Constraints for BETYdb

David LeBauer
University of Illinois at Urbana-Champaign

Scott Rohde
University of Illinois at Urbana-Champaign

February 13, 2015

1 Introduction

We are proposing implementation of database-level constraints. We are consciously violating Ruby's "Active Record" approach. The Rails Guide on Active Record (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.

We think, however that the following quote (see http://ewout.name/2009/12/rails-models-with-teeth-and-database-constraints) expresses compelling reasons for bucking the"Active Record way" and enforcing database integrity at the database level:

> Data tends to outlive its applications. A tight data model is a good foundation for an application and can save you a lot of trouble when migrating the data to a different system (years later). Database constraints can make your models even tighter, and enforce integrity rules that are hard to enforce in a multi-process application environment.

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.

There are certain costs, however, to stepping outside of Rails to manage database structure. First, while we may continue to use Rails migrations to manage updates to database structure, in many cases we will have to write the code of those migrations directly in SQL (using the `execute %{ <SQL statements>`

} construct). This increases the chance that our migrations will no longer be DBMS-agnostic.[1]

Second, we can no longer completely define the structure of the database in the file `db/schema.db`—the language Rails has for expressing database structure is simply not expressive enough. We must switch (and have in fact done so) to using `db/production_structure.db` as the repository of the complete and definitive description of our database schema.

Third, we must handle the errors that arise from attempting a change to the database that would violate one of our constraints. In some cases (non-NULL constraints, for example) there are parallel Rails model validations that can enforce constraints (at the cost of some duplication) on the Rails side. In other cases, we will have to catch and handle exceptions generated by the database adapter when a database constraint is violated. This code is likely to be highly DBMS-dependent, but as remarked in the footnote, we are not overly concerned about this.

2 Categories of Constraints

The kinds of constraints we wish to impose may be roughly classified as follows:

