Development of database and algorithms to support the National Wetland Plant List

Donald J. Leopold, Ph.D.
Email: djleopold@esf.edu

and

Matthew F. Buff, Ph.D. candidate
Email: mfbuff@syr.edu

Department of Environmental and Forest Biology
SUNY-ESF

1 Forestry Drive
Syracuse, New York 13210

Table of Contents

Part 1 – Introduction

Problem Statement

The United States Army Corps of Engineers (USACE) currently has responsibility to administer the National Wetland Plant List (NWPL). While the NWPL is being updated and revised, the wetland indicator status of individual species may be challenged. Research is needed to develop the database and to analyze data submitted because of these challenges.

Basic Approach (from the SOI #W912HZ-11-SOI-0022)

(1) Develop a standardized database of field occurrences in both wetlands and uplands for species at a regional and national scale; and,

(2) Develop algorithms to analyze data to determine if existing data can place a wetland species into one of the various wetland frequency groups. Wetland species in the Facultative (FAC) and Facultative Upland (FACU) categories will be given priority, especially the more common species that may have questionable wetland ratings.

Objectives (from the SOI)

(1) To develop a comprehensive database of various wetland plant species' field data collected from a variety of sources;

(2) To normalize the data such that it can be evaluated equally across the region for frequency of occurrences in wetlands;

(3) To develop algorithms that will evaluate the data mathematically and provide frequency statements about wetland plant occurrences with supporting statistical confidence levels; and

(4) To assess various wetland indicator status ratings (once the models are operational) of species with a strong data set to see if their ratings have been reasonably assigned.

Additional objectives were developed during the course of this research in discussions with the Army Corps of Engineers and the National Technical Committee on Wetland Vegetation:

(5) Identify species with highly variable ratings across regions;

(6) Within the Arid West region, compare ratings between California and the rest of the region.

Part 2. - Developing a database (objective 1)

As detailed in the research proposal, data sources must contain sufficient detail and geographic scope to be useful in addressing the objectives of this study. Namely, the source data must already exist in digital form; field observations must contain sufficient data to allow the identification of each record as wetland or upland; and observations must contain geographic location sufficient to allow grouping by USACE Region.

A search for relevant data was conducted in the scientific literature and via the websites of the following organizations: Global Index of Vegetation-Plot Databases (Dengler *et al.*, 2011), The International Association for Vegetation Science (IAVS) databases for vegetation scientists (http://www.iavs.org/ResourcesDatabases.aspx), the United States National Park Service (NPS) Integrated Resource Management Applications (IRMA) Portal (https://irma.nps.gov), and the United States Forest Service (USFS) Forest Inventory and Analysis (FIA) Program (http://www.fia.fs.fed.us/).

The GIVD was searched for databases that included at least some records within the United States. The search result (Table 1) was further narrowed to eliminate results containing data from single locations, e.g. "Vegetation Database of Observatory Woods, Wisconsin".

Table 1: Data search results from the Global Index of Vegetation-Plot Databases (GIVD; http://www.givd.info)

ID	Name of Database	Vegetation plots
00-00-003	SALVIAS	13,661
NA-US-001	Forest Inventory and Analysis Database of the United States of America (FIA)	538,428
NA-US-002	VegBank	22,629[1]
NA-US-006	Carolina Vegetation Survey Database	8,283
NA-US-007	FIADB Vegetation Diversity and Structure Indicator (VEG)	2,564[2]
NA-US-008	Plant Community Survey and Resurvey Data from the Wisconsin Plant Ecology Laboratory	2,582
NA-US-013	Natural Heritage Vegetation Database for West Virginia	3,896
NA-US-014	Alaska Arctic Vegetation Archive	2,275

[1] The current number of records in VegBank is 75,084.

[2] This plot count refers only to FIADB vegetation plots and does not refer to forest inventory plots.

The IAVS lists the following three sources of vegetation plot data: The Long Term Ecological Research (LTER) Network (http://www.lternet.edu/data/), and the United States Geological Survey (USGS) – National Park Service (NPS) Natural Resource Inventory and Monitoring, Vegetation Mapping Inventory Program (VIP) (http://www.usgs.gov/core_science_systems/csas/vip/index.html, http://science.nature.nps.gov/im/inventory/veg/plots.cfm, https://irma.nps.gov/App/Reference/Profile?Code=2097270) henceforth abbreviated as NPS.

A list of potential sources meeting basic scope and scale requirements was compiled (Table 2). A more detailed examination of each source was conducted in order to determine the suitability of the data for the purposes of this research.

Some sources are not comprised of a single database compiled for a single study, but contain a collection of data from multiple studies. For example, the LTER lists 2,056 datasets under the category "vegetation" that were collected at one or more of 25 LTER sites. The datasets were collected for a large number of different studies and are variable in data layout and content. Data collections are more time consuming to process than uniform databases.

Some surveys designed to map and describe plant communities involve placing plots in a somewhat subjective manner, e.g. in order to maximize the number of communities sampled or to capture representative samples of a plant community. This type of plot placement does not follow a spatial statistical sampling design and therefore precludes certain statistical analyses of the data. Examples of this type of plot arrangement in space can be found in the Carolina Vegetation Survey Database, e.g. Boyle *et al.* (2007) and the NPS data, e.g. Lubinski *et al.* (2003).

Another practical requirement for a potential data source is a consistent use of plant taxonomy. A common method of dealing with taxonomy is to use plant symbol codes from the United States Department of Agriculture (USDA) Natural Resources Conservation Service PLANTS Database (USDA NRCS, 2014). There exists a PLANTS taxonomic record for each currently accepted scientific name and for obsolete names. Each record contains both a "symbol" field and an "accepted symbol" field. For records representing currently accepted taxa, the two fields contain the same symbol. For obsolete taxa, the symbol field represents the obsolete name and the "accepted symbol" field equals the "symbol" field of the currently accepted name. This structure allows for consistent translation between and treatment of synonyms.

SALVIAS

The Synthesis and Analysis of Local Vegetation Inventories Across Scales (SALVIAS, http://www.salvias.net) is a collection of global vegetation inventory data. Although some data are from sites within the United States, the wetland/upland status of plot records are not recorded making it unsuitable for this research.

VegBank

VegBank (http://vegbank.org/) is a vegetation plot data repository created by the created Ecological Society of America's Panel on Vegetation Classification. VegBank contains 75,084 plot records from 105 research projects. The number of plots per project varies from one to over 17,000. Approximately half of the plot records include a description of hydrologic regime, providing a potential means of separating wetland and upland plots. It should be noted that there is a great deal of overlap between VegBank and several other sources. At least some or all of the records from the Carolina Vegetation Survey Database, the NPS, and possibly others, are contained in VegBank. If VegBank were to be added to a combined database, care would need to be taken to avoid duplicate records. Also, VegBank is comprised of data from 105 research projects, each with its own sampling design. Therefore, it is not likely possible to treat the records as a single data source.

NPS

The NPS data are derived from an ongoing effort to characterize and map vegetation communities throughout National Park Service lands. These data are treated in detail in subsequent sections of this report.

BISON

The USGS Biodiversity Information Serving Our Nation (BISON; http://bison.usgs.ornl.gov) database is a collection of species occurrence data from over 300 sources such as museum collections and herbarium records. Each record contains latitude and longitude coordinates, however these coordinates are of variable precision. For example, coordinates may represent the centroid of a county, as in the case of USDA PLANTS records. BISON does not make available any data related to the upland or wetland status of records. Therefore, BISON data are not directly usable in this research. However, the BISON search function may prove useful for locating new sources of data in the future.

FIA

The GIVD refers to two related data sources: the USFS Forest Inventory and Analysis Database of the United States of America and the FIADB Vegetation Diversity and Structure Indicator. Both of these sources are contained in the USFS FIA Database and are henceforth collectively abbreviated FIA. FIA consists of a continuous inventory of permanent plots for both tree species and non-tree vegetation for forest lands throughout the United States. Non-tree vegetation is a relatively recent addition to FIA and the published data for most states contained primarily tree species data at the start of this research. FIA data are treated in detail in subsequent sections of this report.

Table 2: Summary of data source characteristics relative to research requirements.

Database	National scope	Single database or a collection of databases	Single uniform plot record format	Statistical sampling design	Plot wetland / upland status recorded	Normalized taxonomy
SALVIAS	no	collection			no	
Alaska Arctic Vegetation Archive[1]	no	collection				
LTER	yes	collection	no			
Carolina Vegetation Survey Database	no	collection		no		
Plant Community Survey and Resurvey Data from the Wisconsin Plant Ecology Laboratory	no	collection				
Natural Heritage Vegetation Database for West Virginia	no					
VegBank	yes	collection	no[2]	no	variable	yes, USDA PLANTS
USGS Biodiversity Information Serving Our Nation (BISON)	yes	collection	yes	no	no	yes
NPS	yes	single	no[2]	no	variable	variable, mostly USDA PLANTS
FIA	yes	single	yes	yes	yes	USDA PLANTS

[1] The Alaska Arctic Vegetation Archive is a work in progress (Breen *et al.*, 2013) and as of this writing, data are not yet available for download.

[2] VegBank and NPS data are available in a uniform record format, but not all records contain data in all fields.

A very general description of the process of extracting wetland plant ratings from an occurrence database is as follows; acquire an understanding of the structure of the database, especially the relationship between tables; join the tables; apply aggregate functions to extract count data; and apply algebraic functions to the aggregate data. The specific programming code required to complete this process is dependent on the structure of the database and so varies between databases. The amount of effort required to process a database proportional to its complexity and as also impacted by the quality and consistency of the data. Conversely, the amount of effort has relatively little to do with to total number of records in the database.

Consequently, with all other factors being equal, databases with larger numbers of records and species and covering larger geographic areas offer a much greater return on investment than smaller more geographically limited databases. After reviewing the characteristics of each data source, two were selected as having the greatest potential to produce usable estimates of frequency of occurrence in wetlands: NPS and FIA.

Part 3. - Normalize data (objective 2)

General Approach

All data used in this research were imported into a PostgreSQL database server. The PostGIS extension to PostgreSQL adds Geographic Information System (GIS) capabilities to the database server. PostGIS was used to create a USACOE Region GIS layer, to convert non-spatial vegetation plot data to GIS layers, and to link each plot to the appropriate region.

Computer programs were created to automate the download, processing, and analysis of each data source (see Appendices). The programs are written in the Python 3 and Structured Query Language (SQL) languages and also make subprocess calls to the PostgreSQL client program (psql). The programs and software are generally platform agnostic, but have only been tested on a Debian Linux system. There are three Python programs; one for each type of data, i.e., ancillary , NPS, and FIA data. After PostgreSQL, PostGIS, and Python 3 are installed (see Appendix 1), the programs can construct and populate a database and calculate wetland indicator status ratings in less than four hours on a capable modern multi-core desktop PC.

Ancillary Data

Six data tables common to the analyses of all data sources in this research are: 1) “regions” - a table to translate between USACE Region names, abbreviations, and a numeric code for each region; 2) “nwpl_2013” - the current (as of 2013) National Wetlands Plant List; 3) “mlra_v42” - a GIS layer and table of USACE Regions; 4) “usda_plants” - the complete USDA Plants database of current species names and synonyms; 5) “statep010” - a GIS layer and table of state boundaries; and 6) “states_fips” - a table to translate between state names, abbreviations, and Federal Information Processing Standard (FIPS) codes. A Python program (see Appendix 2) was developed to create the database structure, download, import, and process the data for these tables. SQL programs called by the Python program are documented in subsequent appendices (see Appendices 3 through 9).

The first table (Appendix 3) serves as a dictionary to translate between full USACE Region names, abbreviated names, and an integer numeric code for each region. The numeric code allows the database to store region information efficiently. For example, to record the region for the over 1.3 million FIA plot records, it requires significantly less space and processing power to store the region as an integer code rather than the full text of the region name. The full name or abbreviation can then be retrieved only as needed by linking to this table.

The second table (Appendix 8) contains the data from the current (2013) NWPL Excel spreadsheet. This table allows a side by side comparison of the NWPL status ratings with the calculated ratings based on FIA and NPS data.

The third table (Appendix 4) is a spatial representation (GIS layer) of the USACE Regions. A GIS intersection operation between this layer and, e.g., a layer of vegetation plots allows the labeling of each plot by the region it exists within. This table was built for this research effort by modifying a GIS layer of U.S. Department of Agriculture (USDA) Natural Resources Conservation Service (NRCS) Land Resource Region (LRR) and Major Land Resource Area (MLRA) polygons. The USACE Regions are largely defined by a combination of State, LRR, and MLRA (see http://plants.usda.gov/wetinfo.html). However, there are some regions whose definitions include additional qualifications based on elevation. This research defines USACE Region as accurately as possible using only state, LRR, and MLR information. See the discussion for further information on this point.

The fourth table (see Appendices 5 through 7) is a direct import of the complete USDA Plants database of current species names, synonyms, and relevant taxonomic fields. This table allows the grouping and normalization of NPS data by taxonomy. For example, wetland status codes need to be calculated per species. NPS records record the species present on a plot variously using some combination of three fields: USDA PLANTS symbol, scientific name, and/or Integrated Taxonomic Information System (http://www.itis.gov/) Taxonomic Serial Number (ITIS TSN). The values in these three fields for a given record are then linked (see subsequent section on NPS processing), in a specified order of preference, to a single USDA PLANTS record representing the currently accepted scientific name for a species.

The fifth table (see Appendix 2) is a GIS layer and table of state boundaries sourced from the National Atlas of the United States (2012). These boundaries allow for the linking of NPS plots data to states using GIS and based on plot coordinates or park boundaries.

The sixth table (Appendix 9) allows translation between state names, abbreviations, and Federal Information Processing Standard (FIPS) codes. This table was sourced from the United States Cenus Bureau (2014). Translation is necessary because some data sources use FIPS codes to record state while others use two letter abbreviations. The fifth and sixth tables are used as part of the process to link plots with USACOE region in cases where plot location is ambiguous or erroneous due to missing or erroneous plot coordinates or recorded plot state.

NPS Data

The USGS-NPS Vegetation Mapping and Inventory Program is an ongoing effort to characterize the vegetation communities of the over 270 NPS properties. The effort has spanned over twenty years and is approximately 56% complete to date. The program guidelines specify that vegetation plot data be acquired and stored in a database template called PLOTs. These guidelines and preliminary examination of NPS data appeared to satisfy the requirement that data be of consistent format for a given source.

In fact, the PLOTs database design has evolved over the lifetime of the project and the data format for each park at least partially depends upon the year in which it was completed. Additionally, even parks completed at similar times exhibit some differences in data format. This variability impacts efforts to combine vegetation plot data from multiple parks into a single database.

The field data for each NPS property is typically stored in Microsoft Access database conforming to the PLOTs template from the time of its creation, and as two Microsoft Excel spreadsheets; one containing descriptive data for plot characteristics, and one containing a list of species found on each plot. The Access database structure was found to be much more variable between parks, so the Excel files were chosen for this research.

These file are available from two sources: the USGS (http://www.usgs.gov/core_science_systems/csas/vip/themes/fielddata.html); and the NPS (https://irma.nps.gov/App/Reference/Profile?Code=2097270). The files are mostly, but not always, identical at each source. For a given property, newer files and files that generated fewer errors during processing were preferred. This preference required a great deal of manual decision making and the files are mostly static; therefore the download portion of this process was not automated.

A Python program (Appendix 10) and SQL programs were created to automate the processing of the Excel files . The following outline summaries the general tasks completed by the Python program and its related SQL programs (see Appendix 11 through Appendix 14).

1. Create local database structure (Appendix 11) and functions for processing and analyzing data.
2. Copy data from Excel files to PostgreSQL database.
3. Download and copy NPS park boundaries GIS layer.
4. Run processing functions.
   (a) Process (Appendix 12) – correct errors and normalize data.
   (b) Geography (Appendix 13) – add spatial columns to data.
   (c) Summaries (Appendix 14) – calculate ratings and summarize data.

Processing NPS Data

For most parks, the vegetation data were stored in two Excel spreadsheets: one containing plot data and one containing plant species data. Several parks used completely different methods to record vegetation data; methods either incompatible with the PLOTS standard or lacking necessary fields, e.g., the ability to distinguish wetland plots from upland plots. These parks were necessarily excluded from analysis. A few pairs of parks are managed as a single unit and so data files for these parks contain records for multiple properties. 107 pairs of Excel files were processed.

Plot records and species records are linked via a unique code for each plot or plot visit ("event code"). Individual parks are referred to using a four letter code, e.g., "waca" refers to Walnut Canyon National Monument. A list of parks and their codes can be found at http://www.usgs.gov/core_science_systems/csas/vip/parks.html. A number of corrections and adjustments to both plots and species records were required (Table 3) in order to combine park data files into a single database. These changes and the lack of statistical sampling design for locating plots may affect the confidence with which these data can be used for determining wetland status ratings. Therefore, a summary of the changes is presented here. The exact details of adjustments can be found in the relevant SQL program (see Appendix 12).

Table 3: Summary of corrections and adjustments made to NPS data.

Table	Correction or adjustment	Rows affected
Species	1. Set invalid ITIS TSN code ("999999999999.0") to null.	9
Species	2. Drop zero decimal from ITIS TSN code (should be an integer).	235,992
Species	3. Remove negative sign prefix from ITIS TSN code.	3,361
Plots	4. Normalize Cowardin and "community type" case (set to all lower case).	25,275
Plots	5. Normalize Cowardin values for parks waca and wupa (convert numeric code to text).	326
Plots	6. Normalize Cowardin values for parks chcu and nava (convert numeric code to text).	251
Plots	7. Normalize Cowardin values for parks scbl and pinn (convert numeric code to text).	81
Plots	8. Delete rows missing both plot code and event code.	2
Species	9. Delete rows missing both plot code and event code.	65,365
Species	10. Delete rows outside of plots.	19,763

Species	11. Correct inconsistent code/event usage. Move corrected event code to plot code and delete event code.	132,331
Species	12. Update miscoded codes and typographical errors.	445
Plots	13. Correct corrupt codes treated as numbers by Excel.	96
Plots	14. Correct miscoded event (typographical errors).	1
Plots	15. Delete rows that are duplicate in every field.	6,541
Plots	16. Delete rows duplicate except in shape. Target copies having shape = "N/A".	204
Species	17. Delete rows duplicate except in “source” column.	100,362
Species	18. Delete rows duplicate except in “used plants” column.	16
Species	19. Duplicate except in “family” column. Prefer copies with non-null family.	2
Species	20. Delete rows duplicate except in “common name” column. Prefer copies with non-null common name.	20
Plots	21. Delete rows with no match in species table.	2,001
Species	22. Delete rows with no match in plots table.	22,831

The ITIS TSN code for a species is a positive integer. Corrections 1 through 3 (Table 3) are designed to make the recorded codes conform to this standard. The decimal and sign problems may have been introduced by Excel automatically detecting the code as a number.

Corrections 4 through 7 are to normalize the values in the Cowardin classification code (Cowardin *et al.*, 1979) and “community type” fields. Parks record these fields variously in upper case, lower case, or mixed case. Also, several parks (steps 5 through 7) record the Cowardin code as a number, with different parks using different coding schemes. These numbers were converted to text to match the system used by most other parks.

Steps 8 and 9 remove plots and species rows that lack plot and event codes. These types of plot rows cannot be linked to any species rows and *vice versa*. Many of these rows are entirely empty rows that pad the end of several Excel files. Other rows are missing the code values in the Excel files but not in the corresponding Access files. These rows may be recoverable with additional programming effort.

Some vegetation rows represent data observed outside of plot boundaries. The wetland or upland status of these data are not recorded and they are not included in the measures of plot area. Step 10 removes these rows since they cannot be used for this research.

Steps 11 through 14 correct various plot and event code errors, e.g. plot codes incorrectly placed in the event codes column and incorrect punctuation.

Steps 15 through 20 remove duplicate plot and species records. Some duplicate species records are not errors, but represent species found in multiple vertical layers on a plot. However they are duplicate with respect to this research.

Steps 20 and 21 remove plots with no match in the species table and *vice versa* respectively. After these adjustments, there remains 16,527 plot records and 400,040 species records.

In addition to these corrections of data values, adjustments were made to column titles in order to combine data from parks that used different text to represent the same column title. For example, the plant symbol code column in the species table was variously called “plantsymbol”, “plantssymbol”, “plantnames”, “plantcode”, “plantscode”, and “sppcode”. A dictionary of synonyms was constructed in the Python program (Appendix 10) for every field in both tables. Upon processing each Excel file, the Python program scans column headings for matches in its this dictionary and converts them to a common name.

Adding Geography to NPS data

In order to analyze the vegetation data by USACE Region and state (for the Arid West versus California analysis), the location of each plot row must be linked the appropriate region and state via GIS. Each plot row contains several columns related to spatial position, e.g. x and y Universal Transverse Mercator (UTM) coordinates, latitude and longitude, and Geographic Positioning System (GPS) datum. Which columns contain data varies both between parks and within parks as does the GPS datum and the method of recording the spatial data. To minimize the potential for error caused by this variability, the location of each park with respect to regions and states is considered along with plot coordinate data to help identify and correct plot coordinate errors.

The SQL program (Appendix 13) compares individual plot locations to state and USACE Region polygons, using plot latitude and longitude if available, or UTM coordinates if not. The program then updates these plot records with the appropriate region and state codes.

558 of 16,527 plots could not be linked to a region or state because of missing coordinate data. In these cases the program falls back to comparing the GIS boundary data for parks (see Ancillary Data) to the GIS data for USACE Regions and states and identifies the majority region and state in each park. In subsequent analysis, the region and state found via plot coordinates is preferred over the assumed region and state based on park boundaries (for details see Appendix 14). A number of calculations and corrections to plot spatial data were made to allow GIS analyses to take place (Table 4).

Table 4: Summary of calculations and adjustments relating to NPS plot spatial data.

Calculation or correction		Rows affected
1.	Add geography related columns to plots table	

2.	Create/replace parks_regions table and populate with intersection of park boundaries and regions.	
3.	Add primary key to parks_regions (park_code, region_cd).	
4.	Update plots with majority region of each.	all
5.	Create/replace parks_states table and populate with intersection of park boundaries and states.	
6.	Add primary key to parks_states (park_code, state_fips)	
7.	Update plots with state_fips for majority state of each park.	all
8.	Normalize case on geographic fields.	16,527
9.	Correct letter "o" to number zero in utm coordinate fields.	1
10.	Normalize geodetic datum values, e.g. "NAD 1983" and "NAD 83" equal "NAD83".	16,527
11.	Update invalid ("(N/A)") geodetic datum to null.	51
12.	Update utm coordinate pairs to null if either coordinate is invalid.	1,456
13.	Update "corrected" utm coordinate pairs to null if either coordinate is invalid.	474
14.	Manually correct missing utm zones.	1,460
15.	Numerous manual corrections to UTM and lat/lon coordinate fields, e.g. reversed coordinates, order of magnitude errors, etc.	
16.	Update lat/lon in DMS to decimal degrees.	29
17.	Copy lat/lon to geom and make longitude negative.	345
18.	Copy utm to geom and set EPSG code based on declared datum.	15,624
19.	Set region to region_gid from mrla polygons.	15,469
20.	Catch points outside of mlra polygons, e.g. near coastlines, islands. Only null lat/lon remain.	500
21.	Set park_code_bnd for plots covered by park boundary polygons.	13,492
22.	Set park_cd_bnd to nearest park for plots NOT covered by park boundary polygons.	2,477
23.	Set state_cd to fips code from covering statep010 polygons.	15,615
24.	Set state_cd to nearest state for plots NOT covered by statep010 polygons, e.g. near coastlines, islands.	354

Summarizing NPS data

The calculation of wetland status code from the NPS vegetation data requires the ability to group and summarize the data first by USACE Region (enabled by the previous steps), then by species, then by the wetland/upland condition of plots. Grouping and summarizing by species and wetland/upland condition are enabled by an SQL program (see Appendix 14).

Species are identified in these data using one or more of the following columns: USDA PLANTS symbol, ITIS TSN, and scientific name. As with other column data, the use of each of these columns is inconsistent both between and within parks. To make these data as consistent as possible, the SQL program attempts to link each species record to a USDA PLANT currently accepted scientific name. It retrieves that name first by examining the NPS symbol column and converting it to a USDA PLANTS current accepted symbol the symbol is found to be a synonym. If that attempt fails, a similar process is attempted using the ITIS TSN code. The final fallback is to use the scientific name as recorded in the NPS data.

This fallback was used for approximately 9% (35,121) of species records A large percentage of these records include text in the scientific name column that does not match the scientific name derived from USDA PLANTS. Examples of this type of text include: inconsistent presence, absence, or format of species authorship; family or genus where species could not be identified, numbers indicating unidentified species, e.g. "*Carex* 2"; guild, e.g. "forb"; and notes about the plant, e.g. "seedling". Since grouping is based on an exact match of species names, these records fall outside of the groups whose scientific names derive directly from or match exactly the USDA PLANTS data.

The final grouping of data is the plot condition, i.e. wetland or upland. Several plot table columns were examined for there potential use in discriminating wetland from upland: community type (i.e. wetland or upland), Cowardin classification, hydrologic regime, hydrologic evidence, and soil drainage. Again, the values in these columns were inconsistent between and within parks. During discussions with the National Technical Committee for Wetland Vegetation, no consistent and completely reliable method of equating these fields to a legal wetland delineation was identified. For the purposes of this analysis, the following logic was used to label plot condition.

If the community type equals "wetland" or the Cowardin classification is one of "marine", "estuarine", or "palustrine", then the plot is considered wetland. If the community type or the Cowardin classification equals "upland", then the plot is considered upland. If neither condition is met, e.g. due to missing data, then the plot condition is labeled "N/A". The number of plots in each category are: N/A, 2,460 (15%); upland;11,991 (73%); and wetland, 2,076 (13%).

