims that intelligence belongs in your models, not in the database. As such, features such as triggers or foreign key constraints, which push some of that intelligence back into the database, are not heavily used.

In order to add database constraints we moved from using `db/schema.rb` to `db/structure.sql`, so that the schema is stored in SQL rather than in Ruby.

Given that the Ruby web application is only one of the ways in which we use the database, it seems reasonable to go with the SQL database-level constraints.

The `db/structure.sql` approach is more straightforward, allows the database to exist independently of its Rails framework, and provides more flexibility.