1. value constraints
   (a) range constraints on continuous variables
   (b) "enum" constraints on, for example, state or country designations; this is a form of normalization ("US", "USA", and "United States", for example, should be folded into a common designation)
   (c) 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 = 'United States'
   and lon > 0;
   ```

   shouldn't return any rows)
2. foreign key constraints
   We also include here certain consistency constraints involving foreign keys. For example, if `traits.cultivar_id` is non-null, the value of `cultivars.specie_id` in the referred-to row should equal `trait.specie_id`.
3. non-NULL constraints

[1] We can at least partially get around the limitations of built-in Rails constructs for expressing database structures by employing gems that extend these constructs. For example, the *foreigner* gem allows one to write foreign-key constraints within migrations in Ruby rather that having to switch to native SQL code. Given that we think it far more likely that we will abandon using Rails as a front end to our database before we will abandon using PostgreSQL as the management-system for that database, we see little benefit to making use of these extensions.

4. uniqueness constraints–above all, designation of natural key
   In keeping with Rails conventions, we grudgingly continue to use `id` as the primary key for all or nearly all non-join tables, but wherever possible, we shall try to designate some column or set of columns as the "true, natural" key for the table and will back up this designation with a `UNIQUE` constraint on the constituent columns together with non-NULL constraints on those columns. This will help prevent what has been a persistent problem with duplicate data—for example, two rows in the species table that are essentially the same, differing only in the value contained in the `id` column.

In the sections that follow, constraints that have already been implemented are marked with a checkmark (✓).

2.1 Reference Documentation

to do: update these

- Google Doc Spreadsheet
- SQL dump with constraints [Note: The foreign-key constraints listed here have been superceded by the draft "add_foreign_key_contraints" migration in BETYdb git branch ForeignKeyConstraints.]
- redmine issue 1915 and related issues / subtasks

3 Value Constraints

Value constraints serve several useful functions:

- They provide a sanity check on data. For example, we can ensure we don't have negative values for yields or have temperature values of −500 degrees Celsius.
- They help prevent duplicate data. For example, even if we impose a uniqueness constraint on the "name" column of the variables table, this alone won't prevent having one row with name = 'leafC' and another row with name = ' leafC ';
- They help standardize data. For example, without a standardization constraint on country names, a user searching for sites using the string 'United States' may get different results from a user searching with the string 'U.S.'.

Most value constraints will be implemented with a CHECK constraint. More complicated constraints involving multiple tables may require defining a PL/pgSQL function. Even in simpler cases, defining a function to implement a CHECK constraint may make sense if the same sort of constraint will be used

repeatedly. Alternatively, we may wish to use SQL's CREATE DOMAIN statement to define a type that has the constraints we need built in, and then alter the columns we wish to constrain to be of the new type.

Since not-NULL constraints may be viewed as a sort of value constraint and are in any case quite bound up with them, they will be dealt with here as well. The set of not-NULL constraints we wish to use is summarized in a separate section below, together with some general remarks about the use of NULLs.

When a default value other than NULL should be set, these are mentioned here as well.

Each table is given its own section below (excepting certain join tables). In most cases, all columns are mentioned except (1) the 'id' surrogate key columns; (2) foreign-key columns; (3) the `created_at` and `updated_at` timestamp columns (which are dealt with all together in the following "General constraints" section).

After each column name (sometimes with its data type, in parentheses), the constraint that should apply immediately follows, if one has been decided upon. After this, a paragraph (beginning with "Discussion:") commenting on the chosen constraint or discussing considerations for adding additional future constraints may follow.

A checkmark (✓) after a constraint indicates it has been implemented.

3.1 General constraints applying to multiple tables

- Text column values should not have leading or trailing white spaces. To make this easier, we define some PL/pgSQL functions:
  - A function `normalize_whitespace(string)` is defined to return `TRIM(REGEXP_REPLACE(string, '\s+', ' '))`.