FIA Data

The USFS Forest Inventory and Analysis produces a continuously updated database containing approximately 16 million tree records and 1.3 million plot records involving about 400 tree species. About 614,000 plot records occur on what FIA calls "accessible forest land" and therefore include inventory data usable for this research. Each plot record represents a visit to a plot, not a unique plot.

The database covers the entire United States and data are available on a per state basis except for Hawaii. The database structure is comprised of 50 tables for inventory data and 18 reference data tables. Downloading the entire database by state involves a total of 2,568 data files. The files are usually updated several times a month (see http://apps.fs.fed.us/fiadb-downloads/images/recent_load_history.html). To process and update this number of files in a reasonable amount of time necessitated automation of all steps.

The automation was accomplished by constructing a Python program (Appendix 16) that in turn called several SQL programs and the PostgreSQL client utility (psql). An overview of the steps completed by the program to processing FIA data is as follows. There are no steps required to correct or normalize data since they originated from a database with necessary constraints in place, e.g. numbers are stored in numeric fields that preclude the presence of non-numeric data.

1. Download FIA files.
   (a) Download the reference archive files and the state archive files (one per state) only if they are new than the local copies.
   (b) Extract the archive files.
2. Create local FIA database structure on the PostgreSQL server.
   (a) Connect to the local PostgreSQL server, NWPL database.
   (b) Create or replace the FIA schema including table structure (Appendix 17), views, and processing and analysis functions.
3. Copy the FIA state files to the local database.
4. Copy the FIA reference files to the local database.
5. Run processing and analysis functions on local database.
   (a) Add geographic data the database, i.e. use PostGIS and plot coordinates to make the data spatial (Appendix 18).
   (b) Calculate the wetland ratings via a series of cascading views (Appendices 19 and 20).

Adding Geography to FIA data

The latitude and longitude columns for each FIA plot were used converted transform the plot table into a GIS layer. Each plot row was then linked to the appropriate region via a GIS intersection. Some plots fell outside of region polygons due to the imprecision of the region layer with regards to coastlines and islands. In these cases, plots were linked to the nearest region.

Summarizing FIA data

As with NPS data, the calculation of wetland status code from the FIA data requires the ability to group data by region, then species, then the wetland/upland condition of each plot. FIA handles taxonomy consistently by storing genus, species, variety, and subspecies in separate columns. These columns were concatenated into a single "taxon" column allowing analysis at the finest level of taxonomy available. All taxonomic fields are preserved, however, allowing subspecies and varieties to be collapsed into species if desired.

Three fields were identified as having some potential in identifying wetland and upland plot conditions: topographic position (available for Pacific Northwest Research Station data only); "present non-forest code", i.e. the type(s) of non-forest land cover found on a plot, and physiographic class code. Physiographic class codes are grouped into hydric, mesic, and xeric codes. The FIA database can record multiple conditions on a single plot, and the area of each condition. As with the NPS data, no exact and completely reliable method of equating these fields to a legal wetland delineation was identified during discussions with the National Technical Committee for Wetland Vegetation. For the purposes of this analysis, plot conditions were categorized using the following logic.

Plot conditions with hydric codes are considered wetland, all non-hydric codes are upland, and missing codes are labeled “N/A”. The number of usable plot records in each category are: wetland, 38,545 (6%); upland, 480,981 (78%); and, “N/A” 94,066 15%.

Part 4. - Develop algorithms for frequency calculations (objective 3)

For a field survey involving a simple random sample of sites, the presence or absence of a plant species in wetlands or uplands can be expressed as a contingency table.

Table 5: Contingency table for the presence or absence of a plant species in wetland and upland sites.

	wetland	upland	
present	n_{pw}	n_{pu}	$n_{p\bullet}$
absent	n_{aw}	n_{au}	$n_{a\bullet}$
	$n_{\bullet w}$	$n_{\bullet u}$	N

For a species that occurs in both wetlands and uplands, the frequency of occurrence in wetland sites relative to all occurrences (Equation 1) is influenced by the relative proportion (Equation 2) of wetlands in the landscape (Lichvar and Minkin, 2008).

$$\frac{n_{pw}}{n_{p\bullet}} = \frac{n_{pw}}{n_{pw} + n_{pu}} \qquad \text{(Equation 1)}$$

$$\frac{n_{\bullet w}}{N} \qquad \text{(Equation 2)}$$

For example, Equation 1 is biased whenever wetlands and uplands comprise an unequal proportion of the landscape. The bias can be corrected by multiplying n_{pw} by the proportion of uplands relative to wetlands across the landscape (Equation 3).

$$\frac{n_{\bullet u}}{n_{\bullet w}} \qquad \text{(Equation 3)}$$

$$\frac{n_{pw} \cdot \frac{n_{\bullet u}}{n_{\bullet w}}}{n_{pw} \cdot \frac{n_{\bullet u}}{n_{\bullet w}} + n_{pu}} \qquad \text{(Equation 4)}$$

Equation 4 can be simplified to Equation 5.

$$\frac{n_{pw} \cdot \frac{n_{\bullet u}}{n_{\bullet w}}}{n_{pw} \cdot \frac{n_{\bullet u}}{n_{\bullet w}} + n_{pu}} \cdot \frac{\frac{n_{\bullet w}}{n_{\bullet u}}}{\frac{n_{\bullet w}}{n_{\bullet u}}}$$

$$= \frac{n_{pw}}{n_{pw} + \frac{n_{pu} \cdot n_{\bullet w}}{n_{\bullet u}}}$$

$$= \frac{n_{pw}}{n_{pw} + \frac{n_{pu} \cdot n_{\bullet w}}{n_{\bullet u}}} \cdot \frac{\frac{1}{n_{pw}}}{\frac{1}{n_{pw}}}$$

$$= \frac{1}{1 + \frac{n_{pu} \cdot n_{\bullet w}}{n_{pw} \cdot n_{\bullet u}}} \qquad \text{(Equation 5)}$$

In this research, equation 1 is referred to as the “unadjusted frequency of occurrence” and equation 5 as the “adjusted frequency of occurrence”. It is unclear at this time which, if either, of these equations best relate to the concepts behind the wetland indicator status. Results from both equations are provided for NPS and FIA data. Unadjusted and adjusted frequencies were converted to wetland indicator status using the following logic.

The contingency table variables comprise a theoretical representation of a field sampling design using a simple random sample of points in space. In the FIA and NPS analyses, the closest representation of the contingency table variables was constructed. For example, FIA is designed to produce estimates of number of individual trees but NPS contains only presence and absence. The notation n_{pw} is retained in both cases in this research.

In the theoretical example, the ratio of uplands to wetlands in the landscape is represented by numbers of points in each case (Equation 3). In both FIA and NPS, two dimensional plots, not points, are used. The equivalent ratio in the case of FIA is the ratio of the total area of upland plot components to wetland plot components. In NPS, entire plots are labeled as wetland or upland so the equivalent value is the ratio of the total area of upland plots to the total area of wetland plots.

The calculation of this ration for NPS plots is further complicated by the fact that plot size and shape are inconsistently used and inconsistently and often ambiguously and erroneously recorded. For example, for some plots, the plot shape (e.g. circle or rectangle) does not agree with the type of dimensions provided (radius or diameter as opposed to length and width). For approximately twelve percent of the plots, no plot area calculation was possible so a small area (one meter squared) was assumed in order to count the species records but give those plots minimal influence on the area ratios. The exact methods for calculation of plot area and the handling of errors is documented in Appendix 14.

Table 6: The relationship between frequency of occurrence and wetland indicator status.

Frequency (x)	Wetland indicator status	Wetland indicator status abbreviation	Wetland indicator status rank
$x >= 0.99$	obligate wetland	OBL	5
$0.67 <= x < 0.99$	facultative wetland	FACW	4
$0.33 <= x < 0.67$	facultative	FAC	3
$0.01 < x < 0.33$	facultative upland	FACU	2
$x <= 0.01$	obligate upland	UPL	1
N/A	"?"	"?"	NULL

The wetland indicator status rank is used for sorting and to provide a numeric value for quantitatively measuring the variability of a species rank between regions. The "NULL" and "?" values in each column allow a convenient method of marking and excluding data whose frequency values cannot be calculated.

Part of objective 3 was to develop confidence intervals for frequency calculations. The NPS plot data do not follow any statistical sampling design therefore it is not possible to calculate valid confidence intervals. While recognizing the limitations of these data, some estimate of the error inherent in the sample sizes found in the NPS data is useful. For proportional data, the margin of error at the 95% confidence level (Equation 6) for a given sample size (n) is maximized at a proportion (p) of 0.5.

For each species, the maximum margin of error (Equation 7) was calculated for a sample size n, where n is the total number of wetland and upland plots on which the species was found. This margin of error does not apply to the entire wetland frequency value because it does not include the sampling error from the ratio of upland to wetland plots in each region. However, the contribution of that error is relatively small given the large total number of plots and that all plots contribute to that ratio.

$$1.96\sqrt{\frac{p(1-p)}{n}} \qquad \text{(Equation 6)}$$

$$.98\sqrt{n} \qquad \text{(Equation 7)}$$

Confidence intervals algorithms for FIA data were to be based on a modified version of the SQL program found on the USFS FIA EVALIDator website (http://apps.fs.fed.us/Evalidator/evalidator.jsp). However, as of March 11, 2014, the website indicates that there is a problem in the calculation of sampling error estimates that renders them invalid. The error is expected to be corrected by late April, 2014.

Part 5. Analyze wetland indicator status ratings (objectives 4, 5, 6)

Objectives 4 through 6 were met by reporting calculations in several spreadsheets files (Table 7) that are designed to accompany this report. The large number of columns necessitated an abbreviated naming scheme. To aid in interpreting the spreadsheets, a key to column naming schema is provided here (Table 8).

Table 7: A description of the summary data provided as spreadsheet files.

Spreadsheet filename	Description
nps_region.xls	Indicator status ratings for NPS data, grouped and sorted by taxon then region.
nps_compare_nwpl.xls	Indicator status ratings for NPS data and NWPL 2013 data, grouped and sorted by taxon then region.
nps_arid_west_compare_ca.xls	Compare indicator status ratings for NPS data between CA and the rest of the Arid West region (objective 6).
fia_region.xls	Indicator status ratings for FIA data, grouped and sorted by taxon then region.
fia_state.xls	Indicator status ratings for FIA data, grouped and sorted by taxon then state.
fia_compare_nwpl.xls	Indicator status ratings for FIA data and NWPL 2013 data, grouped and sorted by taxon then region.
fia_arid_west_compare_ca.xls	Compare indicator status ratings for FIA data between CA and the rest of the Arid West region (objective 6).
combined_region.xls	Indicator status ratings for species found in both NPS and FIA data, grouped and sorted by taxon then region.
combined_compare_nwpl.xls	Indicator status ratings for NWPL 2013 data compared with ratings for species found in both NPS and FIA data, grouped and sorted by taxon then region.

Unadjusted and adjusted frequencies and respective indicator statuses were calculated for both NPS and FIA. To complete objective 4, the assessment of indicator status ratings, these values were placed side by side with the current (2013) NWPL ratings by matching on scientific name.

For species found in both NPS and FIA, a combined indicator status was calculated based on the mean of the frequencies from each source. This method weighs each source equally which does not account for different sample sizes and methods between each source. However, since FIA and NPS deal with different types of count data (number of individuals versus presence or absence respectively) this method was determined to be most straightforward.

Objective 5 is to identify species with highly variable ratings across regions. Indicator status ratings can be considered interval data in the sense that they represent discrete sections of a continuous scale (frequency of occurrence). However, in the sense that the intervals are asymmetrical, indicator status could be considered ordinal. In light of this uncertainty, the variability of ratings are represented by two metrics: the rating range (maximum status rank minus minimum status rank), and the rating variance (actually the variance of the rank numeric field). These two metrics are calculated for both unadjusted and adjusted status ratings. The identification of highly variable species can be achieved by sorting a spreadsheet by the desired range or variance column.

Objective 6 is to compare ratings between California and the rest of the Arid West region. This objective was accomplished by placing California and Arid West data side by side, matching on scientific name. All California and Arid West species were included, i.e., even species found in one but not both sets.

Table 8: A key to spreadsheet column names.

Column name(s)	Description
taxon	scientific name; may be prefixed with source, e.g. "fia_" or "nwpl_"
species_symbol	USDA PLANTS symbol
common_name	species common name; may be prefixed with source, e.g. "fia_" or "nwpl_"
[source]_[region code]_[adj/unadj]	indicator status, e.g. fia_aw_adj is indicator status based on adjusted frequency for FIA data from the Arid West
[source]_[adj/unadj]_[variance/range]	variance or range of the the indicator status rank
[region code]_max_moe_at[90 or 95]cl	maximum margin of error (see Part 4 of this report) for the species frequencies in the given region at the given confidence level (90% or 95%)
wetland_ind_[adj/unadj]_rank_[range/variance]	range and variance of the species' wetland indicator status rank across regions; higher numbers identify species with more highly variable ratings across regions (Objective 5)
npw, npu, npother, ndotw, ndotu, ndotother	$n_{pw}, n_{pu}, n_{pother}, n_{\bullet w}, n_{\bullet u}, n_{\bullet other}$ (see Part 4 of this report); the "n[p/u]other" columns refer to the number of records that could not be classified as wetland or upland, e.g. due to missing data in the physiographic class code column.

Part 6. - Discussion and Future Research

The ideal data source for this research would be a large simple random sample comprising of vegetation plots on which a wetland delineation was performed and distributed throughout the United States. The minimum qualifications for a useful data source might be a statewide collection of vegetation plots following a valid statistical sampling design and containing enough information on plot condition to confidently infer the wetland or upland status of the plot.

Given the per-source effort demonstrated here to calculate wetland indicator status, it would likely be extremely time consuming to assemble a large collection of small scale data sources. Given the near nation-wide coverage of NPS and FIA, these two sources were determined to be the most likely to return useful results.

Limitations of NPS and FIA

The lack of statistical sampling design in NPS and other potential data sources precludes valid statistical inference from those data. Some assessment of the contribution of sample size to sampling error is included in this research for the NPS data. However, any statistical bias is indiscoverable in the NPS data as published. The FIA data do follow a sampling design

(Bechtold and Patterson, 2005) and so allow statistical inference and related metrics, e.g. confidence intervals.

The statistical universe of NPS and FIA are National Park Service lands and forested lands respectively. It is possible that the data are not representative of conditions outside these settings. A pilot study, e.g. a comparison of FIA plot data with non-forest data, may be able to assess this possibility.

Neither FIA nor NPS define plot condition as wetland or upland in a manner that is equivalent to a wetland delineation. It is possible that plot condition is systematically biased in one direction, e.g. by missing ephemeral wetlands, or contributes to random measurement error, or both. Again, a pilot study may be able to estimate the relationship between recorded plot condition with an actual delineation.

The NPS data have significant problems regarding formating, consistency, duplicate and missing records, and typographical errors. Many of these errors have been corrected in order to produce wetlands status indicator ratings, but they may impact the reliability of these calculations. A detailed treatment of each error and method of correction can be found in the SQL program created to process NPS data (Appendix 12).

Potential Improvements

USACE Regions are defined mostly in terms of NRCS MRLAs and LRRs except for certain caveats with respect to the elevation of the ponderosa pine zone in three regions (the Arid West, Great Plains, and the Western Mountains Valleys, and Coast). There is no means of translating these caveats using only MRLA data and so they were ignored when converting MRLA polygons to USACOE Regions. It is possible that this conversion could be improved with ancillary data such as elevation or a GIS layer relating to ponderosa pine distribution.

Thirty states in the FIA database now have some data in the non-tree vegetation tables. Some new code would need to be developed to produce wetlands rating for these data since they are stored in different tables than tree species. However, much of the basic statistics and programming procedures created for this research could be reused with modest changes. This effort could greatly increase the number of species covered by these analyses.

Currently, FIA data are downloaded and processed via one archive file per state, with each archive file containing fifty files; one file per FIA table. FIA data are now available as a single archive containing one file per table. Switching the Python program to this new data format would significantly reduce the complexity of the program and may reduce the processing time.

References

Bechtold, W.A., Patterson, P.L., 2005. *The enhanced forest inventory and analysis program: national sampling design and estimation procedures. General Technical Report SRS–80*. US Department of Agriculture Forest Service, Southern Research Station Asheville, North Carolina.

Boyle, M.F., Peet, R.K., Wentworth, T.R., Schafale, M.P., 2007. *Natural vegetation of the Carolinas: Classification and description of plant communities of the Francis Marion National Forest and vicinity. A report prepared for the Ecosystem Enhancement Program, North Carolina Department of Environment and Natural Resources in partial fulfillments of contract D07042.*. Carolina Vegetation Survey, Curriculum in Ecology, University of North Carolina, Chapel Hill, NC.

Breen, A.L., Raynolds, M.K., Hennekans, S., Walker, M.D., Walker, D.A., 2013. *Toward an Alaska prototype for the Arctic Vegetation Archive*. In: Walker, D.A., Breen, A.L., Raynolds, M.K., Walker, M.D. (Ed.), *Arctic Vegetation Archive (AVA) Workshop, Krakow, Poland. CAFF Proceedings Report 10. Akureyri, Iceland. ISBN: 978-9935-431-24-0*.

Cowardin, L.M., Carter, V., Golet, F.C., LaRoe, E.T., 1979. *Classification of wetlands and deepwater habitats of the United States*. U.S. Department of the Interior, Fish and Wildlife Service, Washington, D.C..

Dengler, J., Jansen, F., Glöckler, F., Peet, R.K., De Cáceres, M., Chytrý, M., Ewald, J., Oldeland, J., Lopez-Gonzalez, G., Finckh, M., Mucina, L., Rodwell, J.S., Schaminée, J.H.J., Spencer, N., 2011. The Global Index of Vegetation-Plot Databases (GIVD): a new resource for vegetation science. Journal of Vegetation Science 22, 582-597.

Lichvar, R.W., Minkin, P., 2008. *Concepts and Procedures for Updating the National Wetland Plant List. ERDC/CRREL TN-08-3*. US Army Corps of Engineers.

Lubinski, S., Hop, K., Gawler, S., Story, M., Brown, K., 2003. *Acadia National Park, Maine Project Report*. U.S. Geological Survey-National Park Service Vegetation Mapping Program.

National Atlas of the United States, 2012. *1:1,000,000-Scale State Boundaries of the United States*. (http://nationalatlas.gov/atlasftp-1m.html). National Atlas of the United States, Rolla, MO..

United States Cenus Bureau, 2014. *American National Standards Institute (ANSI) and Geographic Names Information System Identifier (GNISID) Codes for States, the District of Columbia, Puerto Rico, and the Insular Areas of the United States*. (http://www.census.gov/geo/reference/docs/state.txt http://www.census.gov/geo/reference/ansi_statetables.html). .

USDA NRCS, 2014. *The PLANTS Database*. (http://plants.usda.gov). National Plant Data Team, Greensboro, NC 27401-4901 USA.

Appendix 1 – Setup instructions for PostgreSQL server

filename:PostgreSQL_setup.txt

1) Install required software and dependencies: PostgreSQL Server v9.3; PostGIS 2.1; Python 3; psycopg2 database adaptor for Python 3; xlrd Python 3 library to extract data from Excel spreadsheets; Geospatial Data Abstraction Library (gdal) utility programs. In the Debian Linux testing distribution:

```
:~$ sudo apt-get install postgresql postgis postgresql-9.3-postgis-2.1 python3
python3-psycopg2 python3-xlrd, gdal-bin
```

```
# READ /usr/share/doc/postgresql-9.3-postgis/README.Debian.gz
# READ /usr/share/doc/postgresql-common/architecture.html
```

2) If desired, change "main" cluster from automatic to manual start.

In /etc/postgresql/9.3/main/start.conf change "auto" to "manual". This setting means that when "/etc/init.d/postgresql start" is issued at boot or on the command line, cluster "main" will NOT start, but could still be started manually via the pg_ctlcluster command.

3) Change port number to default 5432 in /etc/postgresql/9.3/main/postgresql.conf.

4) Start the database cluster.

```
:~$ sudo pg_ctlcluster 9.3 main start
```

5) Create postgresql ROLE with superuser privileges.

ROLE name should match the user's linux username. Issue command as linux user user postgres.

```
:~$ sudo su postgres
postgres:~$ createuser -s <username>
```

6) Create database

```
:~$ createdb nwpl
```

7) Enable postgis, including all functions and comments, for the database (documentation indicates to run these commands as linux user postgres, but any db superuser seems to work).

```
:~$ psql -d nwpl -c "CREATE EXTENSION postgis;"
:~$ psql -d nwpl -c "CREATE EXTENSION postgis_topology;"
```

Appendix 2 – Python program to process ancillary data: region GIS files, and USDA PLANTS data