  - A function to test for normalization, `is_whitespace_normalized(string)`, returns the result of the test `string = normalize_whitespace(string)`.

  Then check constraints of the form

  `CHECK(is_whitespace_normalized(<columnname>))`

  can then be added for each column that should be whitespace-normalized. These constraints, in conjunction with uniqueness constraints, will go a long way toward ensuring that rows that are essentially duplicates, differing only in the value of the `id` column and in the white space that occurs in their textual columns, do not occur.
- All `created_at` and `updated_at` columns should have default value `NOW()`. If feasible, a trigger function should be defined to set the `updated_at` column to `NOW()` upon UPDATE if no explicit value is given in the update.

3.2 citations

author not NULL, whitespace-normalized

Discussion: This *should* always be just a last name, but there are a few cases where full names or a list of names is given.

year (integer) not NULL

Discussion: Consider adding a range restriction—both a lower bound (say 1800—are we ever going to want to cite Aristotle or even Leeuwenhoek?) and some upper bound (say, 2200), or better, a check constraint such as `CHECK(year <= EXTRACT(YEAR FROM NOW()) + 1)`, assuming we would never have a citation year more than one year in the future.

title not NULL, whitespace-normalized

Discussion: The Data Entry guide recommends using 'NA' to denote an unknown title, journal, volume (`vol`) or page (`pg`). Note, however, that 'NA' currently never appears in either the `journal` column or the `pg` column and that it can't be used for `vol` since it has datatype `integer`. Moreover, since 'NA' is commonly used to designate both "not applicable" and "not available", which mean very different things, we feel it is better to write out which one is meant. (A checkbox or radio button on the Rails app form could both make this easier and help standardize these special values.)

journal not NULL, whitespace-normalized

vol (integer) must be > 0.

Discussion: We'll allow this to be NULL for now until we decide how to deal with missing values.

pg not NULL; should match the regular expression
`'^([1-9]\d*(\u2013[1-9]\d*)?)?$'` (tentative)

Discussion: This should be either a single positive integer or two integers separated by a dash. Currently, nearly all values match the regular expression `'^[1-9]\d*([-\u2013][1-9]\d*)?$'`. Ignoring the one case of the value 'test', those that don't either have leading or trailing spaces, a doubled hyphen, have leading zeros in the numbers, look like dates (e.g. 10/21/10) or are the empty string. Values should probably be normalized upon entry to strip spaces and replace hyphen(s) or the word "to"" with an n-dash, the conventional typographic symbol for representing a range of numbers.

url should match a regular expression for URLs or the emtpy string (tentative)

Discussion: At a minimum this should be whitespace-normalized. Unless we want to continue values like 'paper copy available in blue folder', 'NA', and 'not found', we could require entries to either be the empty string or look like an actual URL. (We could relax this somewhat to allow some finite set of prescribed values like 'unknown' or 'not yet available' in addition to bona fide URLs.)

pdf same as for url

Discussion: Many, but by no means all, have a 'pdf' extension in the filename portion of the URL.

doi should match the regular expression `'^(|10\.\d+(\.\d+)?/.+)$'`

Discussion: All but 13 of the existing values match `'^(|10\.\d+(\.\d+)?/.+)$'`. Most of the non-matching values are either 'NA' or they prefix the doi with 'doi:' or with 'http://dx.doi.org/'. These prefixes should be stripped for succinctness and uniformity since they don't add information not already contained in the column name. Unless we wish to distinguish between unknown DOIs and unregistered citations, we can just use the empty string (rather than 'NA') for both.

3.3 covariates

level should be in the range corresponding to variable referenced by `variable_id` ✓

n should be positive (tentative)

Discussion: For now, this is allowed to be NULL. Decide if 1 is a permissible value.

statname and stat These are interdependent. Check that `statname` is one of "SD", "SE", "MSE","95%CI", "LSD", "MSD" or the empty string. Create a domain type for this since it is also used elsewhere. `stat` should be NULL if statname is the empty string and should be non-null otherwise. Tentative: `stat` should also be NULL (and `statname` equal to the empty string) if `n` is NULL or equal to 1 and non-null otherwise.

Discussion: Do any of these statistics have any meaning if `n` = 1? Should these values be *required* if `n` is greater than 1?

3.4 cultivars

name should be whitespace-normalized

ecotype should be one of a small number of finite values (including the empty string) (tentative)

Discussion: Note that in 30 out of 90 cases, `ecotype` has the same value as `name`! This may be an error! For the cases where ecotype does *not* equal name, there are only 5 distinct values: 'Lowland', 'Upland', 'Boreal Forest', the string 'NULL', and the empty string. This suggests that possibly ecotype values can be restricted to a small finite set.

notes not NULL

3.5 dbfiles

file_name not null, no whitespace (tentative)

Discussion: Possibly be more restrictive: For example, currently all values match `'^[\w.-]*$'`.

file_path no whitespace, non-empty, and non-null (tentative)

Discussion: Possibly be more restrictive: For example, currently all values match `'^[\w.:/-]*$'`.

container_type should be in the set ('Model', 'Posterior', 'Input')

container_id This looks like a foreign key, but it isn't exactly, so we include it here. It refers to the `id` column of one of the tables `models`, `posteriors`, or `inputs`; which one depends on the value of `container_type`. We should check that the refered-to row exists.

Discussion: The probably requires triggers on this table and each of the referred-to tables to implement. This may be too complicated to be worthwhile.

md5 should match `'^([\da-z]{32})?$'`

Discussion: In other words, it's either empty or a 32-digit hexadecimal number represented as text using lowercase letters. All current values comply. The datatype for this column is varchar(255) but could be varchar(32).

3.6 ensembles

notes not NULL

runtype not NULL (tentative)

Discussion: Perhaps there is a regular expression it should match also.

3.7 entities

name should be white-space normalized

notes not NULL

3.8 formats

dataformat — to be determined

notes not NULL

name not NULL, whitespace-normalized

header — to be determined

skip — to be determined

3.9 formats_variables

name — to be determined

unit — to be determined

storage_type — to be determined

column_number — to be determined

3.10 inputs

notes not NULL

start_date (timestamp) — to be determined

end_date (timestamp) — to be determined

name not NULL, whitespace-normalized (tentative)

access_level not NULL, in range 1–4 (tentative)

raw not NULL (tentative)

3.11 likelihoods

loglikelihoods — to be determined

n_eff — to be determined

weight — to be determined

residual — to be determined

3.12 machines

hostname no whitespace (tentative)

3.13 managements

citation_id — to be determined

> Discussion: There should be some kind of consistency constraint above and beyond the foreign key constraint. Perhaps: The management should be associated with (at least) one of the treatments associated with the citation specified by `citation_id`.

date not NULL (tentative)

> Discussion: As a kind of consistency check, we could require certain values for the year, month, or day portions of the date for certain values of dateloc. For example, if dateloc = 8 ("year"), we could require the date

to be of the form 'YYYY-01-01'. (PostgreSQL, unlike MySQL, doesn't allow values of the form 'YYYY-00-00'.) NULLs can be eliminated by setting dateloc to 9 ("no data"), which would effectively mean "ignore the value stored in the date column". Nevertheless, as a consistency check, we could choose some value (e.g. '1000-01-01') that should always be used when dateloc = 9.

dateloc use a "dateloc" domain to constrain to specific values; not NULL (tentative)

Discussion: If this is required to be non-null, we must decide how to handle the few values where dateloc is NULL and date is not null.

mgmttype not NULL; must be one of a defined set of values specified in the Rails model file for the Management class

Discussion: Consider storing these in the variables table, or in a separate lookup table. If we used this table to record units and range restrictions, this would provide additional useful consistency checks between mgmttype, level, and units.

level should be non-negative or the special value -999 (tentative)

Discussion: 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).

Please comment.

units should be non-null if level is not null; should be constrained to a prescribed set of values (tentative)

Discussion: Ideally, the value should be constrained to a known set of values on a per mgmttype basis; currently there are several varying designations for the same unit in a number of cases; for example, `kg ha-1` vs. `kg ha^-1`. See discussion under `mgmttype`.

to do: research what standard set of values to use

notes not NULL

3.14 methods

name not NULL, whitespace-normalized

description not NULL

3.15 mimetypes

type_string not NULL; use a regular expression check

Discussion: A fairly tight check against a regular expression is mentioned in GH #194.

3.16 models

model_name not NULL, no whitespace (tentative)

revision not NULL, no whitespace (tentative)

3.17 modeltypes

name not NULL, no whitespace (tentative)

3.18 modeltypes_formats

tag not NULL, no whitespace

Discussion: All existing tags are strings of lowercase letters. Should this be required? Should tags be unique?

required not NULL

input not NULL

3.19 pfts

definition not NULL

name not NULL, should match `'^[-\w]+(\.[-\w]+)*$'` (tentative)