file: base_process.py3

```
#!/usr/bin/python3
# coding: utf-8

###################################################################################
#
# AUTHOR(S):    Matthew F. Buff
# PURPOSE:      process base data
# COPYRIGHT:    Copyright 2013-2014 Matthew F. Buff
#
###################################################################################

import os, os.path, sys, zipfile, subprocess, psycopg2, time, calendar, xlrd, \
   collections, subprocess, shlex, urllib.request, tarfile

from math import log
from psycopg2 import errorcodes

public_nwpl_schema_file = 'public_nwpl_schema.sql'
public_mlra_process_file = 'public_mlra_process.sql'
public_states_fips_schema_file = 'public_states_fips_schema.sql'
public_regions_schema_file = 'public_regions_schema.sql'
plants_schema_file = 'public_usda_plants_schema.sql'
plants_pre_copy_file = 'public_usda_plants_pre_copy.sql'
plants_post_copy_file = 'public_usda_plants_post_copy.sql'

MLRA_url = 'http://www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/'
MLRA_file = 'nrcs142p2_052440.zip'

nwpl_file = 'National_2013v1_modified_for_db.csv'

states_url = 'http://dds.cr.usgs.gov/pub/data/nationalatlas/'
states_file = 'statep010_nt00798.tar.gz'

states_fips_url = 'http://www.census.gov/geo/reference/docs/'
states_fips_file = 'state.txt'

zip_dir = os.path.abspath(os.curdir)
extract_dir = zip_dir
```

```python
code_path = os.path.abspath(os.path.dirname(sys.argv[0]))

IEC_units = ('B', 'KiB', 'MiB', 'GiB')

def get_files(url,file_name):

    print('Checking for file ' + file_name)

    local_path = os.path.join(zip_dir, file_name)
    remote_url = url + file_name

    if os.path.isfile(local_path):
        local_exists = True
        local_size = os.path.getsize(local_path)
    else:
        local_exists = False

    resp = urllib.request.urlopen(remote_url)
    remote_size = int(resp.info().get('content-length'))
    remote_size_fmt = get_IEC_units(remote_size)

    download_file = False
    msg = file_name + ': '
    if local_exists:
        if local_size != remote_size:
            download_file = True
            msg = 'Remote and local file sizes are different: ' + \
                remote_size + ' bytes vs. ' + local_size + ' bytes.'
        else:
            msg = 'Local copy is up to date.'
    else:
        download_file = True
        msg = 'Local copy is missing.'

    print(msg)

    if download_file:
        print('Downloading ' + file_name + '.')
        with open(local_path, 'wb') as f:
            f.write(resp.read())


def truncate(conn, cur, tables):
    print('Truncating and resetting sequence columns for tables: ' + ', '.join(tables) + '.')
```

```
    sql = 'TRUNCATE ' + ', '.join(tables) + ' RESTART IDENTITY;'
    print(sql)
    cur.execute(sql)
    conn.commit()


def vacuum(conn, cur, tables):
    print('Vacuuming tables: ' + ', '.join(tables) + '.')
    iso = conn.isolation_level
    conn.set_isolation_level(0)
    for table in tables:
        cur.execute('VACUUM FULL ANALYZE ' + table + ';')
    conn.set_isolation_level(iso)
    conn.commit()


def get_IEC_units(bytes):
    exponent = int(log(bytes, 1024))
    return '{:.1f} {}'.format(float(bytes) / pow(1024, exponent),
        IEC_units[exponent])


def extract_files(in_file_name):
    file_ext = os.path.splitext(in_file_name)[1].lower()
    print('Extracting archives...', sep = '', end = '')
    in_file_path = os.path.join(zip_dir, in_file_name)
    if file_ext == '.zip':
        zipfile.ZipFile(in_file_path).extractall(path=extract_dir)
    elif file_ext == '.gz':
        tarfile.open(in_file_path).extractall(path=extract_dir)
    print('Finished extracting ' + in_file_name + '.')


def import_MLRA():
    print('Importing MLRA file via ogr...', sep = '', end = '')
    # Use ogr2ogr because it autodetects input projection; postgis shp2pgsql,
    # as of 2.1, requires manual setting of input projection
    cmd = 'ogr2ogr -overwrite -nlt MULTIPOLYGON -t_srs EPSG:2163 -f PostgreSQL
PG:"dbname=nwpl active_schema=public" mlra_v42.shp -lco GEOMETRY_NAME=geom'
    msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT)
    print('done.', sep = '')
    return
```

```
def update_MLRA(conn, cur):
    print('Updating region code in MLRA file ...', sep = '', end = '')
    mlra_sql_path = os.path.join(code_path, public_mlra_process_file)
    sql = open(mlra_sql_path).read()
    try:
        cur.execute(sql)
    except psycopg2.ProgrammingError as e:
        print(sql)
        print('\tpgsql error code:', e.pgcode, psycopg2.errorcodes.lookup(e.pgcode[:2]), ':',
psycopg2.errorcodes.lookup(e.pgcode))
        raise
    except:
        print(sql)
        print('Unknown error.')
        raise

    print('committing...')
    conn.commit()
    print('done.', sep = '')
    return


def import_states():
    print('Importing state boundaries file via ogr...', sep = '', end = '')
    # Use ogr2ogr because it autodetects input projection; postgis shp2pgsql,
    # as of 2.1, requires manual setting of input projection
    cmd = 'ogr2ogr -overwrite -nlt MULTIPOLYGON -t_srs EPSG:2163 -f PostgreSQL
PG:"dbname=nwpl active_schema=public" statep010.shp -lco GEOMETRY_NAME=geom'
    msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT)
    print('done.', sep = '')
    return


def run_sql(conn, cur, sql):
    print(sql)
    try:
        cur.execute(sql)
    except psycopg2.ProgrammingError as e:
        print(sql)
        print('\tpgsql error code:', e.pgcode, psycopg2.errorcodes.lookup(e.pgcode[:2]), ':',
psycopg2.errorcodes.lookup(e.pgcode))
        raise
    except:
        print(sql)
```

```
        print('Unknown error.')
        raise

    for notice in conn.notices:
        print(notice)
    conn.notices.clear()
    conn.commit()
    return



def run_psql(sql_path):
    # psycopg2 can't handle multi-statement sql
    try:
        #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        #check_call will halt python program if subprocess has non-zero return
        #subprocess.check_call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        cmd = 'psql -d nwpl -f ' + sql_path + ' --set=ON_ERROR_STOP=true'
        msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT,
universal_newlines=True)
        print(msg)
    except subprocess.CalledProcessError as e:
        print('command returned error code:', e.returncode)
        print('command was:', e.cmd)
        print('output was:', e.output)
        raise
    except:
        print(cmd)
        print('Unknown error')
        raise


def import_usda_plants():
    # psycopg2 copy_from cannot ignore quoted text in CSV files
    # psycopg2 copy_expert, i.e. COPY requires postgres user

    # PostgreSQL COPY command requires double quotes around db objects to
    # preserve case and single quotes around file path
    cmd = 'psql -d nwpl -c \'\copy usda_plants FROM usda_plants.csv CSV HEADER\'
--set=ON_ERROR_STOP=true'
    try:
        #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        #check_call will halt python program if subprocess has non-zero return
        msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT)
```

```
    except:
        print(cmd)
        raise


def import_nwpl():
    # psycopg2 copy_from cannot ignore quoted text in CSV files
    # psycopg2 copy_expert, i.e. COPY requires postgres user

    # PostgreSQL COPY command requires double quotes around db objects to
    # preserve case and single quotes around file path
    cmd = 'psql -d nwpl -c \'\copy "nwpl_2013" FROM National_2013v1_modified_for_db.csv
CSV HEADER\' --set=ON_ERROR_STOP=true'
    try:
        #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        #check_call will halt python program if subprocess has non-zero return
        msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT)
    except:
        print(cmd)
        raise


def import_states_fips():
    # psycopg2 copy_from cannot ignore quoted text in CSV files
    # psycopg2 copy_expert, i.e. COPY requires postgres user

    # PostgreSQL COPY command requires double quotes around db objects to
    # preserve case and single quotes around file path
    cmd = 'psql -d nwpl -c \"\copy states_fips FROM state.txt DELIMITER AS \'|\' CSV
HEADER\" --set=ON_ERROR_STOP=true'
    try:
        #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        #check_call will halt python program if subprocess has non-zero return
        msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT)
    except:
        print(cmd)
        raise


def main():
    # allow user to cancel the program
    user_continue = input('This program will delete data in the public schema of the nwpl
database. Continue (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
```

```
        print('Program canceled.')
        return 0
    else:
        print('Continuing.')

    # remember to manually start the db cluster if needed
    conn = psycopg2.connect(database = 'nwpl')
    cur = conn.cursor()

    user_continue = input('Create/replace regions definitions table (y/N)?')
    if user_continue.lower() in ('y', 'yes'):
        # create functions from local sql files
        print('Creating regions table.')
        run_psql(os.path.join(code_path, public_regions_schema_file))
        run_sql(conn, cur, 'SELECT public.public_regions_schema();')
    else:
        print('Skipping USDA PLANTS schema.')

    user_continue = input('Retrieve copy of MLRA file (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping check for new copy of MLRA file.')
    else:
        get_files(MLRA_url,MLRA_file)

    user_continue = input('Extract and import MLRA file to local database (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping import of MLRA file to local database.')
    else:
        extract_files(MLRA_file)
        import_MLRA()
        update_MLRA(conn, cur)
        vacuum(conn, cur, ['public.mlra_v42'])

    user_continue = input('Create/replace USDA PLANTS schema (y/N)?')
    if user_continue.lower() in ('y', 'yes'):
        # create functions from local sql files
        print('Creating USDA PLANTS schema.')
        run_psql(os.path.join(code_path, plants_schema_file))
        run_sql(conn, cur, 'SELECT public.public_usda_plants_schema();')
    else:
        print('Skipping USDA PLANTS schema.')

    print('If you wish to process USDA PLANTS data, complete the following steps before
continuing:')
```

```
    print('1) Go to the USDA PLANTS advanced search page:
http://plants.usda.gov/adv_search.html')
    print('2) choose "Any" for "PLANTS Floristic Area or Not".')
    print('3) choose "Any" for "State and Province".')
    print('4) Under section "Taxonomy", check the "Display" box for "Display all Synonyms",
"Display Authors and Scientific Name in separate fields." and for the following fields:')
    print('"Category", "Symbol", under Display Rank: ["Genus", "Species", "Subspecies",
"Variety", "Subvariety", "Forma"], "National Common Name", "Genus", "Family", "Family
Symbol", "Family Common Name", "ITIS TSN"')
    print('5) At the bottom of "Part A", choose "Download text file without formatted display".')
    print('6) At the bottom of "Part A", click "Display results".')
    print('7) Save the results (a text file) as "usda_plants.csv".')
    user_continue = input('Extract and import USDA PLANTS data (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping USDA PLANTS data.')
    else:
        # create functions from local sql files
        print('Importing USDA PLANTS data.')
        run_psql(os.path.join(code_path, plants_pre_copy_file))
        import_usda_plants()
        run_psql(os.path.join(code_path, plants_post_copy_file))

    user_continue = input('Create/replace NWPL schema (y/N)?')
    if user_continue.lower() in ('y', 'yes'):
        # create functions from local sql files
        print('Creating NWPL schema.')
        run_psql(os.path.join(code_path, public_nwpl_schema_file))
        run_sql(conn, cur, 'SELECT public.public_nwpl_schema();')
    else:
        print('Skipping NWPL schema.')

    print('If you wish to process NWPL data, complete the following steps before continuing:')
    print('1. Download
http://rsgisias.crrel.usace.army.mil/NWPL/static/cfg/doc/pdl_2013_pub/National/National_2013
v1.xlsx')
    print('2. Remove all text (near the top) leaving column headings and tabular data.')
    print('3. Save as comma separated values file: National_2013v1_modified_for_db.csv; quote
all fields; replace spaces in column names with underscores.')
    user_continue = input('Extract and import NWPL data (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping NWPL data.')
    else:
        print('Importing NWPL data.')
        import_nwpl()
```

```
    user_continue = input('Retrieve copy of state boundaries file (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping check for new copy of state boundaries file.')
    else:
        get_files(states_url,states_file)

    user_continue = input('Extract and import state boundaries file to local database (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping import of state boundaries file to local database.')
    else:
        extract_files(states_file)
        import_states()
        vacuum(conn, cur, ['public.statep010'])

    user_continue = input('Retrieve copy of state fips codes file (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping check for new copy of state fips codes file.')
    else:
        get_files(states_fips_url,states_fips_file)

    user_continue = input('Create/replace states_fips schema (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping state_fips schema.')
    else:
        # create functions from local sql files
        print('Creating state_fips schema.')
        run_psql(os.path.join(code_path, public_states_fips_schema_file))
        run_sql(conn, cur, 'SELECT public.public_states_fips_schema();')

    user_continue = input('Extract and import state fips codes data (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping state fips codes data.')
    else:
        print('Importing state fips codes data.')
        import_states_fips()

    cur.close()
    conn.close()

    return 0


if __name__ == "__main__":
```

main()

Appendix 3 – SQL program to create USACOE Region Names table schema

filename:public_regions_schema.sql

```
CREATE OR REPLACE FUNCTION public.public_regions_schema()
  RETURNS void AS
$BODY$
BEGIN

   DROP TABLE IF EXISTS public.regions CASCADE;
   CREATE TABLE public.regions
   (
      region_cd integer,
      region_abbr TEXT,
      region_name TEXT
   );

   ALTER TABLE public.regions ADD CONSTRAINT regions_pkey PRIMARY KEY
(region_cd);

   INSERT INTO public.regions
   (
     region_cd,
     region_abbr,
     region_name
   )
   VALUES
   (1,'NCNE','Northcentral and Northeast'),
   (2,'MW','Midwest'),
   (3,'EMP','Eastern Mountains and Piedmont'),
   (4,'GP','Great Plains'),
   (5,'AW','Arid West'),
   (6,'AGCP','Atlantic and Gulf Coastal Plain'),
   (7,'WMVC','Western Mountains, Valleys, and Coast'),
   (8,'AK','Alaska'),
   (9,'HI','Hawaii and Pacific Islands'),
   (10,'CB','Caribbean');

END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Appendix 4 – SQL program to define ACOE regions within MRLA table

filename:public_mlra_process.sql

```
-- define ACOE regions within MRLA table
ALTER TABLE public.mlra_v42
   ADD COLUMN region_cd integer;

--from: http://plants.usda.gov/wetinfo.html
UPDATE public.mlra_v42
SET region_cd =
   CASE
      WHEN lrrsym IN ('K', 'L', 'R')
         OR (mlrarsym = '149B' AND lrrsym = 'S') THEN
         1 --Northcentral and Northeast
      WHEN lrrsym = 'M' THEN
         2 --Midwest
      WHEN lrrsym = 'N'
         OR (mlrarsym = '136' AND lrrsym = 'P')
         OR (mlrarsym IN ('147', '148') AND lrrsym = 'S') THEN
         3 --Eastern Mountains and Piedmont
      WHEN lrrsym IN ('F', 'H', 'I', 'J')
         OR (lrrsym = 'G' AND mlrarsym  <> '62') THEN
         4 --Great Plains
      WHEN lrrsym IN ('B', 'C')
         OR (lrrsym = 'D' AND mlrarsym NOT IN ('22A', '22B', '39')) THEN
         5 --Arid West
      WHEN lrrsym IN ('O', 'T', 'U')
         OR (lrrsym = 'P' AND mlrarsym <> '136')
         OR (lrrsym = 'S' AND mlrarsym = '149A') THEN
         6 --Atlantic and Gulf Coastal Plain
      WHEN lrrsym IN ('A', 'E')
         OR (lrrsym = 'D' AND mlrarsym IN ('22A', '22B', '39'))
         OR (lrrsym = 'G' AND mlrarsym = '62') THEN
         7 --Western Mountains, Valleys, and Coast
      WHEN lrrsym IN ('W1', 'W2', 'X1', 'X2', 'Y') THEN
         8 --Alaska
      WHEN lrrsym IN ('V', 'Q') THEN
         9 --Hawaii and Pacific Islands
      WHEN lrrsym = 'Z' THEN
         10 --Caribbean
   END;
```

Appendix 5 – SQL program to define USDA PLANTS table schema

filename: public_usda_plants_schema.sql

```
CREATE OR REPLACE FUNCTION public.public_usda_plants_schema()
  RETURNS void AS
$BODY$
BEGIN

    DROP TABLE IF EXISTS
        usda_plants
    CASCADE;

    CREATE TABLE usda_plants (
        accepted_symbol TEXT,
        synonym_symbol TEXT,
        symbol TEXT,
        scientific_name TEXT,
        hybrid_genus_indicator TEXT,
        genus TEXT,
        hybrid_species_indicator TEXT,
        species TEXT,
        subspecies_prefix TEXT,
        hybrid_subspecies_indicator TEXT,
        subspecies TEXT,
        variety_prefix TEXT,
        hybrid_variety_indicator TEXT,
        variety TEXT,
        subvariety_prefix TEXT,
        subvariety TEXT,
        forma_prefix TEXT,
        forma TEXT,
        genera_binomial_author TEXT,
        trinomial_author TEXT,
        quadranomial_author TEXT,
        questionable_taxon_indicator TEXT,
        parents TEXT,
        common_name TEXT,
        category TEXT,
        genus2 TEXT,
        family TEXT,
        family_symbol TEXT,
        family_common_name TEXT,
        itis_tsn TEXT
```

```
    );

END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Appendix 6 – SQL program to prepare USDA PLANTS table for data

filename: public_usda_plants_pre_copy.sql

TRUNCATE TABLE usda_plants;

ALTER TABLE usda_plants **DROP CONSTRAINT IF EXISTS** usda_plants_pkey;

DROP INDEX IF EXISTS usda_plants_itis_tsn_idx;

Appendix 7 – SQL program to process USDA PLANTS table after receiving data

filename: public_usda_plants_post_copy.sql

```
BEGIN;

  CREATE TEMPORARY TABLE usda_plants_tmp ON COMMIT DROP AS
  SELECT DISTINCT
    *
  FROM
    usda_plants;

  TRUNCATE usda_plants;

  INSERT INTO
    usda_plants
  SELECT
    *
  FROM
    usda_plants_tmp;

COMMIT;

ALTER TABLE usda_plants ADD CONSTRAINT usda_plants_pkey PRIMARY KEY
(symbol);
CREATE INDEX usda_plants_itis_tsn_idx ON usda_plants (itis_tsn);
```

Appendix 8 – SQL program to define current NWPL table schema

filename: public_nwpl_schema.sql

```
CREATE OR REPLACE FUNCTION public.public_nwpl_schema()
  RETURNS void AS
$BODY$
BEGIN

    DROP TABLE IF EXISTS public.nwpl_2013;

    CREATE TABLE public.nwpl_2013
    (
    species text NOT NULL,
    authorship text,
    agcp text,
    ak text,
    aw text,
    cb text,
    emp text,
    gp text,
    hi text,
    mw text,
    ncne text,
    wmvc text,
    common_name text,
    CONSTRAINT nwpl_2013_pkey PRIMARY KEY (species)
    );

END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Appendix 9 – SQL program to define states FIPS codes table schema

filename: public_states_fips_schema.sql

```
CREATE OR REPLACE FUNCTION public.public_states_fips_schema()
  RETURNS void AS
$BODY$
BEGIN

    DROP TABLE IF EXISTS public.states_fips;

    CREATE TABLE public.states_fips
    (
    state_fips text,
    stusab text,
    state_name text,
    statens text,
    CONSTRAINT states_fips_pkey PRIMARY KEY (state_fips)
    );

END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Appendix 10 – Python program to process NPS data

filename: nps_xls_process.py3

```
#!/usr/bin/python3
# coding: utf-8

###############################################################################
#
# AUTHOR(S):    Matthew F. Buff
# PURPOSE:      process NPS data
# COPYRIGHT:    Copyright 2013 Matthew F. Buff
#
###############################################################################

import os, os.path, sys, zipfile, subprocess, psycopg2, time, calendar, xlrd, \
    collections, subprocess, webbrowser, shlex

from math import log
from psycopg2 import errorcodes

boundaries_file = 'nps_boundary.zip'
schema_file = 'nps_schema.sql'
process_file = 'nps_process.sql'
geography_file = 'nps_geography.sql'
summaries_file = 'nps_summaries.sql'

code_path = os.path.abspath(os.path.dirname(sys.argv[0]))
zip_dir = os.path.abspath(os.curdir)
extract_dir = zip_dir

IEC_units = ('B', 'KiB', 'MiB', 'GiB')

# some AA obs have Cowardin
# check others if time, will increase sample size

DBTable = collections.namedtuple('DBSchema', ['table_name', 'fields'])
DBField = collections.namedtuple('DBField', ['field_name', 'aliases'])
XLField = collections.namedtuple('XLField', ['field_name', 'colnum'])

db_tables = (
    DBTable(
        table_name = 'plots_xls',
        fields = (
```

```
            DBField('plot_code', frozenset(['plotcode', 'plot', 'plcd'])),
            DBField('plot_event', frozenset(['plotevent'])),
            DBField('state', frozenset(['state'])),
            DBField('comm_type', frozenset(['comtype'])),
            DBField('cowardin', frozenset(['cowardinsystem', 'cowsys', 'cowardin'])),
            DBField('hydro_regime', frozenset(['hydroregime', 'hydrology', 'hydro'])),
            DBField('hydro_evidence', frozenset(['hydrologyevidence'])),
            DBField('soil_drainage', frozenset(['soildrainage'])),
            DBField('plot_shape', frozenset(['plotshape'])),
            DBField('plot_radius', frozenset(['plotradius(m)'])),
            DBField('plot_diam', frozenset(['plotdiam'])),
            DBField('x_dim', frozenset(['xdimension', 'xdim', 'plotlength'])),
            DBField('y_dim', frozenset(['ydimension', 'ydim', 'plotwidth'])),
            DBField('utm_x_field', frozenset(['fieldx', 'fieldutmx', 'gpsutmx', 'utmeasting', 'utmx',
'utme', 'fieldx', 'rawutmx'])),
            DBField('utm_y_field', frozenset(['fieldy', 'fieldutmy', 'gpsutmy', 'utmnorthing', 'utmy',
'utmn', 'fieldy', 'rawutmy'])),
            DBField('utm_x_corrected', frozenset(['correctedutmx', 'correctedutme', 'corrutmx'])),
            DBField('utm_y_corrected', frozenset(['correctedutmy', 'correctedutmn', 'corrutmy'])),
            DBField('utm_zone', frozenset(['utmzone'])),
            DBField('lat_field', frozenset(['fieldlat'])),
            DBField('lon_field', frozenset(['fieldlong'])),
            DBField('gps_datum', frozenset(['gpsdatum', 'proj', 'datum', 'mapproj'])),
            DBField('gps_coord_system', frozenset(['coordsystem'])),
            DBField('gps_techniques', frozenset(['gpstechniques']))
        )
    ),
    DBTable(
        table_name = 'sp_cov_xls',
        fields = (
            DBField('plot_code', frozenset(['plotcode', 'plot', 'plcd', 'aaobscode'])),
            DBField('plot_event', frozenset(['plotevent'])),
            DBField('plant_symbol', frozenset(['plantsymbol', 'plantssymbol', 'plantnames',
'plantcode', 'plantscode', 'sppcode'])),
            DBField('itis_tsn', frozenset(['itistsn', 'tsncode', 'tsn'])),
            DBField('in_plot', frozenset(['withinplot'])),
            DBField('sci_name', frozenset(['scientificname', 'fieldname', 'localtaxonname',
'latinname', 'species', 'speciesname'])),
            DBField('com_name', frozenset(['commonname', 'comname', 'species'])),
            DBField('sci_family', frozenset(['family', 'familyname'])),
            DBField('used_plants', frozenset(['usedplants'])),
            DBField('source', frozenset(['source']))
        )
    )
```

```
)


def get_park_files():
    # get lists of MS Excel plots and sp_cov files in current dir
    fls_plots = [entry for entry in os.listdir(os.curdir) if os.path.isfile(entry) and
        os.path.splitext(entry)[1] in ('.xls','.xlsx') and os.path.splitext(entry)[0].endswith(('_plots'))]
    fls_sp_cov = [entry for entry in os.listdir(os.curdir) if os.path.isfile(entry) and
        os.path.splitext(entry)[1] in ('.xls','.xlsx') and os.path.splitext(entry)
[0].endswith(('_sp_cov'))]

    parks_plots = frozenset([os.path.splitext(fl)[0].replace('_plots', '') for fl in fls_plots])
    parks_sp_cov = frozenset([os.path.splitext(fl)[0].replace('_sp_cov', '') for fl in fls_sp_cov])

    # _plots with no _sp_cov
    for park in parks_plots.difference(parks_sp_cov):
        print('Park', park, 'has a plots file, but no matching sp_cov file. Stopping processing.')
        return
    # _sp_cov with no _plots
    for park in parks_sp_cov.difference(parks_plots):
        print('Park', park, 'has a sp_cov file, but no matching plots file. Stopping processing.')
        return

    # parks with both files
    ParkFiles = collections.namedtuple('ParkFiles', ['park_code','plots_xls','sp_cov_xls'])
    parks = []
    parks_both = list(parks_plots.intersection(parks_sp_cov))
    parks_both.sort()
    for park in parks_both:
        fl_p = [fl for fl in fls_plots if os.path.splitext(fl)[0].replace('_plots', '') == park]
        fl_s = [fl for fl in fls_sp_cov if os.path.splitext(fl)[0].replace('_sp_cov', '') == park]
        parks.append(ParkFiles(park_code = park, plots_xls = fl_p[0], sp_cov_xls = fl_s[0]))

    return parks


def get_xls_wrksheet(fl):
    try:
        book = xlrd.open_workbook(fl)
        sheet = book.sheet_by_index(0)
    except:
        print('xlrd encountered an error reading file', fl)

    return sheet
```

```
def simplify_str(s):
    s = s.replace(' ', '').replace('_', '').lower()
    return s


def cast_vals(val_in):
    val_in = str(val_in)
    if val_in.strip() == '':
        val_out = None
    else:
        val_out = val_in.strip()

    return val_out


def get_vals(park, table, fields, xls_file):
    sheet = get_xls_wrksheet(xls_file)
    flds_found = []
    flds_missing = []
    for fld in fields:
        for colnum in range(sheet.ncols):
            found = False
            if simplify_str(sheet.cell_value(0, colnum)) in fld.aliases:
                flds_found.append(XLField(field_name = fld.field_name, colnum = colnum))
                found = True
                # stop searching
                break

        if not found:
            flds_missing.append(fld.field_name)

    err_msg = check_missing_fields(tuple(flds_missing), table)

    data = []

    for rownum in range(1, sheet.nrows):
        data.append(tuple([xls_file, park] + [cast_vals(sheet.cell_value(rownum, fld.colnum)) for
fld in flds_found]))

    sql_flds = ['file_name','park_code'] + [fld.field_name for fld in flds_found]

    sql = 'INSERT INTO nps.' + table + ' (' + ', '.join(sql_flds) + ') VALUES (' + ', '.join(['%s'] *
```

```
len(sql_flds)) + ');'

    return sql, tuple(data), err_msg


def check_missing_fields(fields, table):
    msg = ''
    if table == 'plots_xls':
        must_have = frozenset(['plot_code', 'cowardin'])
        missing_must_have = must_have.intersection(fields)
        msg = msg + ', '.join(list(missing_must_have))
    return msg


def truncate(conn, cur, tables):
    print('Truncating and resetting sequence columns for tables: ' + ', '.join(tables) + '.')
    sql = 'TRUNCATE ' + ', '.join(tables) + ' RESTART IDENTITY;'
    print(sql)
    cur.execute(sql)
    conn.commit()


def vacuum(conn, cur, tables):
    print('Vacuuming tables: ' + ', '.join(tables) + '.')
    iso = conn.isolation_level
    conn.set_isolation_level(0)
    for table in tables:
        cur.execute('VACUUM FULL ANALYZE ' + table + ';')
    conn.set_isolation_level(iso)
    conn.commit()


def copy_tables(park_files, conn, cur):
    park_count = len(park_files)
    for counter, park in enumerate(park_files):
        print('Processing ', counter + 1, ' of ', park_count, ': ', park.park_code, sep = '')
        for table in db_tables:
            #if counter+1 < 104: break # use to skip parks
            print('\ttable', table.table_name)
            sql, data, err_msg = get_vals(park.park_code, table.table_name, table.fields, getattr(park,
table.table_name))
            if len(err_msg) >= 1:
                print('\t\tERROR missing fields', err_msg)
                raise
```

```
        try:
            cur.executemany(sql, data)
        except psycopg2.ProgrammingError as e:
            print(data)
            print(sql)
            print('\tpgsql error code:', e.pgcode, psycopg2.errorcodes.lookup(e.pgcode[:2]), ':',
psycopg2.errorcodes.lookup(e.pgcode))
            raise
        except:
            print(data)
            print(sql)
            print('Unknown error', park)
            raise

    print('Committing.')
    conn.commit()


def get_IEC_units(bytes):
    exponent = int(log(bytes, 1024))
    return '{:.1f} {}'.format(float(bytes) / pow(1024, exponent),
        IEC_units[exponent])


def get_park_boundaries():
    # relevant web pages:
    # https://irma.nps.gov/App/Portal/Home/FeaturedContent
    # https://irma.nps.gov/App/Reference/Profile/2208069
    # https://irma.nps.gov/App/Reference/Profile/2194483?lnv=true
    # https://irma.nps.gov/App/Reference/DownloadDigitalFile?
code=490634&file=nps_boundary.zip
    # http://irmafiles.nps.gov/Reference/Holding/490634/nps_boundary.zip
    webbrowser.open('http://irmafiles.nps.gov/Reference/Holding/490634/nps_boundary.zip')
    print()

def extract_files(in_file_name):
    print('Extracting archives...', sep = '', end = '')
    in_file_path = os.path.join(zip_dir, in_file_name)
    zipfile.ZipFile(in_file_path).extractall(path=extract_dir)
    print('Finished extracting ' + in_file_name + '.')


def import_park_boundaries():
```

```
    print('Importing park boundaries via ogr...', sep = '', end = '', flush=False)
    # Use ogr2ogr because it autodetects input projection; postgis shp2pgsql,
    # as of 2.1, requires manual setting of input projection
    cmd = 'ogr2ogr -overwrite -nlt MULTIPOLYGON -t_srs EPSG:2163 -f PostgreSQL
PG:"dbname=nwpl active_schema=nps" nps_boundary.shp -lco GEOMETRY_NAME=geom'
    msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT)
    print('done.', sep = '')
    return


def run_psql(sql_path):
    # psycopg2 can't handle multi-statement sql
    try:
        #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        #check_call will halt python program if subprocess has non-zero return
        #subprocess.check_call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        cmd = 'psql -d nwpl -f ' + sql_path + ' --set=ON_ERROR_STOP=true'
        msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT,
universal_newlines=True)
        print(msg)
    except subprocess.CalledProcessError as e:
        print('command returned error code:', e.returncode)
        print('command was:', e.cmd)
        print('output was:', e.output)
        raise
    except:
        print(cmd)
        print('Unknown error')
        raise

def run_sql(conn, cur, sql):
    print(sql)
    cur.execute(sql)
    for notice in conn.notices:
        print(notice)
    conn.notices.clear()
    conn.commit()


def main():
    # allow user to cancel the program
    user_continue = input('This program will delete data in the nps database. Continue (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Program canceled.')
```

```
        return 0
    else:
        print('Continuing.')


    # get a dictionary of all parks and corresponding excel files in dir
    park_files = get_park_files()

    # remember to manually start the db cluster
    conn = psycopg2.connect(database = 'nwpl')
    cur = conn.cursor()

    user_continue = input('Create local NPS database structure (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping the creation of local NPS database structure.')
    else:
        print('Creating local NPS database structure.')
        run_sql(conn, cur, 'CREATE SCHEMA IF NOT EXISTS nps;')
        run_psql(os.path.join(code_path, schema_file))
        run_sql(conn, cur, 'SELECT nps.nps_schema();')
        # create functions from local sql files
        run_psql(os.path.join(code_path, process_file))
        run_psql(os.path.join(code_path, geography_file))
        run_psql(os.path.join(code_path, summaries_file))

    user_continue = input('Copy vegetation and plot data to database tables (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping copying of vegetation and plot data.')
    else:
        # truncate tables and reset sequence columns before loading data
        # need double quotes around db objects to preseve case
        truncate(conn, cur, ['"nps".' + table.table_name for table in db_tables])
        vacuum(conn, cur, ['"nps".' + table.table_name for table in db_tables])
        copy_tables(park_files, conn, cur)
        vacuum(conn, cur, ['"nps".' + table.table_name for table in db_tables])

    user_continue = input('Retrieve copy of boundaries file (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping check for new copy of boundaries file.')
    else:
        print('If the link is broken, search for "Current Administrative Boundaries of National Park
System Units" and download the latest version.')
        get_park_boundaries()
```

```
    user_continue = input('Extract and import boundaries file to local database (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping import of boundaries file to local database.')
    else:
        extract_files(boundaries_file)
        import_park_boundaries()

    user_continue = input('Run sql functions (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Skipping sql functions.')
    else:
        print('Running nps.process.')
        run_sql(conn, cur, 'SELECT nps.nps_process();')
        print('Running nps.geography.')
        run_sql(conn, cur, 'SELECT nps.nps_geography();')
        print('Running nps.summaries.')
        run_sql(conn, cur, 'SELECT nps.nps_summaries();')

    cur.close()
    conn.close()

    return 0


if __name__ == "__main__":
    main()
```