Discussion: Existing entries seem to follow a naming pattern so that they all match the given regular expression. We could relax this to "no whitespace" or "whitespace-normalized."

pft_type — to be determined

Discussion: Currently this is always 'plant'.

3.20 priors

phylogeny not NULL, whitespace-normalized

distn not NULL; should be one of 'unif', 'gamma', 'beta', 'exp', 'lnorm', 'weibull', and 'norm'

parama, paramb, paramc Some sanity check based on the value of distn should be possible. (For example, should some distributions allow only positive numbers?)

n — to be determined

Discussion: This should probably always be >= 2 unless we want to allow certain values (1, 0, or negative numbers) as special values—say to indicate an unknown value for n. (1 is not a good choice here, however, since it would easily be misinterpreted as a legitimate sample size.) Note that several existing values are 0! Also, several are NULL.

notes not NULL

3.21 projects

name not NULL, whitespace-normalized

outdir — to be determined

Discussion: At least non-null.

description not NULL

3.22 runs

start_time (timestamp) — to be determined

finish_time (timestamp) — to be determined

outdir — to be determined

outprefix — to be determined

setting — to be determined

parameter_list — to be determined

started_at (timestamp) — to be determined

finsihed_at (timestamp) — to be determined

3.23 sites:

city, state, country — to be determined

Discussion: Standardize geographic names (city, country, state) using TIGER / OpenStreetMap. Note that `state` is currently used not only for U.S. states, but states, regions, or provinces in other countries. This may be harder to standardize. (Question: Does TIGER only deal with U.S. geographic names?)

Use geocoding / reverse geocoding to enfoce consistency between lat, lon and city, state, country

Country names should be standardized, and probably this standardization should be enforced with a constraint.

som should be in range 0–100 (this is a percentage)

mat should be in range -25–40

Discussion: This should be more than adequate: The highest annual mean temperature recorded is 34.4 degrees Celsius and the lowest is -19.4 degrees Celsius.

masl — replaced by geometry

Discussion: Although this has been replaced, an altitude restriction could be placed on `geometry`.

map should be in range 0–12000

Discussion: According to Weather Underground, the wettest place in the world has an average annual precipitation of 11871mm.

soil — to be determined

Discussion: Should at least be whitespace-normalized and non-null, but constraining to some finite list of values may be possible (anomalous information could go into the `soilnotes` column).

Right now, there are 31 distinct descriptors, but many of these are the same if variations in capitalization and whitespace are ignored.

soilnotes not NULL

sitename not NULL, whitespace-normalized

greenhouse (boolean) — to be determined

Discussion: Ideally, a not-NULL constraint should be enforced. But there are 272 rows where this is NULL.

local_time (int4) — to be determined

Discussion: A comment should clarify the meaning; I assume it should mean something like "the number of hours local standard time is ahead of GMT": this column should probably be called `timezone` or better, `utc_offset`. Moreover, integer is a poor choice of datatype since certain locales—Iran and Newfoundland for example—have time zones on the half hour, and some locales even use quarter-hours offsets.

This column can be confined to a finite set of values: see the list of all UTC offsets at http://en.wikipedia.org/wiki/Time_zone#List_of_UTC_offsets.

Some kind of check certainly possible to ensure consistency with the longitude extracted from the geometry. The offset is approximately equal to the longitude divided by 15, but in areas like China and Greenland where one timezone spans a wide longitude range, the difference can be as much as 3 or 4 hours.

sand_pct, clay_pct should be in range 0–100, and `sand_pct` + `clay_pct` should be ≤ 100; more succinctly, each should be non-negative and their sum should be at most 100.

geometry — to be determined

Discussion: This replaces lat, lon, and masl. It is not clear to me what constraints (if any) can or should be placed on geometry.

3.24 species:

genus capitalized, not NULL, no whitespace

species not NULL, whitespace-normalized

scientificname not NULL, whitespace-normalized; also ensure `scientificname` matches `FORMAT('^%s %s', genus, species)` (tentative)

Discussion: There are 205 cases where the above match fails. For most of these, either the genus name or the species name is not contained in the value of `scientificname`. It's not clear if these are due to data entry errors or the use of synonyms. Some are cases where the genus is abbreviated in the `scientificname` column. Often, the variety or subspecies name appears instead of the species name in the `species` column.

A more comprehensive match restriction might be possible—something like