Appendix 11 – SQL program to create NPS tables

filename: nps_schema.sql

```
CREATE OR REPLACE FUNCTION nps.nps_schema()
  RETURNS void AS
$BODY$
BEGIN
  SET search_path TO nps;

  -- table for raw data from plots xls files
  DROP TABLE IF EXISTS plots_xls CASCADE;
  CREATE TABLE plots_xls (
    recid SERIAL PRIMARY KEY,
    file_name TEXT,
    park_code TEXT,
    plot_code TEXT,
    plot_event TEXT,
    state TEXT,
    comm_type TEXT,
    cowardin TEXT,
    hydro_regime TEXT,
    hydro_evidence TEXT,
    soil_drainage TEXT,
    plot_shape TEXT,
    plot_radius TEXT,
    plot_diam TEXT,
    x_dim TEXT,
    y_dim TEXT,
    utm_x_field TEXT,
    utm_y_field TEXT,
    utm_x_corrected TEXT,
    utm_y_corrected TEXT,
    utm_zone TEXT,
    lat_field TEXT,
    lon_field TEXT,
    gps_datum TEXT,
    gps_coord_system TEXT,
    gps_techniques TEXT
  );

  CREATE INDEX ON plots_xls (plot_code);
  CREATE INDEX ON plots_xls (plot_event);
  CREATE INDEX ON plots_xls (cowardin);
```

```
    COMMENT ON COLUMN plots_xls.comm_type IS 'Community type, i.e. wetland/upland';


    -- table for raw data from species xls files
    DROP TABLE IF EXISTS sp_cov_xls CASCADE;
    CREATE TABLE sp_cov_xls (
      recid SERIAL PRIMARY KEY,
      file_name TEXT,
      park_code TEXT,
      plot_code TEXT,
      plot_event TEXT,
      plant_symbol TEXT,
      itis_tsn TEXT,
      sci_name TEXT,
      com_name TEXT,
      sci_family TEXT,
      in_plot TEXT,
      used_plants TEXT,
      source TEXT
    );

    CREATE INDEX ON sp_cov_xls (plot_code);
    CREATE INDEX ON sp_cov_xls (plot_event);

    COMMENT ON COLUMN sp_cov_xls.plot_code IS 'Key field from Plots table';
    COMMENT ON COLUMN sp_cov_xls.plant_symbol IS 'From Plants table';
    COMMENT ON COLUMN sp_cov_xls.used_plants IS 'Yes if name came from the
PLANTS database';
    COMMENT ON COLUMN sp_cov_xls.source IS 'From Plant List table: SS or NS';
END;
$BODY$
    LANGUAGE plpgsql VOLATILE
    COST 100;
```

Appendix 12 – SQL Program to process NPS data

filename:nps_process.sql

```
CREATE OR REPLACE FUNCTION nps.nps_process()
  RETURNS void AS
$BODY$
DECLARE
  rows_affected integer := 0;
BEGIN
  SET search_path TO nps;
  RAISE NOTICE 'Total expected runtime: ~127 seconds.';


  UPDATE
    sp_cov_xls
  SET
    itis_tsn = NULL
  WHERE
    itis_tsn  = '999999999999.0';
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Set invalid itis_tsn code to null.
  % rows affected.', rows_affected;

  UPDATE
    sp_cov_xls
  SET
    itis_tsn = replace(itis_tsn, '.0', '')
  WHERE
    itis_tsn LIKE '%.0';
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Drop zero decimal from itis_tsn code.
  % rows affected.', rows_affected;

  UPDATE
    sp_cov_xls
  SET
    itis_tsn = replace(itis_tsn, '-', '')
  WHERE
    itis_tsn LIKE '-%';
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Remove negative sign prefix from itis_tsn code.
  % rows affected.', rows_affected;
```

```
UPDATE
   plots_xls
SET
   cowardin = lower(cowardin),
   comm_type = lower(comm_type);
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: normalize Cowardin and comm_type case.
% rows affected.', rows_affected;


UPDATE
   plots_xls
SET
   cowardin =
      CASE
      WHEN cowardin IN ('1', '1.0') THEN
         'estuarine'
      WHEN cowardin IN ('2', '2.0') THEN
         'riverine'
      WHEN cowardin IN ('3', '3.0') THEN
         'palustrine'
      WHEN cowardin IN ('4', '4.0') THEN
         'lacustrine'
      WHEN comm_type = 'upland' THEN
         'upland'
      ELSE
         cowardin
      END
WHERE
   park_code IN ('waca', 'wupa')
   AND
      ( cowardin NOT IN ('marine', 'estuarine', 'riverine', 'lacustrine', 'palustrine', 'upland')
      OR
      cowardin IS NULL );
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: normalize Cowardin values for waca and wupa.
% rows affected.', rows_affected;


UPDATE
   plots_xls
SET
   cowardin =
```

```
        CASE cowardin
        WHEN '1.0' THEN
            'lacustrine'
        WHEN '2.0' THEN
            'palustrine'
        WHEN '3.0' THEN
            'riverine'
        WHEN '4.0' THEN
            'upland'
        ELSE
            cowardin
        END
WHERE
    park_code IN ('chcu', 'nava')
    AND
        ( cowardin NOT IN ('marine', 'estuarine', 'riverine', 'lacustrine', 'palustrine', 'upland')
        OR
        cowardin IS NULL );
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: normalize Cowardin values for chcu and nava.
% rows affected.', rows_affected;


UPDATE
    plots_xls
SET
    cowardin =
        CASE cowardin
        WHEN 'palustine' THEN
            'palustrine'
        WHEN 'wetland/riparian' THEN
            'riverine'
        ELSE
            cowardin
        END
WHERE
    park_code IN ('scbl', 'pinn')
    AND
        ( cowardin NOT IN ('marine', 'estuarine', 'riverine', 'lacustrine', 'palustrine', 'upland')
        OR
        cowardin IS NULL);
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: normalize Cowardin values for scbl and pinn.
% rows affected.', rows_affected;
```

```sql
DELETE
FROM
    plots_xls
WHERE
    plot_code IS NULL
    AND plot_event IS NULL;
-- was frhi, nava, 2,024 rows
-- now nava 2 rows (difference in new python3/xlrd?
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: delete rows missing both plot_code and plot_event.
% rows affected.', rows_affected;


DELETE
FROM
    sp_cov_xls
WHERE
    plot_code IS NULL
    AND plot_event IS NULL;
-- was frhi, gewa, and morr have a combined 66,103 blank rows in xls files
-- now gewa, and morr have a combined 65,365 blank rows in xls files (python3/xlrd)
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Species: delete rows missing both plot_code and event_code.
% rows affected.', rows_affected;


DELETE
FROM
    sp_cov_xls
WHERE
    in_plot IN ('0','No');
GET DIAGNOSTICS rows_affected = ROW_COUNT;
/* No guarantee that these rows will be in the same wetland/upland
condition as the plot. Delete 19,763 species rows outside of plots. */
RAISE NOTICE 'Species: delete rows outside of plots.
% rows affected.', rows_affected;


UPDATE
    sp_cov_xls
SET
    plot_code =
```

```
        CASE
        WHEN park_code = 'ozar' AND RIGHT(plot_event, 9) IN ('-ECS 1996', '-RIP 1998')
THEN
          LEFT(plot_event, -9)
        WHEN park_code = 'fopo' AND RIGHT(plot_event, 3) = '_11' THEN
          LEFT(plot_event, -3)
        WHEN park_code = 'pore_goga' AND RIGHT(plot_event, 3) = '_11' THEN
          LEFT(plot_event, -3)
        WHEN park_code IN ('romo','seki') THEN
          LEFT(plot_event, -3)
        WHEN park_code IN ('beol','fopo','ozar','pore_goga','sand','yose') THEN
          LEFT(plot_event, -2)
        END,
      plot_event = NULL
    WHERE
      (
        (park_code = 'ozar' AND RIGHT(plot_event, 9) IN ('-ECS 1996', '-RIP 1998'))
        OR
        (park_code = 'fopo' AND RIGHT(plot_event, 3) = '_11')
        OR
        (park_code = 'pore_goga' AND RIGHT(plot_event, 3) = '_11')
        OR
        (park_code IN ('romo','seki'))
        OR
        (park_code IN ('beol','fopo','ozar','pore_goga','sand','yose'))
      )
      AND
      plot_code IS NULL;
    /* 73,621 rows */
    GET DIAGNOSTICS rows_affected = ROW_COUNT;
    RAISE NOTICE 'Species: correct inconsistent code/event usage. Move corrected event code
to plot code and delete event code.
    % rows affected.', rows_affected;


    UPDATE
      sp_cov_xls
    SET
      plot_code =
        CASE
        WHEN park_code = 'voya' AND plot_code = 'VOYA.8' THEN
          'VOYA.08'
        WHEN park_code = 'voya' AND plot_code = 'VOYA.136A' THEN
          'VOYA.136'
```

```
        WHEN park_code = 'scbl' AND plot_code = 'SEEP' THEN
          'SEEP1'
        WHEN park_code = 'scbl' AND plot_code = 'LS1' THEN
          'CS1'
        WHEN park_code = 'moru' AND plot_code LIKE 'MORU._' THEN
          'MORU.0' || RIGHT(plot_code, 1)
        WHEN park_code = 'ozar' AND plot_code <> UPPER(plot_code) THEN
          UPPER(plot_code)
        END
WHERE
    (park_code = 'voya' AND plot_code = 'VOYA.8')
    OR
    (park_code = 'voya' AND plot_code = 'VOYA.136A')
    OR
    (park_code = 'scbl' AND plot_code = 'SEEP')
    OR
    (park_code = 'scbl' AND plot_code = 'LS1')
    OR
    (park_code = 'moru' AND plot_code LIKE 'MORU._')
    OR
    (park_code = 'ozar' AND plot_code <> UPPER(plot_code));
-- 416 rows
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Species: update miscoded codes and typographical errors.
% rows affected.', rows_affected;


UPDATE
    plots_xls
SET
    plot_code =
        CASE
        WHEN park_code = 'ozar' AND plot_code <> UPPER(plot_code) THEN
          UPPER(plot_code)
        WHEN park_code = 'crmo' AND RIGHT(plot_code, 2) = '.0' THEN
          LEFT(plot_code, -2)
        END
WHERE
    (park_code = 'ozar' AND plot_code <> UPPER(plot_code))
    OR
    (park_code = 'crmo' AND RIGHT(plot_code, 2) = '.0');
-- 94 rows
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: correct corrupt codes treated as numbers by Excel.
```

```
% rows affected.', rows_affected;


UPDATE
   plots_xls
SET
   plot_event =
      CASE
      WHEN park_code = 'liri' AND plot_event IN ('LIRI.7901') THEN
         'LIRI.07901'
      END
WHERE
   (park_code = 'liri' AND plot_event IN ('LIRI.7901'));
/* liri: LIRI.7901 should be LIRI.07901
1 record */
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: correct miscoded event (typographical errors).
% rows affected.', rows_affected;


DELETE
FROM
   plots_xls
WHERE
   recid IN
   (
   SELECT recid
   FROM
      (
      SELECT
         recid,
         row_number() OVER
         (
            PARTITION BY
               file_name,
               park_code,
               plot_code,
               plot_event,
               state,
               comm_type,
               cowardin,
               hydro_regime,
               hydro_evidence,
               soil_drainage,
```

```
                plot_shape,
                plot_radius,
                plot_diam,
                x_dim,
                y_dim
            ) AS row_num
        FROM plots_xls
        ) AS ag1
    WHERE
        row_num > 1);
-- 6,541 rows
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: delete rows that are duplicate in every field.
% rows affected.', rows_affected;


DELETE
FROM
    plots_xls
WHERE
    plot_shape = 'N/A'
    AND
    plot_code IN
    (
    SELECT
        plot_code
    FROM
        (
        SELECT
            file_name,
            plot_code,
            COUNT(DISTINCT plot_shape) as ct
        FROM
            plots_xls
        GROUP BY
            file_name,
            plot_code
        HAVING
            COUNT(DISTINCT plot_shape) > 1
        ORDER BY
            file_name,
            plot_code
        ) AS ag
    );
```

```
-- delete 204 of 408 duplicate shape rows, all in ozar
-- target the ones w/shape = 'N/A'
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: delete rows duplicate except in shape. Target copies w/shape = "N/A".
% rows affected.', rows_affected;


DELETE
FROM
   sp_cov_xls
WHERE
   recid IN
      (
      SELECT recid
      FROM
         (
         SELECT
            recid,
            row_number() OVER
            (
               PARTITION BY
                  file_name,
                  park_code,
                  plot_code,
                  plot_event,
                  plant_symbol,
                  sci_name,
                  com_name,
                  sci_family,
                  in_plot,
                  used_plants
               ORDER BY
                  source
            ) AS row_num
         FROM sp_cov_xls
         ) AS ag1
      WHERE row_num > 1
      );
/* Nulls sort last so keep ASC sort to choose NULL values for deletion.
100,344 rows, ~45 seconds. */
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Species: delete rows duplicate except in source.
% rows affected.', rows_affected;
```

```
DELETE
FROM
  sp_cov_xls
WHERE
  recid IN
    (
    SELECT recid
    FROM
      (
      SELECT
        recid,
        row_number() OVER
        (
          PARTITION BY
            file_name,
            park_code,
            plot_code,
            plot_event,
            plant_symbol,
            sci_name,
            com_name,
            sci_family,
            in_plot
          ORDER BY
            used_plants DESC
        ) AS row_num
      FROM sp_cov_xls
      ) AS ag1
    WHERE row_num > 1
    );
/* Sort DESC to select used_plants = 0 for deletion.
16 rows, ~27 seconds. */
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Species: delete rows duplicate except in used_plants.
% rows affected.', rows_affected;


DELETE
FROM
  sp_cov_xls
WHERE
  recid IN
    (
```

```
        SELECT recid
        FROM
            (
            SELECT
                recid,
                row_number() OVER
                (
                    PARTITION BY
                        file_name,
                        park_code,
                        plot_code,
                        plot_event,
                        plant_symbol,
                        sci_name,
                        com_name
                    ORDER BY
                        sci_family
                ) AS row_num
            FROM
                sp_cov_xls
            ) AS ag1
        WHERE row_num > 1
        );
/* prefer non-null family (NULLs sort last in PostgreSQL by default)
2 rows, ~26 seconds */
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Species: duplicate except in family. Prefer copies with non-null family.
% rows affected.', rows_affected;


DELETE
FROM
    sp_cov_xls
WHERE
    recid IN
        (
        SELECT recid
        FROM
            (
            SELECT
                recid,
                row_number() OVER
                (
                    PARTITION BY
```

```
                file_name,
                park_code,
                plot_code,
                plot_event,
                plant_symbol,
                sci_name
              ORDER BY
                com_name
            ) AS row_num
          FROM
            sp_cov_xls
          ) AS ag1
        WHERE row_num > 1
        );
    /* prefer non-null com_name (NULLs sort last in PostgreSQL by default)
    20 rows, ~26 seconds */
    GET DIAGNOSTICS rows_affected = ROW_COUNT;
    RAISE NOTICE 'Species: delete rows duplicate except in com_name. Prefer copies with non-
null com_name.
    % rows affected.', rows_affected;


    DELETE
    FROM plots_xls
    WHERE
      recid IN
      (
      SELECT
        recid
      FROM
        plots_xls
      EXCEPT
      SELECT
        p.recid
      FROM plots_xls AS p
       INNER JOIN sp_cov_xls AS s
        ON p.park_code = s.park_code
        AND ((p.plot_event = s.plot_event AND s.plot_code IS NULL)
         OR (p.plot_code = s.plot_code AND s.plot_event IS NULL))
      );
    /* Some parks show up for multiple reasons, e.g. stri.
    For the individual plot codes/events here:
    acad: these appear to all involve plots with NO rows where 'Within Plot' = TRUE
    ruca, stri: same for the one affected plot
```

```
These plot rows simply have no match in species rows in the xls files
for the codes/events selected here. These data may be recoverable from the
Access database files.
apis, chcu, cong, deto, flfo, fopo, glac, grsm, hafo, jeca, maca, moru,
nava, ozar, pinn, piro, pore_goga, romo, seki, shen, stri, thst, yose

2,001 rows */
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Plots: delete rows with no match in species table.
% rows affected.', rows_affected;


DELETE
FROM
   sp_cov_xls
WHERE
   park_code = 'moru' AND plot_code = 'MORU.20'
   OR
   park_code = 'isro' AND plot_code = 'ISRO.999'
   OR
   park_code = 'grte' AND (plot_code = 'GT-03B082' OR plot_code NOT LIKE 'GT-03%')
   OR
   park_code = 'cure' AND plot_code IN ('CURE.0012', 'CURE.0039', 'CURE.0203',
      'CURE.0321', 'CURE.0322', 'CURE.0323')
   OR
   park_code = 'blca' AND plot_code IN ('BLCA.0036', 'BLCA.0037', 'BLCA.0038',
      'BLCA.0039', 'BLCA.0040', 'BLCA.0041', 'BLCA.0042', 'BLCA.0082',
      'BLCA.0083')
   OR
   park_code = 'arch' AND plot_code IN ('ARCH.0647', 'ARCH.0649', 'ARCH.0650');
/* NOTE: Most (all?) of these rows can be recovered by working with the
Access database files.
22,831 rows */
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Species: delete rows with no match in plots table.
% rows affected.', rows_affected;


UPDATE
   plots_xls
SET
   plot_diam = replace(plot_diam, ' m', '')
WHERE
```

```
        plot_diam LIKE '% m';
    GET DIAGNOSTICS rows_affected = ROW_COUNT;
    RAISE NOTICE 'Drop unit text from plot_diam.
    % rows affected.', rows_affected;


END;
$BODY$
    LANGUAGE plpgsql VOLATILE
    COST 100;
```

Appendix 13 – SQL program to add spatial columns to NPS data

filename: nps_geography.sql

```
CREATE OR REPLACE FUNCTION nps.nps_geography()
 RETURNS void AS
$BODY$
DECLARE
   rows_affected integer := 0;
BEGIN
   SET search_path TO nps, public;
   RAISE NOTICE 'Total expected runtime: ~308 seconds.';


   ALTER TABLE nps.plots_xls
      DROP COLUMN IF EXISTS geom CASCADE,
      DROP COLUMN IF EXISTS region_cd_bnd CASCADE,
      DROP COLUMN IF EXISTS region_cd_plot CASCADE,
      DROP COLUMN IF EXISTS state_fips_bnd CASCADE,
      DROP COLUMN IF EXISTS state_fips_plot CASCADE,
      DROP COLUMN IF EXISTS park_code_bnd CASCADE;
   RAISE NOTICE 'Drop geography related columns from plots table.';


   ALTER TABLE nps.plots_xls
      ADD COLUMN geom GEOMETRY(POINT,2163),
      ADD COLUMN region_cd_bnd integer,
      ADD COLUMN region_cd_plot integer,
      ADD COLUMN state_fips_bnd TEXT,
      ADD COLUMN state_fips_plot TEXT,
      ADD COLUMN park_code_bnd TEXT;
   /* EPSG:2163 - US National Atlas Lambert Azimuthal Equal Area
   http://spatialreference.org/ref/epsg/2163/ */
   RAISE NOTICE 'Add geography related columns to plots table.';


   DROP TABLE IF EXISTS nps.parks_regions CASCADE;
   CREATE TABLE nps.parks_regions AS
   SELECT
      LOWER(b.unit_code) AS park_code,
      m.region_cd,
      ST_Area(ST_Multi(ST_Union(ST_Intersection(b.geom, m.geom)))) AS area,
      rank() OVER (PARTITION BY b.unit_code ORDER BY
ST_Area(ST_Multi(ST_Union(ST_Intersection(b.geom, m.geom)))) DESC) AS area_rank,
```

```
    ST_Multi(ST_Union(ST_Intersection(b.geom, m.geom))) AS geom
  FROM
    nps.nps_boundary b
    INNER JOIN public.mlra_v42 m
      ON ST_Intersects(b.geom, m.geom)
  GROUP BY
    b.unit_code,
    m.region_cd
  ORDER BY
    b.unit_code,
    area_rank;
  -- ~55 seconds
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Create/replace parks_regions table and populate with intersection of park
boundaries and regions.
  % rows affected.', rows_affected;


  ALTER TABLE nps.parks_regions ADD CONSTRAINT parks_regions_pkey PRIMARY
KEY (park_code, region_cd);
  RAISE NOTICE 'Add primary key to parks_regions (park_code, region_cd).';


--    DROP VIEW IF EXISTS nps.v_parks_single_region;
--    CREATE VIEW
--      nps.v_parks_single_region AS
--    SELECT
--      *
--    FROM
--      nps.parks_regions
--    WHERE
--      park_code NOT IN
--        (
--        SELECT DISTINCT
--          park_code
--        FROM
--          nps.parks_regions
--        WHERE area_rank > 1
--        );
--    RAISE NOTICE 'Recreate view v_parks_single_region to find parks that occur in only one
region.';
--
--
--    UPDATE
```

```
--       nps.plots_xls AS p
--    SET
--       region_cd_bnd = r.region_cd
--    FROM
--       nps.v_parks_single_region AS r
--    WHERE
--       LOWER(LEFT(p.park_code, 4)) = LOWER(r.park_code)
--       OR
--       (LOWER(p.park_code) = 'seki' AND LOWER(r.park_code) = 'sequ');
--    GET DIAGNOSTICS rows_affected = ROW_COUNT;
--    RAISE NOTICE 'Update plots with region for parks in a single region, including parks with
"double" codes (lamr_alfl, pore_goga) and seki is Sequoia (sequ) and Kings Canyon (kica)
Parks.
--    % rows affected.', rows_affected;

   -- now prefer region cd derived from plot coordinates (in summary views); fall back to this
field: the majority region in each park
   UPDATE
      nps.plots_xls AS p
   SET
      region_cd_bnd = r.region_cd
   FROM
      nps.parks_regions AS r
   WHERE
      r.area_rank = 1
      AND
      (LOWER(LEFT(p.park_code, 4)) = LOWER(r.park_code)
      OR
      (LOWER(p.park_code) = 'seki' AND LOWER(r.park_code) = 'sequ'));
   GET DIAGNOSTICS rows_affected = ROW_COUNT;
   RAISE NOTICE 'Update plots with region for majority region, including parks with "double"
codes (lamr_alfl, pore_goga) and seki is Sequoia (sequ) and Kings Canyon (kica) Parks.
   % rows affected.', rows_affected;


   DROP TABLE IF EXISTS nps.parks_states CASCADE;
   CREATE TABLE nps.parks_states AS
   SELECT
      LOWER(b.unit_code) AS park_code,
      s.state_fips,
      ST_Area(ST_Multi(ST_Union(ST_Intersection(b.geom, s.geom)))) AS area,
      rank() OVER (PARTITION BY b.unit_code ORDER BY
ST_Area(ST_Multi(ST_Union(ST_Intersection(b.geom, s.geom)))) DESC) AS area_rank,
      ST_Multi(ST_Union(ST_Intersection(b.geom, s.geom))) AS geom
```

```
  FROM
    nps.nps_boundary b
    INNER JOIN public.statep010 s
      ON ST_Intersects(b.geom, s.geom)
  GROUP BY
    b.unit_code,
    s.state_fips
  ORDER BY
    b.unit_code,
    area_rank;
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Create/replace parks_states table and populate with intersection of park
boundaries and states.
  % rows affected.', rows_affected;


  ALTER TABLE nps.parks_states ADD CONSTRAINT parks_states_pkey PRIMARY
KEY (park_code, state_fips);
  RAISE NOTICE 'Add primary key to parks_states (park_code, state_fips).';


  -- prefer state_fips derived from plot coordinates (in summary views)?; fall back to this field:
the majority state in each park
  UPDATE
    nps.plots_xls AS p
  SET
    state_fips_bnd = s.state_fips
  FROM
    nps.parks_states AS s
  WHERE
    s.area_rank = 1
    AND
    (LOWER(LEFT(p.park_code, 4)) = LOWER(s.park_code)
    OR
    (LOWER(p.park_code) = 'seki' AND LOWER(s.park_code) = 'sequ'));
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Update plots with state_fips for majority state, including parks with
"double" codes (lamr_alfl, pore_goga) and seki is Sequoia (sequ) and Kings Canyon (kica)
Parks.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls
```

```
SET
    gps_datum = upper(gps_datum),
    utm_x_field = lower(utm_x_field),
    utm_y_field = lower(utm_y_field),
    utm_x_corrected = lower(utm_x_corrected),
    utm_y_corrected = lower(utm_y_corrected);
/* needed for utm coordinate fields to simplify later correction for letters, e.g. 'o' */
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Normalize case on geographic fields.
% rows affected.', rows_affected;


UPDATE
    nps.plots_xls
SET
    utm_x_field = replace(utm_x_field, 'o', '0'),
    utm_y_field = replace(utm_y_field, 'o', '0'),
    utm_x_corrected = replace(utm_x_corrected, 'o', '0'),
    utm_y_corrected = replace(utm_y_corrected, 'o', '0')
WHERE
    CONCAT(utm_x_field, utm_y_field, utm_x_corrected, utm_y_corrected) LIKE '%o%';
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Correct letter "o" to number zero in utm coordinate fields.
% rows affected.', rows_affected;


UPDATE
    nps.plots_xls
SET
    gps_datum =
        CASE
            WHEN gps_datum IN ('NAD 1983', 'NAD 83') THEN
                'NAD83'
            WHEN gps_datum IS NULL THEN
                'WGS84'
            ELSE
                gps_datum
        END;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Normalize geodetic datum values, e.g. "NAD 1983" and "NAD 83" equal
"NAD83".
% rows affected.', rows_affected;
```

```
UPDATE
   nps.plots_xls
SET
   gps_datum = NULL
WHERE
   gps_datum = '(N/A)'; -- datum now always uppercase
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Update invalid ("(N/A)") geodetic datum to null.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   utm_x_field = NULL,
   utm_y_field = NULL
WHERE
   utm_x_field IN ('(n/a)', '0.0') -- coordinates now always lowercase
   OR utm_y_field IN ('(n/a)', '0.0');
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Update utm coordinate pairs to null if either coordinate is invalid.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   utm_x_corrected = NULL,
   utm_y_corrected = NULL
WHERE
   utm_x_corrected IN ('(n/a)', '0.0') -- coordinates now always lowercase
   OR utm_y_corrected IN ('(n/a)', '0.0');
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Update "corrected" utm coordinate pairs to null if either coordinate is
invalid.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   utm_zone =
      CASE park_code
      WHEN 'agfo' THEN '13'
```

```
        WHEN 'buff' THEN '15'
        WHEN 'cong' THEN '17'
        WHEN 'fiis' THEN '18'
        WHEN 'flfo' THEN '13'
        WHEN 'grsm' THEN '17' -- a very small piece is in zone 16
        WHEN 'hafo' THEN '11' -- incorrectly recorded as 12 in the original data
        WHEN 'meve' THEN '12'
        WHEN 'nava' THEN '12' -- incorrectly recorded as 13 in the some of the original data
        WHEN 'rocr' THEN '18'
        WHEN 'slbe' THEN '16'
        WHEN 'thro' THEN '13'
        WHEN 'waca' THEN '12'
        WHEN 'wefa' THEN '18'
        WHEN 'wica' THEN '13'
        WHEN 'wupa' THEN '12'
        WHEN 'lamr_alfl' THEN '14'
        WHEN 'pore_goga' THEN '10'
        END
    WHERE
      utm_zone IS NULL
      OR (park_code = 'hafo' and utm_zone <> '11')
      OR (park_code = 'nava' and utm_zone <> '12')
      OR (park_code IN ('cong','grsm') and utm_zone <> '17');
    GET DIAGNOSTICS rows_affected = ROW_COUNT;
    RAISE NOTICE 'Manually correct missing utm zones.
    % rows affected.', rows_affected;


    UPDATE
      nps.plots_xls
    SET
      utm_x_field = utm_y_field,
      utm_y_field = utm_x_field
    WHERE
      utm_x_field LIKE '39%'
      AND utm_y_field LIKE '2%'
      AND park_code = 'chcu';
    GET DIAGNOSTICS rows_affected = ROW_COUNT;
    RAISE NOTICE 'Fix reversed x and y utm fields in Chaco Culture National Historical Park
  (chcu) plots.
    % rows affected.', rows_affected;


    UPDATE
```

```
    nps.plots_xls
  SET
    utm_y_field = '3995034'
  WHERE
    utm_y_field = '2995034'
    AND park_code = 'chcu';
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Fix a typographical error in a Chaco Culture National Historical Park (chcu) plot.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls
  SET
    utm_y_field = '3793551.0'
  WHERE
    utm_y_field = '4163551.0'
    AND park_code = 'liri';
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Fix a typographical error in a liri plot.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls
  SET
    utm_y_field = '3935896.0'
  WHERE
    utm_y_field = '3985896.0'
    AND park_code = 'grsm';
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Fix a typographical error in a liri plot.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls
  SET
    utm_y_field = '4211872.0'
  WHERE
    utm_y_field = '4311872.0'
    AND park_code = 'yose';
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
```

```
RAISE NOTICE 'Fix a typographical error in a yose plot.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   utm_x_field = '259200.0'
WHERE
   utm_x_field = '599200.0'
   AND park_code = 'lamr_alfl';
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix a typographical error in a lamr_alfl plot.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   utm_y_field = '4824856.0'
WHERE
   utm_y_field = '4324856.0'
   AND park_code = 'badl';
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix a typographical error in a badl plot.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   lat_field = '35.9846'
WHERE
   lat_field = '39.9846'
   AND park_code = 'buff';
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix typographical error in a Buffalo National River (buff) plot.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   lat_field = '36.1059'
```

```
WHERE
    lat_field = '35.1059'
    AND park_code = 'buff';
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix typographical error in a Buffalo National River (buff) plot.
% rows affected.', rows_affected;


UPDATE
    nps.plots_xls
SET
    lon_field = '93.35045'
WHERE
    lon_field = '93.93045'
    AND park_code = 'buff';
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix typographical error in a Buffalo National River (buff) plot.
% rows affected.', rows_affected;


UPDATE
    nps.plots_xls
SET
    utm_x_field = CAST(utm_x_field AS NUMERIC) / 10
WHERE
    CAST(utm_x_field AS NUMERIC) > 999999;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix utm_x_field values too large by an order of magnitude.
% rows affected.', rows_affected;


UPDATE
    nps.plots_xls
SET
    utm_x_field = CAST(utm_x_field AS NUMERIC) * 10
WHERE
    CAST(utm_x_field AS NUMERIC) < 99999;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix utm_y_field values too small by an order of magnitude.
% rows affected.', rows_affected;


UPDATE
    nps.plots_xls
```

```
SET
   utm_y_field = CAST(utm_y_field AS NUMERIC) / 10
WHERE
   CAST(utm_y_field AS NUMERIC) > 9999999;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix utm_y_field values too large by an order of magnitude.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls
SET
   utm_y_field = CAST(utm_y_field AS NUMERIC) * 10
WHERE
   CAST(utm_y_field AS NUMERIC) < 999999;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Fix utm_y_field values too small by an order of magnitude.
% rows affected.', rows_affected;


UPDATE
   plots_xls
SET
   lat_field =
      CAST(split_part(lat_field, ' ', 1) AS NUMERIC) +
      (CAST(split_part(lat_field, ' ', 2) AS NUMERIC) / 60) +
      (CAST(split_part(lat_field, ' ', 3) AS NUMERIC) / 3600),
   lon_field =
      CAST(0 || split_part(lon_field, ' ', 1) AS NUMERIC) +
      (CAST(0 || split_part(lon_field, ' ', 2) AS NUMERIC) / 60) +
      (CAST(0 || split_part(lon_field, ' ', 3) AS NUMERIC) / 3600)
WHERE
   lat_field IS NOT NULL
   AND lon_field IS NOT NULL
   AND split_part(lat_field, ' ', 2) <> ''
   AND split_part(lat_field, ' ', 3) <> ''
   AND split_part(lon_field, ' ', 2) <> ''
   AND split_part(lon_field, ' ', 3) <> '';
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Update lat/lon in DMS to decimal degrees.
% rows affected.', rows_affected;


UPDATE
```

```
    nps.plots_xls
SET
    geom =
        ST_Transform(
            ST_GeomFromText(
                'POINT(' ||
                CASE
                WHEN SUBSTRING(lon_field FROM 1 FOR 1) <> '-' THEN '-'
                ELSE ''
                END ||
                lon_field || ' ' ||
                lat_field || ')',
                CAST(
                    CASE gps_datum -- find epsg code
                    WHEN 'NAD83' THEN
                        '4269'
                    WHEN 'WGS84' THEN
                        '4326'
                    END
                AS INTEGER)
            ),
        2163)
WHERE
    lat_field IS NOT NULL
    AND lon_field IS NOT NULL
    AND CAST(lat_field AS NUMERIC) BETWEEN -90 AND 90
    AND CAST(lon_field AS NUMERIC) BETWEEN -180 AND 180
    AND geom IS NULL;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Copy lat/lon to geom and make longitude negative.
% rows affected.', rows_affected;


UPDATE
    nps.plots_xls
SET
    geom =
        ST_Transform(
            ST_GeomFromText(
                'POINT(' ||
                COALESCE(utm_x_corrected, utm_x_field) ||
                ' ' ||
                COALESCE(utm_y_corrected, utm_y_field) ||
                ')',
```

```
            CAST(
               CASE gps_datum -- set epsg code
               WHEN 'NAD83' THEN
                  '269'
               WHEN 'NAD27' THEN
                  '267'
               WHEN 'WGS84' THEN
                  '326'
               END ||
               SUBSTRING(utm_zone FROM 1 FOR 2)
            AS INTEGER)
         ),
      2163)
WHERE
   SUBSTRING(utm_zone FROM 1 FOR 1) IN ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')
   AND
   SUBSTRING(utm_zone FROM 2 FOR 1) IN ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')
   AND
   (
      (utm_x_field IS NOT NULL AND utm_y_field IS NOT NULL)
      OR
      (utm_x_corrected IS NOT NULL AND utm_y_corrected IS NOT NULL)
   )
   AND geom IS NULL;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Copy utm to geom and set EPSG code based on declared datum.
% rows affected.', rows_affected;


UPDATE
   nps.plots_xls AS p
SET
   region_cd_plot = m.region_cd
FROM
   public.mlra_v42 AS m
WHERE
   ST_Covers(m.geom, p.geom);
-- full update using PostGIS only
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Set region to region_gid from mrla polygons.
% rows affected.', rows_affected;


UPDATE
```

```
    nps.plots_xls AS p
  SET
    region_cd_plot = n.region_cd
  FROM
    (
    SELECT DISTINCT ON
      (p.recid)
      p.recid,
      m.region_cd
    FROM
      nps.plots_xls AS p,
      public.mlra_v42 AS m
    WHERE
      p.region_cd_plot IS NULL
      AND ST_DWithin(m.geom, p.geom, 30000)
    ORDER BY
      p.recid,
      ST_Distance(m.geom, p.geom)
    ) AS n
  WHERE
    p.recid = n.recid;
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Catch points outside of mlra polygons, e.g. near coastlines, islands. Only
null lat/lon remain.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls AS p
  SET
    park_code_bnd = LOWER(b.unit_code)
  FROM
    nps.nps_boundary AS b
  WHERE
    ST_Covers(b.geom, p.geom);
  -- full update using PostGIS only
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Set park_code_bnd for plots covered by park boundary polygons.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls AS p
  SET
```

```
    park_code_bnd = LOWER(c.unit_code)
  FROM
    (
    SELECT DISTINCT ON
      (p.recid)
      p.recid,
      b.unit_code
    FROM
      nps.plots_xls AS p,
      nps.nps_boundary AS b
    WHERE
      p.park_code_bnd IS NULL
      AND ST_DWithin(b.geom, p.geom, 30000)
    ORDER BY
      p.recid,
      ST_Distance(b.geom, p.geom)
    ) AS c
  WHERE
    p.recid = c.recid;
  -- full update using PostGIS only
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Set park_cd_bnd to nearest park for plots NOT covered by park boundary
polygons.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls AS p
  SET
    state_fips_plot = s.state_fips
  FROM
    public.statep010 AS s
  WHERE
    ST_Covers(s.geom, p.geom);
  -- full update using PostGIS only
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Set state_cd to fips code from covering statep010 polygons.
  % rows affected.', rows_affected;


  UPDATE
    nps.plots_xls AS p
  SET
    state_fips_plot = s2.state_fips
```

```
    FROM
        (
        SELECT DISTINCT ON
            (p.recid)
            p.recid,
            s.state_fips
        FROM
            nps.plots_xls AS p,
            public.statep010 AS s
        WHERE
            p.state_fips_plot IS NULL
            AND ST_DWithin(s.geom, p.geom, 30000)
        ORDER BY
            p.recid,
            ST_Distance(s.geom, p.geom)
        ) AS s2
    WHERE
        p.recid = s2.recid;
    -- full update using PostGIS only
    GET DIAGNOSTICS rows_affected = ROW_COUNT;
    RAISE NOTICE 'Set state_cd to nearest state for plots NOT covered by statep010 polygons,
e.g. near coastlines, islands.
    % rows affected.', rows_affected;


 --    UPDATE
--         nps.plots_xls AS p
--     SET
--         state_cd = s.state_fips
--     FROM
--         public.states_fips AS s,
--         nps.nps_boundary AS b
--     WHERE
--         p.state_cd IS NULL
--         AND
--         (LOWER(LEFT(p.park_code, 4)) = LOWER(b.unit_code)
--         OR
--         (LOWER(p.park_code) = 'seki' AND LOWER(b.unit_code) = 'sequ'))
--         AND
--         s.stusab = UPPER(b.state);
--     GET DIAGNOSTICS rows_affected = ROW_COUNT;
--     RAISE NOTICE 'Set state_cd to fips code for plots with no coordinates.
--     % rows affected.', rows_affected;
```

```
 END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Appendix 14 – SQL program to create NPS summaries

filename: nps_summaries.sql

```
CREATE OR REPLACE FUNCTION nps.nps_summaries()
   RETURNS void AS
$BODY$
DECLARE
   rows_affected integer := 0;
BEGIN
   SET search_path TO nps;
   RAISE NOTICE 'Total expected runtime: ~106 seconds.';


   -- normalize species
   ALTER TABLE sp_cov_xls
      DROP COLUMN IF EXISTS accepted_symbol CASCADE,
      DROP COLUMN IF EXISTS accepted_symbol2 CASCADE;
   GET DIAGNOSTICS rows_affected = ROW_COUNT;
   RAISE NOTICE 'Drop accepted_symbol(2) columns from plots table.';


   ALTER TABLE sp_cov_xls
      ADD COLUMN accepted_symbol TEXT,
      ADD COLUMN accepted_symbol2 TEXT;
   GET DIAGNOSTICS rows_affected = ROW_COUNT;
   RAISE NOTICE 'Add accepted_symbol(2) columns from plots table.';


   -- convert plant_symbol to usda_plants accepted symbol
   UPDATE
      sp_cov_xls AS s
   SET
      accepted_symbol = spp.accepted_symbol
   FROM
      public.usda_plants AS spp
   WHERE
      upper(s.plant_symbol) = spp.symbol
      AND s.plant_symbol IS NOT NULL
      AND s.accepted_symbol IS NULL;
   GET DIAGNOSTICS rows_affected = ROW_COUNT;
   RAISE NOTICE 'Get accepted_symbol from PLANTS by matching on sp_cov symbol.
   % rows affected.', rows_affected;
```

```
-- convert itis_tsn to usda_plants accepted symbol
UPDATE
   sp_cov_xls AS s
SET
   accepted_symbol2 = spp.accepted_symbol
FROM
   public.usda_plants AS spp
WHERE
   CAST(CAST(s.itis_tsn AS NUMERIC) AS INTEGER) = CAST(spp.itis_tsn AS
INTEGER)
   AND spp.itis_tsn <> ''
   AND s.itis_tsn IS NOT NULL
   AND s.accepted_symbol2 IS NULL;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Get accepted_symbol from USDA PLANTS by matching on itis_tsn.
% rows affected.', rows_affected;


DROP VIEW IF EXISTS v_taxa CASCADE;
CREATE VIEW v_taxa AS
select
   s.recid AS sp_recid,
   s.park_code,
   s.plot_code,
   s.plot_event,
   s.sci_name,
   s.com_name,
   s.sci_family,
   CONCAT_WS(' ',
      spp.genus,
      spp.species) AS usda_taxon
FROM
   sp_cov_xls AS s
   LEFT OUTER JOIN public.usda_plants AS spp
      ON COALESCE(s.accepted_symbol, s.accepted_symbol2) = spp.symbol;
RAISE NOTICE 'Add USDA PLANTS columns to sp_cov rows.';


DROP VIEW IF EXISTS v_plots_normalize CASCADE;
CREATE VIEW v_plots_normalize AS
SELECT
   recid,
   park_code,
```

```
        park_code_bnd,
        plot_event,
        plot_code,
        COALESCE(region_cd_plot, region_cd_bnd) AS region_cd,
        COALESCE(state_fips_plot, s.state_fips, state_fips_bnd) AS state_fips, --state from plot coords, recorded state, park boundary
        CASE
        WHEN comm_type = 'wetland' OR cowardin IN ('marine', 'estuarine', 'riverine', 'lacustrine', 'palustrine') THEN
          'w'
        WHEN comm_type = 'upland' OR cowardin = 'upland' THEN
          'u'
        ELSE
          'N/A'
        END AS wetland_cd,
        CASE
        WHEN lower(plot_shape) IN ('rectamgi;ar', 'rectangle', 'rectangular', 'square') THEN
          'rectangle'
        WHEN lower(plot_shape) IN ('circle', 'circular') THEN
          'circle'
        ELSE
          lower(plot_shape)
        END as plot_shape,
        CAST(COALESCE(plot_radius, '0') AS NUMERIC) AS plot_radius,
        CAST(COALESCE(plot_diam, '0') AS NUMERIC) AS plot_diam,
        CAST(COALESCE(x_dim, '0') AS NUMERIC) AS x_dim,
        CAST(COALESCE(y_dim, '0') AS NUMERIC) AS y_dim
    FROM
        plots_xls AS p
        LEFT OUTER JOIN public.states_fips s
          ON UPPER(p.state) = s.stusab;
    RAISE NOTICE 'normalize plot fields, e.g. radius to numeric, add region and wetland/upland status to plots, etc.';


    DROP VIEW IF EXISTS v_plots_species CASCADE;
    CREATE VIEW v_plots_species AS
    SELECT
        p.recid AS p_recid,
        park_code_bnd, -- park_code is pulled in by s.*
        region_cd,
        state_fips,
        wetland_cd,
        plot_shape,
```

```
    plot_radius,
    plot_diam,
    x_dim,
    y_dim,
    s.*,
    CASE -- cannot use coalesce
    WHEN s.usda_taxon IS NOT NULL AND s.usda_taxon <> '' THEN
      usda_taxon
    ELSE
      CONCAT_WS(' ',
        split_part(s.sci_name, ' ', 1),
        split_part(s.sci_name, ' ', 2))
    END AS taxon
  FROM
    v_plots_normalize AS p
    INNER JOIN v_taxa AS s
      ON p.park_code = s.park_code
      AND ((p.plot_event = s.plot_event AND s.plot_code IS NULL)
      OR (p.plot_code = s.plot_code AND s.plot_event IS NULL));
  RAISE NOTICE 'Join plots records to species records and add scientific name (taxon)
preferring name derived from USDA PLANTS symbol, then ITIS, then sp_cov.';


  DROP TABLE IF EXISTS spp_by_region CASCADE;
  CREATE TABLE spp_by_region
  (
  region_cd integer,
  taxon TEXT,
  genus TEXT,
  species TEXT,
  common_name TEXT,
  wetland_cd VARCHAR(3),
  plot_count NUMERIC
  );

  --tmp_ prefix avoids ambiguity during grouping
  INSERT INTO spp_by_region
  SELECT
    ps.region_cd,
    ps.taxon,
    split_part(ps.taxon, ' ', 1) AS tmp_genus,
    split_part(ps.taxon, ' ', 2) AS tmp_species,
    spp.common_name AS tmp_common_name,
    ps.wetland_cd,
```

```
    COUNT(ps.p_recid) AS plot_count
  FROM
    v_plots_species ps
    INNER JOIN public.usda_plants AS spp
    ON split_part(ps.taxon, ' ', 1) = spp.genus
    AND split_part(ps.taxon, ' ', 2) = spp.species
    AND spp.symbol = spp.accepted_symbol
    AND spp.genus <> ''
    AND spp.species <> ''
    AND spp.subspecies = ''
    AND spp.variety = ''
    AND spp.subvariety = ''
    AND spp.forma = ''
    AND split_part(taxon, ' ', 2) <> ''
  GROUP BY
    region_cd,
    taxon,
    tmp_genus,
    tmp_species,
    tmp_common_name,
    wetland_cd
  ORDER BY
    region_cd,
    taxon,
    wetland_cd;
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Create summary table of species by regions.
  % rows affected.', rows_affected;


  -- add pkey AFTER insert to speed up insert
  ALTER TABLE spp_by_region ADD CONSTRAINT spp_by_region_pkey PRIMARY
KEY (region_cd, taxon, wetland_cd);


  DROP TABLE IF EXISTS spp_arid_west_minus_ca CASCADE;
  CREATE TABLE spp_arid_west_minus_ca
  (
  region_cd integer,
  taxon TEXT,
  genus TEXT,
  species TEXT,
  common_name TEXT,
  wetland_cd VARCHAR(3),
```

```
plot_count NUMERIC
);

-- count spp by Arid West region minus CA, wetland code
--tmp_ prefix avoids ambiguity during grouping
INSERT INTO spp_arid_west_minus_ca
SELECT
   ps.region_cd,
   ps.taxon,
   split_part(ps.taxon, ' ', 1) AS tmp_genus,
   split_part(ps.taxon, ' ', 2) AS tmp_species,
   spp.common_name AS tmp_common_name,
   ps.wetland_cd,
   COUNT(ps.p_recid) AS plot_count
FROM
   v_plots_species ps
   INNER JOIN public.usda_plants AS spp
   ON split_part(ps.taxon, ' ', 1) = spp.genus
   AND split_part(ps.taxon, ' ', 2) = spp.species
   AND spp.symbol = spp.accepted_symbol
   AND spp.genus <> ''
   AND spp.species <> ''
   AND spp.subspecies = ''
   AND spp.variety = ''
   AND spp.subvariety = ''
   AND spp.forma = ''
   AND split_part(taxon, ' ', 2) <> ''
WHERE
   ps.region_cd = 5 --'aw'
   AND state_fips <> '06'
GROUP BY
   region_cd,
   taxon,
   tmp_genus,
   tmp_species,
   tmp_common_name,
   wetland_cd
ORDER BY
   region_cd,
   taxon,
   wetland_cd;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Create summary table of species in the Arid West region minus CA.
% rows affected.', rows_affected;
```

```
-- add pkey AFTER insert to speed up insert
ALTER TABLE spp_arid_west_minus_ca ADD CONSTRAINT
spp_arid_west_minus_ca_pkey PRIMARY KEY (region_cd, taxon, wetland_cd);


DROP TABLE IF EXISTS spp_by_state CASCADE;
CREATE TABLE spp_by_state
(
state_fips TEXT,
taxon TEXT,
genus TEXT,
species TEXT,
common_name TEXT,
wetland_cd VARCHAR(3),
plot_count NUMERIC
);


--tmp_ prefix avoids ambiguity during grouping
INSERT INTO spp_by_state
SELECT
    ps.state_fips,
    ps.taxon,
    split_part(ps.taxon, ' ', 1) AS tmp_genus,
    split_part(ps.taxon, ' ', 2) AS tmp_species,
    spp.common_name AS tmp_common_name,
    ps.wetland_cd,
    COUNT(ps.p_recid) AS plot_count
FROM
    v_plots_species ps
    INNER JOIN public.usda_plants AS spp
    ON split_part(ps.taxon, ' ', 1) = spp.genus
    AND split_part(ps.taxon, ' ', 2) = spp.species
    AND spp.symbol = spp.accepted_symbol
    AND spp.genus <> ''
    AND spp.species <> ''
    AND spp.subspecies = ''
    AND spp.variety = ''
    AND spp.subvariety = ''
    AND spp.forma = ''
    AND split_part(taxon, ' ', 2) <> ''
GROUP BY
```