```
CHECK(scientificname ~
FORMAT('^%s %s( (ssp\.|var\.?|\u00d7) \w+)?$', genus, species))
```

though this doesn't account for authority designations, e.g. "Hyacinthoides italica (L.) Rothm." In particular, it would be desirable to standardize the hybrid designator to the "times" symbol with a space on either side and no longer use the letter "x".

commonname not NULL, whitespace-normalized

notes not NULL

The remaining columns come from the USDA plant database, and we won't be overly concerned with them. Nevertheless, here are some notes:

- Except for species, most taxinomic divisions ("Family", "Class", "Division", "Kingdom", etc.) should probably be constrained to a single capitalized word (no spaces) or the empty string (if the information is not given).
- Symbols: These should contain no whitespace and should consist of digits and upper-case latin letters. Note that "Symbol" and "AcceptedSymbol" are almost always identical.
- Duration: This is always 'Annual', 'Biennial', or 'Perennial', or some combination of these. The combinations should be standardized. For example, both 'Annual, Perennial' and 'Perennial, Annual' occur. The empty string

(or some other special value) should be allowed for unspecified information.

- GrowthHabit: Of the 40-some thousand non-NULL, non-empty values given in this column there are only 66 distinct ones. This number could be reduced by standardizing: Most values are comma-separated combinations of 'Tree', 'Shrub', 'Vine', etc. but the order varies.
- Many columns are essentially booleans but use the varchar(255) with values 'Yes', 'No', or NULL instead. These should be converted to bona fide boolean types or to a subdomain of varchar with values 'Yes', 'No', and some special values to indicate unknown, non-yet-entered, or inapplicable information.
- Many columns use only the values 'High', 'Medium', 'Low', and 'None', (and perhaps also NULL and the empty string). A domain should be created for this column type.
- pH_Minimum, pH_Maximum: These obviously should be confined to the range 0–14, with pH_Minimun < pH_Maximum. Over 40,000 rows have 0.00 in both these columns, which is nonsensical! A single column range could be used in place of these two column.
- Several columns refer to seasons of the year. A domain should be created.
- Several quantitative columns should be constrained to be non-negative.
- MinFrostFreeDays: Obviously should be non-negative and at most 365 (366). Similarly for other "Days" columns.
- A few other columns have values that are confined to a small finite set of string values.

3.25 trait_covariate_associations

required not NULL

3.26 traits

date, dateloc, time, timeloc, date_year, date_month, date_day, time_hour, time_minute — to be determined: but at least constrain dateloc and timeloc to be in the set of recognized values; define a domain for this (see discussion of `managements.dateloc`)
Discussion: 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 See comments above on covariates.

specie_id and cultivar_id these need to be consistent with one another

notes not NULL

checked not NULL, equal to 1, 0, or -1

access_level not NULL; range is 1–4

3.27 treatments:

name not NULL, whitespace-normalized

Discussion: Ideally, two names that are identical except for capitalization should in fact be identical but this would be hard to enforce.

definition not NULL, whitespace-normalized

control not NULL (tentative)

Discussion: The value is constrained in another way be the requirement stated in the Uniqueness portion of the constraints spreadsheet: "there must be a control for each (citation_id, site_id) tuple". "A control" means a treatment for which `control = true`. The meaning of this is not clear, however, since `site_id` and `citation_id` are not columns of this table.

3.28 users:

login — to be determined: at least not NULL

Description: Enforce any constraints required by the Rails interface.

name not NULL, whitespace-normalized

email not NULL; constrain to (potentially) valid email addresses

city not NULL, whitespace-normalized

country not NULL, whitespace-normalized

area to be determined—at least not NULL

This currently isn't very meaningful. We have a mixture of values like "tundra" with values like "Agriculture" and "Industry".

crypted_password not NULL

salt not NULL

remember_token to be determined

remember_token_expires_at (timestamp) to be determined

access_level not NULL; range is 1–4.

page_access_level not NULL; range is 1–4.