```
    state_fips,
    taxon,
    tmp_genus,
    tmp_species,
    tmp_common_name,
    wetland_cd
  ORDER BY
    state_fips,
    taxon,
    wetland_cd;
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Create summary table of species by regions.
  % rows affected.', rows_affected;


  -- add pkey AFTER insert to speed up insert
  ALTER TABLE spp_by_state ADD CONSTRAINT spp_by_state_pkey PRIMARY KEY
(state_fips, taxon, wetland_cd);


  DROP TABLE IF EXISTS plots_area CASCADE;
  CREATE TABLE plots_area
  (
  recid TEXT,
  region_cd INTEGER,
  park_code TEXT,
  park_code_bnd TEXT,
  state_fips TEXT,
  wetland_cd VARCHAR(3),
  plot_shape TEXT,
  plot_radius NUMERIC,
  plot_diam NUMERIC,
  x_dim NUMERIC,
  y_dim NUMERIC,
  area NUMERIC
  );


  INSERT INTO plots_area
  SELECT
    recid,
    region_cd,
    park_code,
    park_code_bnd,
```

```
    state_fips,
    wetland_cd,
    plot_shape,
    plot_radius,
    plot_diam,
    x_dim,
    y_dim,
    0 as area
  FROM
    v_plots_normalize;
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Create table of plot areas.
  % rows affected.', rows_affected;


  -- add pkey AFTER insert to speed up insert
  ALTER TABLE plots_area ADD CONSTRAINT plots_area_pkey PRIMARY KEY
(recid);


  -- calculate area using best available data
  UPDATE
    plots_area
  SET
    area = pi() *
      CASE
      WHEN plot_radius > 0 THEN plot_radius
      WHEN plot_diam > 0 THEN plot_diam / 2
      ELSE GREATEST(x_dim, y_dim) / 2
      END ^ 2
  WHERE
    area = 0
    AND plot_shape = 'circle'
    AND (plot_radius > 0
      OR plot_diam > 0
      OR x_dim > 0
      OR y_dim > 0);
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Calculate area for circular plots.
  % rows affected.', rows_affected;


  UPDATE
    plots_area
```

```
SET
    area = pi() * x_dim * y_dim
WHERE
    area = 0
    AND plot_shape = 'oval'
    AND x_dim > 0
    AND y_dim > 0;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Calculate area for oval plots.
% rows affected.', rows_affected;


UPDATE
    plots_area
SET
    area = x_dim * y_dim
WHERE
    area = 0
    AND plot_shape = 'rectangle'
    AND x_dim > 0
    AND y_dim > 0;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Calculate area for rectangular plots.
% rows affected.', rows_affected;


UPDATE
    plots_area
SET
    area = x_dim * y_dim
WHERE
    area = 0
    AND x_dim > 0
    AND y_dim > 0;
GET DIAGNOSTICS rows_affected = ROW_COUNT;
RAISE NOTICE 'Calculate area for possibly rectangular plots.
% rows affected.', rows_affected;


UPDATE
    plots_area
SET
    area = GREATEST(x_dim, y_dim) ^ 2
WHERE
```

```
    area = 0
    AND (x_dim > 0
      OR y_dim > 0);
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Calculate area for possibly square plots.
  % rows affected.', rows_affected;


  UPDATE
    plots_area
  SET
    area = plot_diam ^ 2
  WHERE
    area = 0
    AND plot_shape = 'rectangle'
    AND plot_diam > 0;
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Calculate area for rectangular plots with a diameter value.
  % rows affected.', rows_affected;


  UPDATE
    plots_area
  SET
    area = pi() * GREATEST(plot_radius, plot_diam / 2) ^ 2
  WHERE
    area = 0
    AND (plot_radius > 0
      OR plot_diam > 0);
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Calculate area for possibly circular plots.
  % rows affected.', rows_affected;


  UPDATE
    plots_area
  SET
    area = 1.0
  WHERE
    area = 0
    AND GREATEST(plot_radius, plot_diam, x_dim, y_dim) = 0;
  GET DIAGNOSTICS rows_affected = ROW_COUNT;
  RAISE NOTICE 'Finally, set area to 1.0 for plots with no dimensions. Allows use of data but
influence is low.
```

```
% rows affected.', rows_affected;


DROP VIEW IF EXISTS v_area_by_region CASCADE;
CREATE VIEW v_area_by_region AS
SELECT
   region_cd,
   wetland_cd,
   SUM(area) as area
FROM
   plots_area
GROUP BY
   region_cd,
   wetland_cd
ORDER BY
   region_cd,
   wetland_cd;
RAISE NOTICE 'Create view to sum area by region, wetland_cd.';


DROP VIEW IF EXISTS v_area_arid_west_minus_ca CASCADE;
CREATE VIEW v_area_arid_west_minus_ca AS
SELECT
   region_cd,
   wetland_cd,
   SUM(area) as area
FROM
   plots_area
WHERE
   region_cd = 5 --'aw'
   AND state_fips <> '06'
GROUP BY
   region_cd,
   wetland_cd
ORDER BY
   region_cd,
   wetland_cd;
RAISE NOTICE 'Create view to sum area in arid west minus CA, wetland_cd.';


DROP VIEW IF EXISTS v_area_by_state CASCADE;
CREATE VIEW v_area_by_state AS
SELECT
   state_fips,
```

```
    wetland_cd,
    SUM(area) as area
FROM
    plots_area
GROUP BY
    state_fips,
    wetland_cd
ORDER BY
    state_fips,
    wetland_cd;
RAISE NOTICE 'Create view to sum area by state, wetland_cd.';


DROP VIEW IF EXISTS v_area_by_region_flat CASCADE;
CREATE VIEW v_area_by_region_flat AS
SELECT
    region_cd,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            area
        ELSE
            0
        END
    ) AS ndotw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            area
        ELSE
            0
        END
    ) AS ndotu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            area
        ELSE
            0
        END
    ) AS ndotother
FROM
    v_area_by_region
GROUP BY
```

```
    region_cd;
RAISE NOTICE 'Create flattened view of area by region, wetland_cd.';


DROP VIEW IF EXISTS v_area_arid_west_minus_ca_flat CASCADE;
CREATE VIEW v_area_arid_west_minus_ca_flat AS
SELECT
    region_cd,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            area
        ELSE
            0
        END
    ) AS ndotw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            area
        ELSE
            0
        END
    ) AS ndotu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            area
        ELSE
            0
        END
    ) AS ndotother
FROM
    v_area_arid_west_minus_ca
GROUP BY
    region_cd;
RAISE NOTICE 'Create flattened view of area in arid west minus CA, wetland_cd.';


DROP VIEW IF EXISTS v_area_by_state_flat CASCADE;
CREATE VIEW v_area_by_state_flat AS
SELECT
    state_fips,
    SUM(
```

```
        CASE
        WHEN wetland_cd = 'w' THEN
            area
        ELSE
            0
        END
    ) AS ndotw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            area
        ELSE
            0
        END
    ) AS ndotu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            area
        ELSE
            0
        END
    ) AS ndotother
FROM
    v_area_by_state
GROUP BY
    state_fips;
RAISE NOTICE 'Create flattened view of area by state, wetland_cd.';


DROP VIEW IF EXISTS v_spp_by_region_flat CASCADE;
CREATE VIEW v_spp_by_region_flat AS
SELECT
    taxon,
    genus,
    species,
    common_name,
    region_cd,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            plot_count
        ELSE
            0
```

```
      END
    ) AS npw,
    SUM(
      CASE
      WHEN wetland_cd = 'u' THEN
        plot_count
      ELSE
        0
      END
    ) AS npu,
    SUM(
      CASE
      WHEN wetland_cd NOT IN ('w', 'u') THEN
        plot_count
      ELSE
        0
      END
    ) AS npother
FROM
    spp_by_region
GROUP BY
    taxon,
    genus,
    species,
    common_name,
    region_cd;
RAISE NOTICE 'Create flattened view of species by regions.';


DROP VIEW IF EXISTS v_spp_arid_west_minus_ca_flat CASCADE;
CREATE VIEW v_spp_arid_west_minus_ca_flat AS
SELECT
    taxon,
    genus,
    species,
    common_name,
    region_cd,
    SUM(
      CASE
      WHEN wetland_cd = 'w' THEN
        plot_count
      ELSE
        0
      END
```

```
    ) AS npw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            plot_count
        ELSE
            0
        END
    ) AS npu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            plot_count
        ELSE
            0
        END
    ) AS npother
FROM
    spp_arid_west_minus_ca
GROUP BY
    taxon,
    genus,
    species,
    common_name,
    region_cd;
RAISE NOTICE 'Create flattened view of species in Arid West region minus CA.';


DROP VIEW IF EXISTS v_spp_by_state_flat CASCADE;
CREATE VIEW v_spp_by_state_flat AS
SELECT
    taxon,
    genus,
    species,
    common_name,
    state_fips,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            plot_count
        ELSE
            0
        END
    ) AS npw,
```

```
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            plot_count
        ELSE
            0
        END
    ) AS npu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            plot_count
        ELSE
            0
        END
    ) AS npother
FROM
    spp_by_state
GROUP BY
    taxon,
    genus,
    species,
    common_name,
    state_fips;
RAISE NOTICE 'Create flattened view of species by state.';


DROP VIEW IF EXISTS v_spp_freq_region CASCADE;
CREATE VIEW v_spp_freq_region AS
SELECT
    s.*,
    a.ndotw,
    a.ndotu,
    a.ndotother,
    CASE
    WHEN s.npw + s.npu = 0 THEN
        NULL
    ELSE
        s.npw / (s.npw + s.npu)
    END AS wetland_freq_unadj,
    CASE
    WHEN s.npw + s.npu = 0 THEN
        NULL
    WHEN s.npw = 0 THEN
```

```
      0
   ELSE
      1 / (1 + (s.npu * a.ndotw / (s.npw * a.ndotu)))
   END AS wetland_freq_adj,
   CASE
   WHEN s.npw + s.npu = 0 THEN
      NULL
   ELSE
      0.98 * SQRT(1 / (s.npw + s.npu))
   END AS max_moe_at95cl,
   CASE
   WHEN s.npw + s.npu = 0 THEN
      NULL
   ELSE
      0.82 * SQRT(1 / (s.npw + s.npu))
   END AS max_moe_at90cl
FROM
   v_spp_by_region_flat AS s,
   v_area_by_region_flat AS a
WHERE
   s.region_cd = a.region_cd;
RAISE NOTICE 'Create view of frequency of species by regions.';


DROP VIEW IF EXISTS v_spp_freq_arid_west_minus_ca CASCADE;
CREATE VIEW v_spp_freq_arid_west_minus_ca AS
SELECT
   s.*,
   a.ndotw,
   a.ndotu,
   a.ndotother,
   CASE
   WHEN s.npw + s.npu = 0 THEN
      NULL
   ELSE
      s.npw / (s.npw + s.npu)
   END AS wetland_freq_unadj,
   CASE
   WHEN s.npw + s.npu = 0 THEN
     NULL
   WHEN s.npw = 0 THEN
     0
   ELSE
     1 / (1 + (s.npu * a.ndotw / (s.npw * a.ndotu)))
```

```
    END AS wetland_freq_adj,
    CASE
    WHEN s.npw + s.npu = 0 THEN
      NULL
    ELSE
      0.98 * SQRT(1 / (s.npw + s.npu))
    END AS max_moe_at95cl,
    CASE
    WHEN s.npw + s.npu = 0 THEN
      NULL
    ELSE
      0.82 * SQRT(1 / (s.npw + s.npu))
    END AS max_moe_at90cl
FROM
    v_spp_arid_west_minus_ca_flat AS s,
    v_area_arid_west_minus_ca_flat AS a
WHERE
    s.region_cd = a.region_cd
;
RAISE NOTICE 'Create view of frequency of species in the Arid West region minus CA.';


DROP VIEW IF EXISTS v_spp_freq_state CASCADE;
CREATE VIEW v_spp_freq_state AS
SELECT
    s.*,
    a.ndotw,
    a.ndotu,
    a.ndotother,
    CASE
    WHEN s.npw + s.npu = 0 THEN
      NULL
    ELSE
      s.npw / (s.npw + s.npu)
    END AS wetland_freq_unadj,
    CASE
    WHEN s.npw + s.npu = 0 THEN
      NULL
    WHEN s.npw = 0 THEN
      0
    ELSE
      1 / (1 + (s.npu * a.ndotw / (s.npw * a.ndotu)))
    END AS wetland_freq_adj,
    CASE
```

```
    WHEN s.npw + s.npu = 0 THEN
      NULL
    ELSE
      0.98 * SQRT(1 / (s.npw + s.npu))
    END AS max_moe_at95cl,
    CASE
    WHEN s.npw + s.npu = 0 THEN
      NULL
    ELSE
      0.82 * SQRT(1 / (s.npw + s.npu))
    END AS max_moe_at90cl
FROM
    v_spp_by_state_flat AS s,
    v_area_by_state_flat AS a
WHERE
    s.state_fips = a.state_fips;
RAISE NOTICE 'Create view of frequency of species by state.';


DROP VIEW IF EXISTS v_spp_ind_status_region CASCADE;
CREATE VIEW v_spp_ind_status_region AS
SELECT
    s.*,
    r.region_abbr,
    r.region_name,
    CASE
    WHEN wetland_freq_unadj >= 0.99 THEN
      'OBL'
    WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
      'FACW'
    WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
      'FAC'
    WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
      'FACU'
    WHEN wetland_freq_unadj <= 0.01 THEN
      'UPL'
    ELSE
      '?'
    END AS wetland_ind_unadj,
    CASE
    WHEN wetland_freq_adj >= 0.99 THEN
      'OBL'
    WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
      'FACW'
```

```
    WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
      'FAC'
    WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
      'FACU'
    WHEN wetland_freq_adj <= 0.01 THEN
      'UPL'
    ELSE
      '?'
    END AS wetland_ind_adj,
    CASE
    WHEN wetland_freq_unadj >= 0.99 THEN
      5
    WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
      4
    WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
      3
    WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
      2
    WHEN wetland_freq_unadj <= 0.01 THEN
      1
    ELSE
      NULL
    END AS wetland_ind_unadj_rank,
    CASE
    WHEN wetland_freq_adj >= 0.99 THEN
      5
    WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
      4
    WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
      3
    WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
      2
    WHEN wetland_freq_adj <= 0.01 THEN
      1
    ELSE
    NULL
    END AS wetland_ind_adj_rank
FROM
    v_spp_freq_region s,
    public.regions as r
WHERE
    s.region_cd = r.region_cd
ORDER BY
    s.taxon,
```

```
    s.region_cd;
RAISE NOTICE 'Create view of indicator status of species by regions.';


DROP VIEW IF EXISTS v_spp_ind_status_arid_west_minus_ca CASCADE;
CREATE VIEW v_spp_ind_status_arid_west_minus_ca AS
SELECT
    s.*,
    r.region_abbr,
    r.region_name,
    CASE
    WHEN wetland_freq_unadj >= 0.99 THEN
        'OBL'
    WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
        'FACW'
    WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
        'FAC'
    WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
        'FACU'
    WHEN wetland_freq_unadj <= 0.01 THEN
        'UPL'
    ELSE
        '?'
    END AS wetland_ind_unadj,
    CASE
    WHEN wetland_freq_adj >= 0.99 THEN
        'OBL'
    WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
        'FACW'
    WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
        'FAC'
    WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
        'FACU'
    WHEN wetland_freq_adj <= 0.01 THEN
        'UPL'
    ELSE
        '?'
    END AS wetland_ind_adj,
    CASE
    WHEN wetland_freq_unadj >= 0.99 THEN
        5
    WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
        4
    WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
```

```
        3
    WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
        2
    WHEN wetland_freq_unadj <= 0.01 THEN
        1
    ELSE
        NULL
    END AS wetland_ind_unadj_rank,
    CASE
    WHEN wetland_freq_adj >= 0.99 THEN
        5
    WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
        4
    WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
        3
    WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
        2
    WHEN wetland_freq_adj <= 0.01 THEN
        1
    ELSE
    NULL
    END AS wetland_ind_adj_rank
FROM
    v_spp_freq_arid_west_minus_ca s,
    public.regions as r
WHERE
    s.region_cd = r.region_cd
ORDER BY
    s.taxon,
    s.region_cd;
RAISE NOTICE 'Create view of indicator status of species in Arid West minus CA.';


DROP VIEW IF EXISTS v_spp_ind_status_state CASCADE;
CREATE VIEW v_spp_ind_status_state AS
SELECT
    s.*,
    f.stusab,
    f.state_name,
    CASE
    WHEN wetland_freq_unadj >= 0.99 THEN
        'OBL'
    WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
        'FACW'
```

```
WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
   'FAC'
WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
   'FACU'
WHEN wetland_freq_unadj <= 0.01 THEN
   'UPL'
ELSE
   '?'
END AS wetland_ind_unadj,
CASE
WHEN wetland_freq_adj >= 0.99 THEN
   'OBL'
WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
   'FACW'
WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
   'FAC'
WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
   'FACU'
WHEN wetland_freq_adj <= 0.01 THEN
   'UPL'
ELSE
   '?'
END AS wetland_ind_adj,
CASE
WHEN wetland_freq_unadj >= 0.99 THEN
   5
WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
   4
WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
   3
WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
   2
WHEN wetland_freq_unadj <= 0.01 THEN
   1
ELSE
   NULL
END AS wetland_ind_unadj_rank,
CASE
WHEN wetland_freq_adj >= 0.99 THEN
   5
WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
   4
WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
   3
```

```
        WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
          2
        WHEN wetland_freq_adj <= 0.01 THEN
          1
        ELSE
        NULL
        END AS wetland_ind_adj_rank
    FROM
        v_spp_freq_state s,
        public.states_fips as f
    WHERE
        s.state_fips = f.state_fips
    ORDER BY
        s.taxon,
        s.state_fips;
    RAISE NOTICE 'Create view of indicator status of species by state.';


    ------------------------------------------------------------
    DROP VIEW IF EXISTS v_spp_ind_status_region_variance CASCADE;
    CREATE VIEW v_spp_ind_status_region_variance AS
    SELECT
        taxon,
        MAX(wetland_ind_unadj_rank) - MIN(wetland_ind_unadj_rank) AS
wetland_ind_unadj_rank_range,
        VAR_POP(wetland_ind_unadj_rank) AS wetland_ind_unadj_rank_variance,
        MAX(wetland_ind_adj_rank) - MIN(wetland_ind_adj_rank) AS
wetland_ind_adj_rank_range,
        VAR_POP(wetland_ind_adj_rank) AS wetland_ind_adj_rank_variance
    FROM
        v_spp_ind_status_region
    GROUP BY
        taxon;


    DROP VIEW IF EXISTS v_spp_ind_status_arid_west_minus_ca_variance CASCADE;
    CREATE VIEW v_spp_ind_status_arid_west_minus_ca_variance AS
    SELECT
        taxon,
        MAX(wetland_ind_unadj_rank) - MIN(wetland_ind_unadj_rank) AS
wetland_ind_unadj_rank_range,
        VAR_POP(wetland_ind_unadj_rank) AS wetland_ind_unadj_rank_variance,
        MAX(wetland_ind_adj_rank) - MIN(wetland_ind_adj_rank) AS
wetland_ind_adj_rank_range,
```

```
    VAR_POP(wetland_ind_adj_rank) AS wetland_ind_adj_rank_variance
  FROM
    v_spp_ind_status_arid_west_minus_ca
  GROUP BY
    taxon;


  DROP VIEW IF EXISTS v_spp_ind_status_state_variance CASCADE;
  CREATE VIEW v_spp_ind_status_state_variance AS
  SELECT
    taxon,
    MAX(wetland_ind_unadj_rank) - MIN(wetland_ind_unadj_rank) AS
wetland_ind_unadj_rank_range,
    VAR_POP(wetland_ind_unadj_rank) AS wetland_ind_unadj_rank_variance,
    MAX(wetland_ind_adj_rank) - MIN(wetland_ind_adj_rank) AS
wetland_ind_adj_rank_range,
    VAR_POP(wetland_ind_adj_rank) AS wetland_ind_adj_rank_variance
  FROM
    v_spp_ind_status_state
  GROUP BY
    taxon;


  DROP VIEW IF EXISTS v_spp_ind_status_region_flat CASCADE;
  CREATE VIEW v_spp_ind_status_region_flat AS
  SELECT DISTINCT ON (s1.taxon)
    s1.taxon,
    s1.genus,
    s1.species,
    s1.common_name,
    s_ncne.ndotw AS ncne_ndotw,
    s_ncne.ndotu AS ncne_ndotu,
    s_ncne.ndotother AS ncne_ndotother,
    s_ncne.npw AS ncne_npw,
    s_ncne.npu AS ncne_npu,
    s_ncne.npother AS ncne_npother,
    s_ncne.wetland_freq_unadj AS ncne_freq_unadj,
    s_ncne.wetland_freq_adj AS ncne_freq_adj,
    s_ncne.max_moe_at95cl AS ncne_max_moe_at95cl,
    s_ncne.max_moe_at90cl AS ncne_max_moe_at90cl,
    s_ncne.wetland_ind_unadj AS ncne_ind_unadj,
    s_ncne.wetland_ind_adj AS ncne_ind_adj,
    s_mw.ndotw AS mw_ndotw,
    s_mw.ndotu AS mw_ndotu,
```

s_mw.ndotother **AS** mw_ndotother,
s_mw.npw **AS** mw_npw,
s_mw.npu **AS** mw_npu,
s_mw.npother **AS** mw_npother,
s_mw.wetland_freq_unadj **AS** mw_freq_unadj,
s_mw.wetland_freq_adj **AS** mw_freq_adj,
s_mw.max_moe_at95cl **AS** mw_max_moe_at95cl,
s_mw.max_moe_at90cl **AS** mw_max_moe_at90cl,
s_mw.wetland_ind_unadj **AS** mw_ind_unadj,
s_mw.wetland_ind_adj **AS** mw_ind_adj,
s_emp.ndotw **AS** emp_ndotw,
s_emp.ndotu **AS** emp_ndotu,
s_emp.ndotother **AS** emp_ndotother,
s_emp.npw **AS** emp_npw,
s_emp.npu **AS** emp_npu,
s_emp.npother **AS** emp_npother,
s_emp.wetland_freq_unadj **AS** emp_freq_unadj,
s_emp.wetland_freq_adj **AS** emp_freq_adj,
s_emp.max_moe_at95cl **AS** emp_max_moe_at95cl,
s_emp.max_moe_at90cl **AS** emp_max_moe_at90cl,
s_emp.wetland_ind_unadj **AS** emp_ind_unadj,
s_emp.wetland_ind_adj **AS** emp_ind_adj,
s_gp.ndotw **AS** gp_ndotw,
s_gp.ndotu **AS** gp_ndotu,
s_gp.ndotother **AS** gp_ndotother,
s_gp.npw **AS** gp_npw,
s_gp.npu **AS** gp_npu,
s_gp.npother **AS** gp_npother,
s_gp.wetland_freq_unadj **AS** gp_freq_unadj,
s_gp.wetland_freq_adj **AS** gp_freq_adj,
s_gp.max_moe_at95cl **AS** gp_max_moe_at95cl,
s_gp.max_moe_at90cl **AS** gp_max_moe_at90cl,
s_gp.wetland_ind_unadj **AS** gp_ind_unadj,
s_gp.wetland_ind_adj **AS** gp_ind_adj,
s_aw.ndotw **AS** aw_ndotw,
s_aw.ndotu **AS** aw_ndotu,
s_aw.ndotother **AS** aw_ndotother,
s_aw.npw **AS** aw_npw,
s_aw.npu **AS** aw_npu,
s_aw.npother **AS** aw_npother,
s_aw.wetland_freq_unadj **AS** aw_freq_unadj,
s_aw.wetland_freq_adj **AS** aw_freq_adj,
s_aw.max_moe_at95cl **AS** aw_max_moe_at95cl,
s_aw.max_moe_at90cl **AS** aw_max_moe_at90cl,

s_aw.wetland_ind_unadj **AS** aw_ind_unadj,
s_aw.wetland_ind_adj **AS** aw_ind_adj,
s_agcp.ndotw **AS** agcp_ndotw,
s_agcp.ndotu **AS** agcp_ndotu,
s_agcp.ndotother **AS** agcp_ndotother,
s_agcp.npw **AS** agcp_npw,
s_agcp.npu **AS** agcp_npu,
s_agcp.npother **AS** agcp_npother,
s_agcp.wetland_freq_unadj **AS** agcp_freq_unadj,
s_agcp.wetland_freq_adj **AS** agcp_freq_adj,
s_agcp.max_moe_at95cl **AS** agcp_max_moe_at95cl,
s_agcp.max_moe_at90cl **AS** agcp_max_moe_at90cl,
s_agcp.wetland_ind_unadj **AS** agcp_ind_unadj,
s_agcp.wetland_ind_adj **AS** agcp_ind_adj,
s_wmvc.ndotw **AS** wmvc_ndotw,
s_wmvc.ndotu **AS** wmvc_ndotu,
s_wmvc.ndotother **AS** wmvc_ndotother,
s_wmvc.npw **AS** wmvc_npw,
s_wmvc.npu **AS** wmvc_npu,
s_wmvc.npother **AS** wmvc_npother,
s_wmvc.wetland_freq_unadj **AS** wmvc_freq_unadj,
s_wmvc.wetland_freq_adj **AS** wmvc_freq_adj,
s_wmvc.max_moe_at95cl **AS** wmvc_max_moe_at95cl,
s_wmvc.max_moe_at90cl **AS** wmvc_max_moe_at90cl,
s_wmvc.wetland_ind_unadj **AS** wmvc_ind_unadj,
s_wmvc.wetland_ind_adj **AS** wmvc_ind_adj,
s_ak.ndotw **AS** ak_ndotw,
s_ak.ndotu **AS** ak_ndotu,
s_ak.ndotother **AS** ak_ndotother,
s_ak.npw **AS** ak_npw,
s_ak.npu **AS** ak_npu,
s_ak.npother **AS** ak_npother,
s_ak.wetland_freq_unadj **AS** ak_freq_unadj,
s_ak.wetland_freq_adj **AS** ak_freq_adj,
s_ak.max_moe_at95cl **AS** ak_max_moe_at95cl,
s_ak.max_moe_at90cl **AS** ak_max_moe_at90cl,
s_ak.wetland_ind_unadj **AS** ak_ind_unadj,
s_ak.wetland_ind_adj **AS** ak_ind_adj,
s_hi.ndotw **AS** hi_ndotw,
s_hi.ndotu **AS** hi_ndotu,
s_hi.ndotother **AS** hi_ndotother,
s_hi.npw **AS** hi_npw,
s_hi.npu **AS** hi_npu,
s_hi.npother **AS** hi_npother,

```sql
    s_hi.wetland_freq_unadj AS hi_freq_unadj,
    s_hi.wetland_freq_adj AS hi_freq_adj,
    s_hi.max_moe_at95cl AS hi_max_moe_at95cl,
    s_hi.max_moe_at90cl AS hi_max_moe_at90cl,
    s_hi.wetland_ind_unadj AS hi_ind_unadj,
    s_hi.wetland_ind_adj AS hi_ind_adj,
    s_cb.ndotw AS cb_ndotw,
    s_cb.ndotu AS cb_ndotu,
    s_cb.ndotother AS cb_ndotother,
    s_cb.npw AS cb_npw,
    s_cb.npu AS cb_npu,
    s_cb.npother AS cb_npother,
    s_cb.wetland_freq_unadj AS cb_freq_unadj,
    s_cb.wetland_freq_adj AS cb_freq_adj,
    s_cb.max_moe_at95cl AS cb_max_moe_at95cl,
    s_cb.max_moe_at90cl AS cb_max_moe_at90cl,
    s_cb.wetland_ind_unadj AS cb_ind_unadj,
    s_cb.wetland_ind_adj AS cb_ind_adj,
    v.wetland_ind_unadj_rank_range,
    v.wetland_ind_unadj_rank_variance,
    v.wetland_ind_adj_rank_range,
    v.wetland_ind_adj_rank_variance
FROM
    v_spp_ind_status_region s1
    LEFT OUTER JOIN v_spp_ind_status_region s_ncne
        ON s1.taxon = s_ncne.taxon
        AND s_ncne.region_abbr = 'NCNE'
    LEFT OUTER JOIN v_spp_ind_status_region s_mw
        ON s1.taxon = s_mw.taxon
        AND s_mw.region_abbr = 'MW'
    LEFT OUTER JOIN v_spp_ind_status_region s_emp
        ON s1.taxon = s_emp.taxon
        AND s_emp.region_abbr = 'EMP'
    LEFT OUTER JOIN v_spp_ind_status_region s_gp
        ON s1.taxon = s_gp.taxon
        AND s_gp.region_abbr = 'GP'
    LEFT OUTER JOIN v_spp_ind_status_region s_aw
        ON s1.taxon = s_aw.taxon
        AND s_aw.region_abbr = 'AW'
    LEFT OUTER JOIN v_spp_ind_status_region s_agcp
        ON s1.taxon = s_agcp.taxon
        AND s_agcp.region_abbr = 'AGCP'
    LEFT OUTER JOIN v_spp_ind_status_region s_wmvc
        ON s1.taxon = s_wmvc.taxon
```

```
    AND s_wmvc.region_abbr = 'WMVC'
  LEFT OUTER JOIN v_spp_ind_status_region s_ak
    ON s1.taxon = s_ak.taxon
    AND s_ak.region_abbr = 'AK'
  LEFT OUTER JOIN v_spp_ind_status_region s_hi
    ON s1.taxon = s_hi.taxon
    AND s_hi.region_abbr = 'HI'
  LEFT OUTER JOIN v_spp_ind_status_region s_cb
    ON s1.taxon = s_cb.taxon
    AND s_cb.region_abbr = 'CB'
  LEFT OUTER JOIN v_spp_ind_status_region_variance v
    ON s1.taxon = v.taxon;
---------------------------------------------------------


DROP VIEW IF EXISTS v_spp_compare_nwpl_flat CASCADE;
CREATE VIEW v_spp_compare_nwpl_flat AS
SELECT
  s.taxon AS nps_taxon,
  n.species AS nwpl_taxon,
  s.common_name AS nps_common_name,
  n.common_name AS nwpl_common_name,
  n.ncne AS nwpl_ind_ncne,
  s.ncne_ndotw AS nps_ncne_ndotw,
  s.ncne_ndotu AS nps_ncne_ndotu,
  s.ncne_ndotother AS nps_ncne_ndotother,
  s.ncne_npw AS nps_ncne_npw,
  s.ncne_npu AS nps_ncne_npu,
  s.ncne_npother AS nps_ncne_npother,
  s.ncne_freq_adj AS nps_ncne_freq_adj,
  s.ncne_freq_unadj AS nps_ncne_freq_unadj,
  s.ncne_max_moe_at95cl AS nps_ncne_max_moe_at95cl,
  s.ncne_max_moe_at90cl AS nps_ncne_max_moe_at90cl,
  s.ncne_ind_adj AS nps_ncne_ind_adj,
  s.ncne_ind_unadj AS nps_ncne_ind_unadj,
  n.mw AS nwpl_ind_mw,
  s.mw_ndotw AS nps_mw_ndotw,
  s.mw_ndotu AS nps_mw_ndotu,
  s.mw_ndotother AS nps_mw_ndotother,
  s.mw_npw AS nps_mw_npw,
  s.mw_npu AS nps_mw_npu,
  s.mw_npother AS nps_mw_npother,
  s.mw_freq_adj AS nps_mw_freq_adj,
  s.mw_freq_unadj AS nps_mw_freq_unadj,
```

s.mw_max_moe_at95cl **AS** nps_mw_max_moe_at95cl,
s.mw_max_moe_at90cl **AS** nps_mw_max_moe_at90cl,
s.mw_ind_adj **AS** nps_mw_ind_adj,
s.mw_ind_unadj **AS** nps_mw_ind_unadj,
n.emp **AS** nwpl_ind_emp,
s.emp_ndotw **AS** nps_emp_ndotw,
s.emp_ndotu **AS** nps_emp_ndotu,
s.emp_ndotother **AS** nps_emp_ndotother,
s.emp_npw **AS** nps_emp_npw,
s.emp_npu **AS** nps_emp_npu,
s.emp_npother **AS** nps_emp_npother,
s.emp_freq_adj **AS** nps_freq_emp_adj,
s.emp_freq_unadj **AS** nps_emp_freq_unadj,
s.emp_max_moe_at95cl **AS** nps_emp_max_moe_at95cl,
s.emp_max_moe_at90cl **AS** nps_emp_max_moe_at90cl,
s.emp_ind_adj **AS** nps_ind_emp_adj,
s.emp_ind_unadj **AS** nps_emp_ind_unadj,
n.gp **AS** nwpl_ind_gp,
s.gp_ndotw **AS** nps_gp_ndotw,
s.gp_ndotu **AS** nps_gp_ndotu,
s.gp_ndotother **AS** nps_gp_ndotother,
s.gp_npw **AS** nps_gp_npw,
s.gp_npu **AS** nps_gp_npu,
s.gp_npother **AS** nps_gp_npother,
s.gp_freq_adj **AS** nps_freq_gp_adj,
s.gp_freq_unadj **AS** nps_gp_freq_unadj,
s.gp_max_moe_at95cl **AS** nps_gp_max_moe_at95cl,
s.gp_max_moe_at90cl **AS** nps_gp_max_moe_at90cl,
s.gp_ind_adj **AS** nps_ind_gp_adj,
s.gp_ind_unadj **AS** nps_gp_ind_unadj,
n.aw **AS** nwpl_ind_aw,
s.aw_ndotw **AS** nps_aw_ndotw,
s.aw_ndotu **AS** nps_aw_ndotu,
s.aw_ndotother **AS** nps_aw_ndotother,
s.aw_npw **AS** nps_aw_npw,
s.aw_npu **AS** nps_aw_npu,
s.aw_npother **AS** nps_aw_npother,
s.aw_freq_adj **AS** nps_freq_aw_adj,
s.aw_freq_unadj **AS** nps_aw_freq_unadj,
s.aw_max_moe_at95cl **AS** nps_aw_max_moe_at95cl,
s.aw_max_moe_at90cl **AS** nps_aw_max_moe_at90cl,
s.aw_ind_adj **AS** nps_ind_aw_adj,
s.aw_ind_unadj **AS** nps_aw_ind_unadj,
n.agcp **AS** nwpl_ind_agcp,

s.agcp_ndotw **AS** nps_agcp_ndotw,
s.agcp_ndotu **AS** nps_agcp_ndotu,
s.agcp_ndotother **AS** nps_agcp_ndotother,
s.agcp_npw **AS** nps_agcp_npw,
s.agcp_npu **AS** nps_agcp_npu,
s.agcp_npother **AS** nps_agcp_npother,
s.agcp_freq_adj **AS** nps_agcp_freq_adj,
s.agcp_freq_unadj **AS** nps_agcp_freq_unadj,
s.agcp_max_moe_at95cl **AS** nps_agcp_max_moe_at95cl,
s.agcp_max_moe_at90cl **AS** nps_agcp_max_moe_at90cl,
s.agcp_ind_adj **AS** nps_agcp_ind_adj,
s.agcp_ind_unadj **AS** nps_agcp_ind_unadj,
n.wmvc **AS** nwpl_ind_wmvc,
s.wmvc_ndotw **AS** nps_wmvc_ndotw,
s.wmvc_ndotu **AS** nps_wmvc_ndotu,
s.wmvc_ndotother **AS** nps_wmvc_ndotother,
s.wmvc_npw **AS** nps_wmvc_npw,
s.wmvc_npu **AS** nps_wmvc_npu,
s.wmvc_npother **AS** nps_wmvc_npother,
s.wmvc_freq_adj **AS** nps_wmvc_freq_adj,
s.wmvc_freq_unadj **AS** nps_wmvc_freq_unadj,
s.wmvc_max_moe_at95cl **AS** nps_wmvc_max_moe_at95cl,
s.wmvc_max_moe_at90cl **AS** nps_wmvc_max_moe_at90cl,
s.wmvc_ind_adj **AS** nps_wmvc_ind_adj,
s.wmvc_ind_unadj **AS** nps_wmvc_ind_unadj,
n.ak **AS** nwpl_ind_ak,
s.ak_ndotw **AS** nps_ak_ndotw,
s.ak_ndotu **AS** nps_ak_ndotu,
s.ak_ndotother **AS** nps_ak_ndotother,
s.ak_npw **AS** nps_ak_npw,
s.ak_npu **AS** nps_ak_npu,
s.ak_npother **AS** nps_ak_npother,
s.ak_freq_adj **AS** nps_ak_freq_adj,
s.ak_freq_unadj **AS** nps_ak_freq_unadj,
s.ak_max_moe_at95cl **AS** nps_ak_max_moe_at95cl,
s.ak_max_moe_at90cl **AS** nps_ak_max_moe_at90cl,
s.ak_ind_adj **AS** nps_ak_ind_adj,
s.ak_ind_unadj **AS** nps_ak_ind_unadj,
n.hi **AS** nwpl_ind_hi,
s.hi_ndotw **AS** nps_hi_ndotw,
s.hi_ndotu **AS** nps_hi_ndotu,
s.hi_ndotother **AS** nps_hi_ndotother,
s.hi_npw **AS** nps_hi_npw,
s.hi_npu **AS** nps_hi_npu,

```
   s.hi_npother AS nps_hi_npother,
   s.hi_freq_adj AS nps_hi_freq_adj,
   s.hi_freq_unadj AS nps_hi_freq_unadj,
   s.hi_max_moe_at95cl AS nps_hi_max_moe_at95cl,
   s.hi_max_moe_at90cl AS nps_hi_max_moe_at90cl,
   s.hi_ind_adj AS nps_hi_ind_adj,
   s.hi_ind_unadj AS nps_hi_ind_unadj,
   n.cb AS nwpl_ind_cb,
   s.cb_ndotw AS nps_cb_ndotw,
   s.cb_ndotu AS nps_cb_ndotu,
   s.cb_ndotother AS nps_cb_ndotother,
   s.cb_npw AS nps_cb_npw,
   s.cb_npu AS nps_cb_npu,
   s.cb_npother AS nps_cb_npother,
   s.cb_freq_adj AS nps_cb_freq_adj,
   s.cb_freq_unadj AS nps_cb_freq_unadj,
   s.cb_max_moe_at95cl AS nps_cb_max_moe_at95cl,
   s.cb_max_moe_at90cl AS nps_cb_max_moe_at90cl,
   s.cb_ind_adj AS nps_cb_ind_adj,
   s.cb_ind_unadj AS nps_cb_ind_unadj,
   s.wetland_ind_unadj_rank_range AS nps_wetland_ind_unadj_rank_range,
   s.wetland_ind_unadj_rank_variance AS nps_wetland_ind_unadj_rank_variance,
   s.wetland_ind_adj_rank_range AS nps_wetland_ind_adj_rank_range,
   s.wetland_ind_adj_rank_variance AS nps_wetland_ind_adj_rank_variance
FROM
   v_spp_ind_status_region_flat s INNER JOIN public.nwpl_2013 n
      ON s.taxon = n.species
ORDER BY
   s.taxon;


DROP VIEW IF EXISTS v_spp_compare_arid_west_ca CASCADE;
CREATE VIEW v_spp_compare_arid_west_ca AS
SELECT
   COALESCE(aw.taxon, ca.taxon) AS taxon,
   COALESCE(aw.common_name, ca.common_name) AS common_name,
   aw.npw AS aw_minus_ca_npw,
   aw.npu AS aw_minus_ca_npu,
   aw.npother AS aw_minus_ca_npother,
   aw.ndotw AS aw_minus_ca_ndotw,
   aw.ndotu AS aw_minus_ca_ndotu,
   aw.ndotother AS aw_minus_ca_ndotother,
   aw.wetland_freq_unadj AS aw_minus_ca_freq_unadj,
   aw.wetland_freq_adj AS aw_minus_minus_ca_freq_adj,
```

```
    aw.wetland_ind_unadj AS aw_minus_ca_ind_unadj,
    aw.wetland_ind_adj AS aw_minus_minus_ca_ind_adj,
    aw.max_moe_at95cl AS aw_max_moe_at95cl,
    aw.max_moe_at90cl AS aw_max_moe_at90cl,
    ca.npw AS ca_npw,
    ca.npu AS ca_npu,
    ca.npother AS ca_npother,
    ca.ndotw AS ca_ndotw,
    ca.ndotu AS ca_ndotu,
    ca.ndotother AS ca_ndotother,
    ca.wetland_freq_unadj AS ca_freq_unadj,
    ca.wetland_freq_adj AS ca_freq_adj,
    ca.wetland_ind_unadj AS ca_ind_unadj,
    ca.wetland_ind_adj AS ca_ind_adj,
    ca.max_moe_at95cl AS ca_max_moe_at95cl,
    ca.max_moe_at90cl AS ca_max_moe_at90cl
  FROM
    v_spp_ind_status_arid_west_minus_ca aw LEFT OUTER JOIN v_spp_ind_status_state
ca
      ON (aw.taxon = ca.taxon AND ca.state_fips = '06')
      OR (aw.taxon IS NULL AND ca.state_fips = '06')
      OR (ca.taxon IS NULL AND aw.taxon IS NOT NULL)
  ORDER BY
    aw.genus,
    ca.genus,
    aw.species,
    ca.species;


END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Appendix 15 – SQL program to display NPS analysis

Filename: nps_analysis.sql

SET search_path **TO** nps;

select * from v_spp_ind_status_region_flat **ORDER BY** taxon;
select * from v_spp_compare_nwpl_flat **ORDER BY** nwpl_taxon;
select * from v_spp_compare_arid_west_ca **ORDER BY** taxon;

Appendix 16 – Python program to process FIAD data

Filename: process_fiad.py3

```
#!/usr/bin/python3
# coding: utf-8

################################################################################
#
# AUTHOR(S):    Matthew F. Buff
# PURPOSE:      process USFS FIA data
# COPYRIGHT:    Copyright 2013 Matthew F. Buff
#
################################################################################

import os, os.path, sys, http.client, zipfile, subprocess, psycopg2, time, \
calendar, shlex
from math import log
from psycopg2 import errorcodes


schema_file = 'fiad_schema_v5_1_7.sql'
geography_file = 'fiad_geography.sql'
summaries_file = 'fiad_summaries.sql'

code_path = os.path.abspath(os.path.dirname(sys.argv[0]))

# no file for HI
geo_areas = ('AK','AL','AR','AZ','CA','CO','CT','DE','FL','GA','IA','ID','IL',
    'IN','KS','KY','LA','MA','MD','ME','MI','MN','MO','MS','MT','NC','ND',
    'NE','NH','NJ','NM','NV','NY','OH','OK','OR','PA','PR','RI','SC','SD',
    'TN','TX','UT','VA','VI','VT','WA','WI','WV','WY')
fiad_host = 'apps.fs.fed.us'
fiad_url = '/fiadb-downloads/'
zip_dir = os.path.abspath(os.curdir)
# extract_dir = 'extract'
extract_dir = zip_dir

state_tables = ('boundary','cond','cond_dwm_calc','county',
    'dwm_coarse_woody_debris','dwm_duff_litter_fuel','dwm_fine_woody_debris',
    'dwm_microplot_fuel','dwm_residual_pile','dwm_transect_segment','dwm_visit',
    'lichen_lab','lichen_plot_summary','lichen_visit','ozone_biosite_summary',
    'ozone_plot','ozone_plot_summary','ozone_species_summary',
    'ozone_validation','ozone_visit','p2veg_subplot_spp','p2veg_subp_structure',
```

```
    'plot','plotgeom','plotsnap','pop_estn_unit','pop_eval',
    'pop_eval_attribute','pop_eval_grp','pop_eval_typ','pop_plot_stratum_assgn',
    'pop_stratum','seedling','sitetree','soils_erosion','soils_lab',
    'soils_sample_loc','soils_visit','subp_cond','subp_cond_chng_mtrx',
    'subplot','survey','tree','tree_grm_estn','tree_regional_biomass',
    'veg_plot_species','veg_quadrat','veg_subplot','veg_subplot_spp',
    'veg_visit')

ref_tables = ('lichen_species_summary','ref_citation','ref_fiadb_version',
    'ref_forest_type','ref_forest_type_group','ref_habtyp_description',
    'ref_habtyp_publication','ref_invasive_species','ref_lichen_species',
    'ref_lichen_spp_comments','ref_plant_dictionary','ref_pop_attribute',
    'ref_pop_eval_typ_descr','ref_research_station','ref_species',
    'ref_species_group','ref_state_elev','ref_unit')

IEC_units = ('B', 'KiB', 'MiB', 'GiB')

def get_IEC_units(bytes):
    exponent = int(log(bytes, 1024))
    return '{:.1f} {}'.format(float(bytes) / pow(1024, exponent),
        IEC_units[exponent])

def get_fia_files():
    geo_area = ''

    # example url: http://apps.fs.fed.us/fiadb-downloads/AK.ZIP
    print('Checking files at %s' % (fiad_host))

    for geo_area in geo_areas + ('FIADB_REFERENCE',):
        file_name = geo_area + '.ZIP'
        local_path = os.path.join(zip_dir, file_name)
        remote_url = fiad_url + file_name

        if os.path.isfile(local_path):
            local_exists = True
            local_mtime = os.path.getmtime(local_path)
            local_mtime_struct = time.gmtime(local_mtime)
            local_mtime_fmt = time.strftime("%a, %d %b %Y %H:%M:%S %Z",
                                            time.gmtime(local_mtime))
            local_size = os.path.getsize(local_path)
        else:
            local_exists = False

        conn = http.client.HTTPConnection(fiad_host)
```

```
conn.request('HEAD', remote_url)
resp = conn.getresponse()
# need to read response in order process future requests:
#data = resp.read()
conn.close()
remote_mtime_struct = time.strptime(resp.getheader('Last-Modified'),
                                    "%a, %d %b %Y %H:%M:%S %Z")
remote_mtime_fmt = time.strftime("%a, %d %b %Y %H:%M:%S %Z",
                                 remote_mtime_struct)
remote_size = int(resp.getheader('Content-Length'))
remote_size_fmt = get_IEC_units(remote_size)

download_file = False
msg = file_name + ': '
if local_exists:
    if remote_mtime_struct != local_mtime_struct:
        download_file = True
        msg = msg + 'Remote and local file dates are different: ' + \
            remote_mtime_fmt + ' vs. ' + local_mtime_fmt + '.'
    elif local_size != remote_size:
        download_file = True
        msg = msg + 'Remote and local file sizes are different: ' + \
            remote_size + ' bytes vs. ' + local_size + ' bytes.'
    elif (remote_mtime_struct == local_mtime_struct and
          local_size == remote_size):
        msg = msg + 'Local copy is up to date.'
else:
    download_file = True
    msg = msg + 'Local copy is missing.'

print(msg)

if download_file:
    print('Downloading ' + file_name + '(' + remote_size_fmt + ').')
    conn.request('GET', remote_url)
    resp = conn.getresponse()
    data = resp.read()
    with open(local_path, 'wb') as f:
        f.write(data)
        os.utime(local_path, (round(time.time()),
            round(calendar.timegm(remote_mtime_struct))))

conn.close()
```

```
def extract_fia_files():
    print('Extracting archives')
    for geo_area in geo_areas + ('FIADB_REFERENCE',):
        in_file_name = geo_area + '.ZIP'
        in_file_path = os.path.join(zip_dir, in_file_name)
        zipfile.ZipFile(in_file_path).extractall(path=extract_dir)
        print('Finished extracting ' + geo_area + '.ZIP')


def truncate(conn, cur, tables):
    print('Truncating and resetting sequence columns for tables: ' + ', '.join(tables) + '.')
    sql = 'TRUNCATE ' + ', '.join(tables) + ' RESTART IDENTITY;'
    cur.execute(sql)
    conn.commit()


def vacuum(conn, cur, tables):
    print('Vacuuming tables: ' + ', '.join(tables) + '.')
    iso = conn.isolation_level
    conn.set_isolation_level(0)
    for table in tables:
        cur.execute('VACUUM FULL ANALYZE ' + table + ';')
    conn.set_isolation_level(iso)
    conn.commit()


def copy_state_tables():
    # psycopg2 copy_from cannot ignore quoted text in CSV files
    # psycopg2 copy_expert, i.e. COPY requires postgres user

    # PostgreSQL COPY command requires double quotes around db objects to
    # preserve case and single quotes around file path
    for geo_area in geo_areas:
        print(geo_area)
        for table in state_tables:
            file_path = os.path.join(extract_dir, geo_area + '_' + table.upper() + '.CSV')
            sql = '\copy \"fiad\".\"' + table + '" FROM \'' + file_path + '\' WITH CSV HEADER
ENCODING \'WIN1252\''
            #sql = '\copy \"fiad\".\"' + table + '" FROM \'' + file_path + '\' CSV HEADER'
            try:
                print('\t' + table)
                #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
                #check_call will halt python program if subprocess has non-zero return
                subprocess.check_call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
```

```
        except:
            print(sql)
            print('Unknown error', geo_area)
            raise


def copy_ref_tables():
    # psycopg2 copy_from cannot ignore quoted text in CSV files
    # psycopg2 copy_expert, i.e. COPY requires postgres user

    # PostgreSQL COPY command requires double quotes around db objects to
    # preseve case and single quotes around file path
    for table in ref_tables:
        file_path = os.path.join(extract_dir, table.upper() + '.CSV')
        sql = '\copy \"fiad\".\"' + table + '" FROM \'' + file_path + '\' CSV HEADER'
        try:
            print('\t' + table)
            #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
            #check_call will halt python program if subprocess has non-zero return
            subprocess.check_call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        except:
            print(sql)
            print('Unknown error', geo_area)
            raise


def run_sql(conn, cur, sql):
    print(sql)
    cur.execute(sql)
    for notice in conn.notices:
        print(notice)
    conn.notices.clear()
    conn.commit()


def run_psql(sql_path):
    # psycopg2 can't handle multi-statement sql
    try:
        #subprocess.call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        #check_call will halt python program if subprocess has non-zero return
        #subprocess.check_call(['psql', '-d', 'nwpl', '-c', sql, '--set=ON_ERROR_STOP=true'])
        cmd = 'psql -d nwpl -f ' + sql_path + ' --set=ON_ERROR_STOP=true'
        msg = subprocess.check_output(shlex.split(cmd), stderr=subprocess.STDOUT,
universal_newlines=True)
```

```
        print(msg)
    except subprocess.CalledProcessError as e:
        print('command returned error code:', e.returncode)
        print('command was:', e.cmd)
        print('output was:', e.output)
        raise
    except:
        print(cmd)
        print('Unknown error')
        raise


def main():
    # allow user to cancel the program
    user_continue = input('This program will delete data in the fiad database. Continue (Y/n)?')
    if user_continue.lower() in ('n', 'no'):
        print('Program canceled.')
        return 0
    else:
        print('Continuing.')

    # comment out if files already exist
    # allow user to skip
    user_continue = input('Download data from USFS website (y/N)?')
    if user_continue.lower() in ('y', 'yes'):
        get_fia_files()
        extract_fia_files()
    else:
        print('Skipping download.')

    # remember to manually start the db cluster
    conn = psycopg2.connect(database = 'nwpl')
    cur = conn.cursor()

    # uncomment to offer truncation as a separate step
    #print('State tables include: ', ', '.join(state_tables))
    #print('Reference tables include: ', ', '.join(ref_tables))
    #user_continue = input('Truncate all state and reference tables (y/N)?')
    #if user_continue.lower() in ('y', 'yes'):
        ## truncate tables and reset sequence columns before loading data
        ## need double quotes around db objects to preseve case
        #truncate(conn, cur, ['"fiad"."' + table + '"' for table in state_tables])
        #vacuum(conn, cur, ['"fiad"."' + table + '"' for table in state_tables])
        #truncate(conn, cur, ['"fiad"."' + table + '"' for table in ref_tables])
```

```
    #vacuum(conn, cur, ['"fiad"."' + table + '"' for table in ref_tables])
#else:
    #print('Skipping truncation of all tables.')

user_continue = input('Create local FIAD database structure (Y/n)?')
if user_continue.lower() in ('n', 'no'):
    print('Skipping the creation of local FIAD database structure.')
else:
    print('Creating local FIAD database structure.')
    run_sql(conn, cur, 'CREATE SCHEMA IF NOT EXISTS fiad;')
    #with open(os.path.join(code_path, schema_file)) as f:
        #run_psql(f.read())
    run_psql(os.path.join(code_path, schema_file))
    # create functions from local sql files
    run_psql(os.path.join(code_path, geography_file))
    run_psql(os.path.join(code_path, summaries_file))

print('State tables include: ', ', '.join(state_tables))
user_continue = input('Copy state table files to database (y/N)?')
if user_continue.lower() in ('y', 'yes'):
    # truncate tables and reset sequence columns before loading data
    # need double quotes around db objects to preseve case
    truncate(conn, cur, ['"fiad"."' + table + '"' for table in state_tables])
    vacuum(conn, cur, ['"fiad"."' + table + '"' for table in state_tables])
    copy_state_tables()
    vacuum(conn, cur, ['"fiad"."' + table + '"' for table in state_tables])
else:
    print('Skipping copying of state tables.')

print('Reference tables include: ', ', '.join(ref_tables))
user_continue = input('Copy reference table files to database (y/N)?')
if user_continue.lower() in ('y', 'yes'):
    # truncate tables and reset sequence columns before loading data
    # need double quotes around db objects to preseve case
    truncate(conn, cur, ['"fiad"."' + table + '"' for table in ref_tables])
    vacuum(conn, cur, ['"fiad"."' + table + '"' for table in ref_tables])
    copy_ref_tables()
    vacuum(conn, cur, ['"fiad"."' + table + '"' for table in ref_tables])
else:
    print('Skipping copying of reference tables.')

user_continue = input('Run sql functions (Y/n)?')
if user_continue.lower() in ('n', 'no'):
    print('Skipping sql functions.')
```

```
    else:
        print('Running fiad.geography.')
        run_sql(conn, cur, 'SELECT fiad.fiad_geography();')
        print('Running fiad.summaries.')
        run_sql(conn, cur, 'SELECT fiad.fiad_summaries();')

    cur.close()
    conn.close()

    return 0

if __name__ == "__main__":
    main()
```