state_prov to be determined—at least not NULL

Discussion: For U.S. users, this could be constrained to valid state or territory names.

postal_code to be determined–at least not NULL

Discussion: 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 "NNNNN" or "NNNNN-NNNN".

3.29 variables

description not NULL, whitespace-normalized

units to be determined

Discussion: These should be standardized. See also discussion of `managments.units`.

notes not NULL

name not NULL, whitespace-normalized

min, max require `min` < `max` if both are non-null

Discussion: Note that these are both of type varchar. Ideally, a single column of type `numrange` could be used. If we stay with type varchar for these columns, there is no reason to allow them to be null. But in this case we should also require that the strings look like a number (e.g. `~ '^-?\d+(\.\d*)?$'`) possibly allowing in addition certain prescribed values such as `infinity`, `-infinity`, `N/A`, `unspecified`, etc.

standard_name, standard_units, label, type — to be determined

Discussion: None of these columns is currently used: every row has either NULL or the empty string in each of these columns. Do we really want to keep them?

3.30 workflows

folder — to be determined: at least not NULL and no whitespace

started_at (timestamp) — to be determined

finished_at (timestamp) — to be determined

hostname — to be determined: at least not NULL and no whitespace

params — to be determined: at least not NULL and whitespace-normalized

advanced_edit (bool) not NULL

start_date (timestamp) — to be determined

end_date — to be determined

3.31 yields:

date, dateloc, date_year, date_month, date_day — to be determined
Discussion: Check date 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 should be constrained to certain meaningful values. (See comment above on `managements.dateloc`.)

mean not NULL, at least 0 and at most 1000 (tentative)
Discussion: The current maximum occurring in this column is 777.0.

n, stat, statname See comments above on covariates.

specie_id and cultivar_id these need to be consistent with one another

notes not NULL

checked not NULL, equal to 1, 0, or -1

access_level not NULL; range is 1–4

4 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. Often, however, we will use more meaningful names instead of these auto-generated ones.

5 Not-Null Constraints

5.1 Reasons to Avoid Nulls

1. The interpretation of a NULL is almost never defined for the database user. A NULL may be used for any of the following reasons, and generally unclear what the reason was in any particular case:
   (a) The data entry operator has to look up the information and hasn't yet done so. (And maybe they will forget to ever do so!)
   (b) The attribute is not applicable for the row in which the null appears. For example, `city` might be null for a site in the middle of the desert not near any city. Or `cultivar_id` may not apply for a trait measurement carried out on a non-domesticated species.
   (c) The information is pending. For example, `citation.doi` might be null until a DOI has been assigned.
   (d) The information was never collected and is irretrievably missing. For example, perhaps a long-ago trait measurement was carried out and

recorded on a farm crop, but the citation author failed to specify the cultivar of the species measured.

(e) The information is missing, is retrievable, but it is not considered worth the effort to retrieve and add it. For example, a site with NULL in the `city` column may be in or near a city, but we may not consider it important to fill in this information.

2. The logic for using nulls defies common sense and is therefore exceptionally prone to yielding erroneous results. Here are some examples:

   (a) Supposed we have a table `stats` with integer columns `a` and `b`. Then, normally, we should expect the queries

   ```
   SELECT sum(a) + sum(b) FROM stats;
   ```

   and

   ```
   SELECT sum(a + b) FROM stats;
   ```

   to produce the same result, but they probably won't if either column is allowed to contain a NULL.

   (b) It is commonly said the NULL stands for a value, but for a value we don't happen to know. But consider a column `a` of some numeric type. Then even in cases where we don't know the value of `a`, we *do* know that `a - a = 0`. But most SQL products don't. For example, compare these query results:

   ```
   bety=# select count(*) from traits;
    count
   -------
    13064
   (1 row)

   bety=# select count(*) from traits where mean - mean = 0;
    count
   -------
    13058
   (1 row)
   ```

   The six missing rows in the second query result were rows in which `mean` was NULL. In such a case PostgreSQL considers `mean - mean` to be NULL, not zero, and in the context of a WHERE clause (but not in other contexts!) NULL is considered to be false.

   (c) Nulls are grouped together by the `GROUP BY` clause but don't compare equal if we use the = operator. (`NULL = NULL` is evaluated to `NULL`, not `true`.) Consider for example the difference between the results of the following two queries:

   ```
   SELECT
       COUNT (*),
   ```

```
    sitename
FROM
    sites
GROUP BY
    sitename
HAVING
    COUNT (*) > 1
ORDER BY
    sitename;

SELECT
    COUNT (*),
    sitename
FROM
    sites s1
WHERE
    EXISTS (
        SELECT
            1
        FROM
            sites s2
        WHERE
            s1. ID != s2. ID
        AND s1.sitename = s2.sitename
    )
GROUP BY
    sitename
ORDER BY
    sitename;
```

The first query groups togther the rows with NULL in the sitename column. The second query ignores these rows completely and thus has one fewer group in the result. To get back the missing row, we have to replace `AND s1.sitename = s2.sitename` with

```
AND (s1.sitename = s2.sitename
   OR
     s1.sitename IS NULL
       AND
     S2.sitename IS NULL)
```

These are just a few of the common sense-defying properties that SQL's 3-valued logic uses to deal with NULLs.

In summary, if the inherent ambiguity of NULLs makes it nearly impossible to come up with a well-defined predicate that defines the relation associated with a table and tells us exactly what fact the presense of a given row in the

table is supposed to represent. But even should we do so, the slipperiness of the logic SQL uses to manipulate NULLs makes it highly likely that the query we write to get results from that data won't mean what we intend it to.

5.2 Summary of Not-NULL constraints

Some nulls are worse than others perhaps. We probably don't care much that over half the rows in the treatments table have NULL in the `user_id` column since it is unlikely we will write any queries using this column.

Given the overwhelming task of eliminating the use of nulls from a database that has allowed them for so long, we have to make some priorities. The following is a list of columns from which we wish to eliminate (and prevent future) nulls in the near term.

This is a list of fields that should not be allowed to be null. In all cases, columns making up part of a candidate key (shown below in parenthesized groups) should not be null. For many-to-many relationship tables, the foreign keys should be non-null. In general, nulls can be eschewed from all textual columns since the empty string can easily be used instead.

- citations: (author, year, title), journal, pg
- citations_sites: (citation_id, site_id)
- citations_treatments: (citation_id, treatment_id)
- covariates: (trait_id, variable_id)
- cultivars: (specie_id, name), ecotype, notes
- dbfiles: (file_name, file_path, machine_id), container_type, container_id
- ensembles: workflow_id
- entities: name, notes (missing values should be the empty string)
- formats: (dataformat)
- formats_variables: (format_id, variable_id)
- inputs: name, access_level, (site_id, start_date, end_date, format_id)
- inputs_runs: (input_id, run_id)
- inputs_variables: (input_id, variable_id)
- likelihoods: (run_id, variable_id, input_id)
- machines: (hostname)
- managements: (date, management_type)
- managements_treatments: (treatment_id, management_id)
- methods: (name, citation_id), description
- mimetypes: (type_string)
- models: (model_name, revision)[2], model_path, model_type
- pfts: definition, (name, modeltype_id)
- pfts_priors: (pft_id, prior_id)
- posteriors: (pft_id, format_id)

[2]Using this as a key is still at the proposal stage.

- priors: (phylogeny, variable_id, notes), (phylogeny, variable_id, citation_id), distn, parama, paramb [we've repeated column names that are part of two keys]; n: If we require n to be non-NULL, we have to decide how to handle missing information.
- runs: (model_id, site_id, start_time, finish_time, parameter_list, ensemble_id), outdir, outprefix, setting, started_at (note: finished_at will not be available when record is created)
- sites: (geometry, sitename), greenhouse
- species: genus, species, (scientificname)
- 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), mean, checked, access_level[3]
- treatments: name, control, definition (name should probably also be constrained to be non-empty; but definition may be empty.)
- users: (login), name, email, crypted_password, salt, access_level, page_access_level, (apikey)[4]
- variables: (name), units
- workflows: folder, started_at, (site_id, model_id, params, advanced_edit, start_date, end_date), hostname
- yields: (citation_id, site_id, specie_id, treatment_id, cultivar_id, method_id, entity_id, date_year, date_month, date_day), checked, access_level, mean[5]

6 Uniqueness constraints

These are "natural keys", that is, combinations of columns that provide a natural way to identify, select, and distinguish members of the set of entities the table is meant to represent. More generally, each is a candidate key, that is, a combination of non-NULL columns guaranteed to be unique within a table.[6] Ideally, each table would have a natural key, but a table may have more than one candidate key. Each table should *always* have at least one candidate key, and ideally this will be something other than the auto-numbered `id` column that Rails expects each table to have by default.

For many-to-many relationship tables, the foreign key pairs should be unique; these should be implemented but are not listed here for brevity except where the table contains columns other than the foreign key and timestamp columns.

- citations: author, year, title
- covariates: trait_id, variable_id

[3] The key is still at the proposal stage. Moreover, *many* values in some of these columns are currently NULL.

[4] It is yet to be decided with certainty which columns should be keys.

[5] The key here is yet to be finalized.

[6] Note that SQL's UNIQUE constraint does not prevent duplicate rows if even one column of the constraint is allowed to be NULL. It only guarantees a row is unique in the case where all of its columns that are involved in the uniqueness constraint are non-NULL.

- cultivars: specie_id, name
- dbfiles: file_name, file_path, machine_id
- formats: site_id, start_date, end_date, format_id
- formats_variables: format_id, variable_id
- likelihoods: run_id, variable_id, input_id
- machines: hostname
- managements: date, management_type
- methods: name, citation_id
- models: model_name, revision (tentative)
- pfts: name, modeltype_id
- posteriors: pft_id, format_id
- priors: phylogeny, variable_id, ~~distn, parama, paramb~~, citation_id
- priors: phylogeny, variable_id, notes
- runs: (?) model_id, site_id, start_time, finish_time, parameter_list, ensemble_id
- sites: geometry, 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. (Note that there is no citation_id column in the treatments table. The association of treatments with citations is a many-to-many one via the citations_treatments table. So the constraint, in words, is something like this: "Given two rows of the treatments table with distinct values for "name", no citation should be associated with both rows." Is this really the restriction we want?)
  - For a given citation and site, there should be only one control.

  It is not at all clear how treatment is associated with a site!
- users: (each of the following fields should be independently unique from other records)
  - login
  - email [disputed]
  - crypted_password [disputed]
  - salt [disputed]
  - 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