Appendix 17 – SQL program to create FIAD table schema

Filename: fiad_schema_v5_1_7.sql

```
-- MFB changes:

-- add missing contraints and convert unique indexes to constraints
-- unique contraints create implicit unique indexes
-- some but not all primary keys were already present and some other indexes
-- were present

-- table subp_cond_chng_mtrx
-- changed the postgres unknown type fields to integer in table

-- table trees, added:
--      disease_srs                 integer,
--      dieback_severity_srs                integer

-- table ref_species
-- changed following fields from integer to numeric
--      dwm_carbon_ratio
--      standing_dead_decay_ratio1
--      standing_dead_decay_ratio2
--      standing_dead_decay_ratio3
--      standing_dead_decay_ratio4

-- table subplot
-- changed following fields from integer to numeric
--      waterdep,

-- tables dwm_duff_litter_fuel, p2veg_subp_structure
-- updated to FIA v 5.1.5

-- tables evalidator_changes
-- removed because it is only present in template .accdb

SET search_path TO fiad;

-- ----------------------------------------------------------
-- mdb tools - a library for reading ms access database files
-- copyright (c) 2000-2011 brian bruns and others.
-- files in libmdb are licensed under lgpl and the utilities under
-- the gpl, see copying.lib and copying files respectively.
-- check out http://mdbtools.sourceforge.net
```

```
-- ----------------------------------------------------------

DROP TABLE IF EXISTS boundary CASCADE;
CREATE TABLE boundary
 (
	cn			varchar (68),
	plt_cn			varchar (68),
	invyr			integer,
	statecd			integer,
	unitcd			integer,
	countycd		integer,
	plot			integer,
	subp			integer,
	subptyp			integer,
	bndchg			integer,
	contrast		integer,
	azmleft			integer,
	azmcorn			integer,
	distcorn		integer,
	azmright		integer,
	cycle			integer,
	subcycle		integer,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance	varchar (100),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance	varchar (100)
);

-- create indexes ...
ALTER TABLE boundary ADD CONSTRAINT boundary_pkey PRIMARY KEY (cn);
--ALTER TABLE boundary ADD CONSTRAINT boundary_ukey UNIQUE (plt_cn, subp,
subptyp, azmleft, azmright);
--CREATE INDEX boundary_plt_cn_idx ON cond (plt_cn);

DROP TABLE IF EXISTS cond CASCADE;
CREATE TABLE cond
 (
	cn			varchar (68),
	plt_cn			varchar (68),
	invyr			integer,
	statecd			integer,
	unitcd			integer,
```

```
countycd                    integer,
plot                integer,
condid                      integer,
cond_status_cd                    integer,
cond_nonsample_reasn_cd                    integer,
reservcd                     integer,
owncd                 integer,
owngrpcd                      integer,
forindcd                     integer,
adforcd                      integer,
fortypcd                     integer,
fldtypcd                     integer,
mapden                       integer,
stdage                      integer,
stdszcd                      integer,
fldszcd                      integer,
siteclcd                    integer,
sicond                      integer,
sibase                      integer,
sisp                integer,
stdorgcd                     integer,
stdorgsp                     integer,
prop_basis                    varchar (24),
condprop_unadj                       numeric(5,4),
micrprop_unadj                       numeric(5,4),
subpprop_unadj                       numeric(5,4),
macrprop_unadj                       numeric(5,4),
slope                integer,
aspect                      integer,
physclcd                      integer,
gsstkcd                      integer,
alstkcd                     integer,
dstrbcd1                     integer,
dstrbyr1                     integer,
dstrbcd2                     integer,
dstrbyr2                     integer,
dstrbcd3                     integer,
dstrbyr3                     integer,
trtcd1                     integer,
trtyr1                     integer,
trtcd2                     integer,
trtyr2                     integer,
trtcd3                     integer,
trtyr3                     integer,
```

```
presnfcd integer,
balive numeric(9,4),
fldage integer,
alstk numeric(7,4),
gsstk numeric(7,4),
fortypcdcalc integer,
habtypcd1 varchar (20),
habtypcd1_pub_cd varchar (20),
habtypcd1_descr_pub_cd varchar (20),
habtypcd2 varchar (20),
habtypcd2_pub_cd varchar (20),
habtypcd2_descr_pub_cd varchar (20),
mixedconfcd varchar (2),
vol_loc_grp varchar (400),
siteclcdest integer,
sitetree_tree integer,
sitecl_method integer,
carbon_down_dead numeric(13,6),
carbon_litter numeric(13,6),
carbon_soil_org numeric(13,6),
carbon_standing_dead numeric(13,6),
carbon_understory_ag numeric(13,6),
carbon_understory_bg numeric(13,6),
created_by varchar (60),
created_date timestamp without time zone,
created_in_instance varchar (12),
modified_by varchar (60),
modified_date timestamp without time zone,
modified_in_instance varchar (12),
cycle integer,
subcycle integer,
soil_rooting_depth_pnw varchar (2),
ground_land_class_pnw varchar (6),
plant_stockability_factor_pnw double precision,
stnd_cond_cd_pnwrs integer,
stnd_struc_cd_pnwrs integer,
stump_cd_pnwrs varchar (2),
fire_srs integer,
grazing_srs integer,
harvest_type1_srs integer,
harvest_type2_srs integer,
harvest_type3_srs integer,
land_use_srs integer,
operability_srs integer,
```

```
        stand_structure_srs                 integer,
        nf_cond_status_cd                   integer,
        nf_cond_nonsample_reasn_cd                  integer,
        canopy_cvr_sample_method_cd                 integer,
        live_canopy_cvr_pct                 integer,
        live_missing_canopy_cvr_pct                 integer,
        nbr_live_stems                      integer,
        ownsubcd                    integer,
        industrialcd_fiadb                  integer,
        reservcd_fld                integer,
        admin_withdrawn_cd                  integer
);

-- create indexes ...
ALTER TABLE cond ADD CONSTRAINT cond_pkey PRIMARY KEY (cn);
-- keep cond_ukey:
ALTER TABLE cond ADD CONSTRAINT cond_ukey UNIQUE (plt_cn, condid);
CREATE INDEX cond_plt_cn_idx ON cond (plt_cn);

DROP TABLE IF EXISTS cond_dwm_calc;
CREATE TABLE cond_dwm_calc
 (
        cn                  varchar (68),
        statecd                     integer,
        countycd                    integer,
        plot                integer,
        measyear                    integer,
        invyr               integer,
        condid                      integer,
        evalid                      integer,
        plt_cn                      varchar (68),
        cnd_cn                      varchar (68),
        stratum_cn                  varchar (68),
        phase               varchar (6),
        condprop_cwd                numeric(13,12),
        condprop_fwd_sm                     numeric(13,12),
        condprop_fwd_md                     numeric(13,12),
        condprop_fwd_lg                     numeric(13,12),
        condprop_duff               numeric(13,12),
        cwd_tl_cond                 numeric(13,10),
        cwd_tl_unadj                numeric(13,10),
        cwd_tl_adj                  numeric(13,10),
        cwd_lpa_cond                double precision,
        cwd_lpa_unadj               double precision,
```

```
cwd_lpa_adj double precision,
cwd_volcf_cond double precision,
cwd_volcf_unadj double precision,
cwd_volcf_adj double precision,
cwd_drybio_cond double precision,
cwd_drybio_unadj double precision,
cwd_drybio_adj double precision,
cwd_carbon_cond double precision,
cwd_carbon_unadj double precision,
cwd_carbon_adj double precision,
fwd_sm_tl_cond numeric(13,10),
fwd_sm_tl_unadj numeric(13,10),
fwd_sm_tl_adj numeric(13,10),
fwd_sm_cnt_cond double precision,
fwd_sm_volcf_cond double precision,
fwd_sm_volcf_unadj double precision,
fwd_sm_volcf_adj double precision,
fwd_sm_drybio_cond double precision,
fwd_sm_drybio_unadj double precision,
fwd_sm_drybio_adj double precision,
fwd_sm_carbon_cond double precision,
fwd_sm_carbon_unadj double precision,
fwd_sm_carbon_adj double precision,
fwd_md_tl_cond numeric(13,10),
fwd_md_tl_unadj numeric(13,10),
fwd_md_tl_adj numeric(13,10),
fwd_md_cnt_cond double precision,
fwd_md_volcf_cond double precision,
fwd_md_volcf_unadj double precision,
fwd_md_volcf_adj double precision,
fwd_md_drybio_cond double precision,
fwd_md_drybio_unadj double precision,
fwd_md_drybio_adj double precision,
fwd_md_carbon_cond double precision,
fwd_md_carbon_unadj double precision,
fwd_md_carbon_adj double precision,
fwd_lg_tl_cond numeric(13,10),
fwd_lg_tl_unadj numeric(13,10),
fwd_lg_tl_adj numeric(13,10),
fwd_lg_cnt_cond double precision,
fwd_lg_volcf_cond double precision,
fwd_lg_volcf_unadj double precision,
fwd_lg_volcf_adj double precision,
fwd_lg_drybio_cond double precision,
```

```
    fwd_lg_drybio_unadj              double precision,
    fwd_lg_drybio_adj              double precision,
    fwd_lg_carbon_cond              double precision,
    fwd_lg_carbon_unadj              double precision,
    fwd_lg_carbon_adj              double precision,
    pile_sample_area_cond              numeric(13,12),
    pile_sample_area_unadj                   numeric(13,12),
    pile_sample_area_adj              numeric(13,12),
    pile_volcf_cond              double precision,
    pile_volcf_unadj              double precision,
    pile_volcf_adj              double precision,
    pile_drybio_cond              double precision,
    pile_drybio_unadj              double precision,
    pile_drybio_adj              double precision,
    pile_carbon_cond              double precision,
    pile_carbon_unadj              double precision,
    pile_carbon_adj              double precision,
    fuel_depth              double precision,
    fuel_biomass              double precision,
    fuel_carbon              double precision,
    duff_depth              double precision,
    duff_biomass              double precision,
    duff_carbon              double precision,
    litter_depth              double precision,
    litter_biomass                   double precision,
    litter_carbon              double precision,
    duff_tc_cond              numeric(14,12),
    duff_tc_unadj              numeric(14,12),
    duff_tc_adj              numeric(14,12),
    avg_wood_density                   numeric(12,10),
    created_by              varchar (60),
    created_date              timestamp without time zone,
    created_in_instance                   varchar (100),
    modified_by              varchar (60),
    modified_date              timestamp without time zone,
    modified_in_instance                   varchar (100),
    cycle              integer,
    subcycle              integer,
    unitcd              integer,
    rscd              integer
);

-- create indexes ...
ALTER TABLE cond_dwm_calc ADD CONSTRAINT cond_dwm_calc_pkey PRIMARY
```

```
KEY (cn);
--ALTER TABLE cond_dwm_calc ADD CONSTRAINT cond_dwm_calc_ukey UNIQUE
(plt_cn, condid, evalid, rscd);
--CREATE INDEX cond_dwm_calc_condid_idx ON cond_dwm_calc (condid);
--CREATE INDEX cond_dwm_calc_evalid_idx ON cond_dwm_calc (evalid);

DROP TABLE IF EXISTS county;
CREATE TABLE county
 (
	statecd			integer,
	unitcd			integer,
	countycd			integer,
	countynm			varchar (100),
	cn		varchar (68),
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...
ALTER TABLE county ADD CONSTRAINT county_pkey PRIMARY KEY (cn);
--ALTER TABLE county ADD CONSTRAINT county_ukey UNIQUE (statecd, countycd);

DROP TABLE IF EXISTS dwm_coarse_woody_debris;
CREATE TABLE dwm_coarse_woody_debris
 (
	cn		varchar (68),
	plt_cn			varchar (68),
	invyr		integer,
	statecd			integer,
	countycd			integer,
	plot		integer,
	subp		integer,
	transect			integer,
	cwdid		double precision,
	measyear			integer,
	condid			integer,
	slopdist			double precision,
	horiz_dist			double precision,
	spcd		integer,
	decaycd			integer,
```

```
	transdia			integer,
	smalldia			integer,
	largedia			integer,
	length			integer,
	hollowcd			varchar (2),
	cwdhstcd			integer,
	volcf		double precision,
	drybio			double precision,
	carbon			double precision,
	cover_pct			double precision,
	lpa_unadj			double precision,
	lpa_plot			double precision,
	lpa_cond			double precision,
	lpa_unadj_rgn			double precision,
	lpa_plot_rgn			double precision,
	lpa_cond_rgn			double precision,
	cover_pct_rgn			double precision,
	chrcd_pnwrs			integer,
	orntcd_pnwrs			varchar (2),
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...
ALTER TABLE dwm_coarse_woody_debris ADD CONSTRAINT
dwm_coarse_woody_debris_pkey PRIMARY KEY (cn);
--ALTER TABLE dwm_coarse_woody_debris ADD CONSTRAINT
dwm_coarse_woody_debris_ukey UNIQUE (plt_cn, transect, subp, cwdid);
--CREATE INDEX dwm_coarse_woody_debris_condid_idx ON dwm_coarse_woody_debris
(condid);
--CREATE INDEX dwm_coarse_woody_debris_cwdid_idx ON dwm_coarse_woody_debris
(cwdid);

DROP TABLE IF EXISTS dwm_duff_litter_fuel;
CREATE TABLE dwm_duff_litter_fuel
 (
	cn		varchar (68),
	plt_cn			varchar (68),
	invyr		integer,
	statecd			integer,
```

```
        countycd                integer,
        plot            integer,
        transect                integer,
        subp            integer,
        smploccd                integer,
        measyear                integer,
        smpldcd                 integer,
        condid                  integer,
        duffdep                 double precision,
        littdep                 double precision,
        fueldep                 double precision,
        created_by              varchar (60),
        created_date            timestamp without time zone,
        created_in_instance             varchar (12),
        modified_by             varchar (60),
        modified_date           timestamp without time zone,
        modified_in_instance            varchar (12)
);

-- create indexes ...
ALTER TABLE dwm_duff_litter_fuel ADD CONSTRAINT dwm_duff_litter_fuel_pkey
PRIMARY KEY (cn);
--ALTER TABLE dwm_duff_litter_fuel ADD CONSTRAINT dwm_duff_litter_fuel_ukey
UNIQUE (plt_cn, transect, subp, smploccd);

DROP TABLE IF EXISTS dwm_fine_woody_debris;
CREATE TABLE dwm_fine_woody_debris
 (
        cn              varchar (68),
        plt_cn                  varchar (68),
        invyr           integer,
        statecd                 integer,
        countycd                integer,
        plot            integer,
        transect                integer,
        subp            integer,
        condid                  integer,
        measyear                integer,
        smallct                 integer,
        mediumct                integer,
        largect                 integer,
        rsnctcd                 integer,
        pilescd                 integer,
        small_tl_cond           double precision,
```

```
    small_tl_plot          double precision,
    small_tl_unadj              double precision,
    medium_tl_cond               double precision,
    medium_tl_plot              double precision,
    medium_tl_unadj              double precision,
    large_tl_cond          double precision,
    large_tl_plot          double precision,
    large_tl_unadj              double precision,
    created_by             varchar (60),
    created_date           timestamp without time zone,
    created_in_instance          varchar (12),
    modified_by            varchar (60),
    modified_date          timestamp without time zone,
    modified_in_instance         varchar (12)
);

-- create indexes ...
ALTER TABLE dwm_fine_woody_debris ADD CONSTRAINT
dwm_fine_woody_debris_pkey PRIMARY KEY (cn);
--ALTER TABLE dwm_fine_woody_debris ADD CONSTRAINT
dwm_fine_woody_debris_ukey UNIQUE (plt_cn, transect, subp, condid);

DROP TABLE IF EXISTS dwm_microplot_fuel;
CREATE TABLE dwm_microplot_fuel
 (
    cn           varchar (68),
    plt_cn            varchar (68),
    invyr         integer,
    statecd           integer,
    countycd            integer,
    plot         integer,
    subp          integer,
    measyear            integer,
    lvshrbcd           integer,
    dshrbcd            integer,
    lvhrbcd            integer,
    dhrbcd             integer,
    littercd          double precision,
    lvshrbht           double precision,
    dshrbht            double precision,
    lvhrbht            double precision,
    dhrbht             double precision,
    created_by             varchar (60),
    created_date           timestamp without time zone,
```

```
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...
ALTER TABLE dwm_microplot_fuel ADD CONSTRAINT dwm_microplot_fuel_pkey
PRIMARY KEY (cn);
--ALTER TABLE dwm_microplot_fuel ADD CONSTRAINT dwm_microplot_fuel_ukey
UNIQUE (plt_cn, subp);

DROP TABLE IF EXISTS dwm_residual_pile;
CREATE TABLE dwm_residual_pile
 (
	cn		varchar (510),
	plt_cn		varchar (510),
	invyr		integer,
	statecd		integer,
	countycd		integer,
	plot		integer,
	subp		integer,
	pile		integer,
	measyear		integer,
	condid		integer,
	shapecd		integer,
	azimuth		integer,
	density		integer,
	height1		integer,
	width1		integer,
	length1		integer,
	height2		integer,
	width2		integer,
	length2		integer,
	volcf		varchar (510),
	drybio		varchar (510),
	carbon		varchar (510),
	ppa_unadj		varchar (510),
	ppa_plot		varchar (510),
	ppa_cond		varchar (510),
	created_by		varchar (510),
	created_date		timestamp without time zone,
	created_in_instance		varchar (510),
	modified_by		varchar (510),
```

```
	modified_in_instance		varchar (510),
	modified_date		timestamp without time zone
);

-- create indexes ...
ALTER TABLE dwm_residual_pile ADD CONSTRAINT dwm_residual_pile_pkey
PRIMARY KEY (cn);
--ALTER TABLE dwm_residual_pile ADD CONSTRAINT dwm_residual_pile_ukey UNIQUE
(plt_cn, subp, transect, segmnt);
--CREATE INDEX dwm_residual_pile_condid_idx ON dwm_residual_pile (condid);

DROP TABLE IF EXISTS dwm_transect_segment;
CREATE TABLE dwm_transect_segment
 (
	cn		varchar (68),
	plt_cn		varchar (68),
	invyr		integer,
	statecd		integer,
	countycd		integer,
	plot		integer,
	subp		integer,
	transect		integer,
	segmnt		integer,
	measyear		integer,
	condid		integer,
	slope_begndist		double precision,
	slope_enddist		double precision,
	slope		integer,
	horiz_length		double precision,
	horiz_begndist		double precision,
	horiz_enddist		double precision,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_in_instance		varchar (12),
	modified_date		timestamp without time zone
);

-- create indexes ...
ALTER TABLE dwm_transect_segment ADD CONSTRAINT dwm_transect_segment_pkey
PRIMARY KEY (cn);
--ALTER TABLE dwm_transect_segment ADD CONSTRAINT dwm_transect_segment_pkey
PRIMARY KEY (cn);
```

```
--CREATE INDEX dwm_transect_segment_condid_idx ON dwm_transect_segment (condid);

DROP TABLE IF EXISTS dwm_visit;
CREATE TABLE dwm_visit
 (
	cn		varchar (68),
	plt_cn			varchar (68),
	invyr		integer,
	statecd			integer,
	countycd			integer,
	plot		integer,
	measday			integer,
	measmon			integer,
	measyear			integer,
	qastatcd			integer,
	crwtypcd			integer,
	smpkndcd			integer,
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...
ALTER TABLE dwm_visit ADD CONSTRAINT dwm_visit_pkey PRIMARY KEY (cn);
--ALTER TABLE dwm_visit ADD CONSTRAINT dwm_visit_ukey UNIQUE (plt_cn);

DROP TABLE IF EXISTS lichen_lab;
CREATE TABLE lichen_lab
 (
	cn		varchar (68),
	plt_cn			varchar (68),
	invyr		integer,
	statecd			integer,
	countycd			integer,
	plot		integer,
	lich_sppcd			integer,
	measyear			integer,
	abundance_class			integer,
	origin_flag			integer,
	spp_comments			text,
	created_by			varchar (60),
```

```
    created_date              timestamp without time zone,
    created_in_instance              varchar (12),
    modified_by               varchar (60),
    modified_date             timestamp without time zone,
    modified_in_instance              varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS lichen_plot_summary;
CREATE TABLE lichen_plot_summary
 (
    cn                varchar (68),
    plt_cn                 varchar (68),
    invyr             integer,
    statecd                integer,
    countycd                integer,
    plot              integer,
    measyear                integer,
    summation                double precision,
    richness                integer,
    evenness                real,
    diversity               real,
    created_by                varchar (60),
    created_date              timestamp without time zone,
    created_in_instance              varchar (12),
    modified_by               varchar (60),
    modified_date             timestamp without time zone,
    modified_in_instance              varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS lichen_species_summary;
CREATE TABLE lichen_species_summary
 (
    cn                varchar (68),
    invyr             integer,
    lichen_region                integer,
    lich_sppcd               integer,
    measyear                integer,
    lichen_region_descr              varchar (160),
    spp_acronym               varchar (12),
    genus             varchar (80),
```

```
	sum_abundance		real,
	frequency_pct		integer,
	species			varchar (100),
	plots_in_region		integer,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance	varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance	varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS lichen_visit;
CREATE TABLE lichen_visit
 (
	cn		varchar (68),
	plt_cn		varchar (68),
	invyr		integer,
	statecd		integer,
	countycd	integer,
	plot		integer,
	measday		integer,
	measmon		integer,
	measyear	integer,
	lichen_statcd	integer,
	liprojcd	integer,
	smplstrt	integer,
	smplstp		integer,
	smpltime	integer,
	sftwdpct	integer,
	hrdwdpct	integer,
	shrubpct	integer,
	gappct		integer,
	gaprcnt		integer,
	tallshrb	integer,
	ftrcd1		double precision,
	ftrcd2		double precision,
	ftrcd3		double precision,
	ftrcd4		double precision,
	issuecd1	double precision,
	issuecd2	double precision,
	issuecd3	double precision,
```

```
    issuecd4                double precision,
    szclscd1                integer,
    szclscd2                integer,
    szclscd3                integer,
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance     varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance    varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS ozone_biosite_summary;
CREATE TABLE ozone_biosite_summary
 (
    cn                      varchar (68),
    invyr                   integer,
    statecd                 integer,
    countycd                integer,
    o3plot                  double precision,
    field_id                integer,
    location_cnt            double precision,
    ground_loc_cd           integer,
    measyear                integer,
    plant_inj_cnt           double precision,
    plant_eval_cnt          double precision,
    plant_ratio             double precision,
    species_eval_cnt        double precision,
    biosite_index           double precision,
    biosite_index_multiplier double precision,
    svrty_class_zero        double precision,
    svrty_class_one         double precision,
    svrty_class_two         double precision,
    svrty_class_three       double precision,
    svrty_class_four        double precision,
    svrty_class_five        double precision,
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance     varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance    varchar (12)
```

```
);

-- create indexes ...

DROP TABLE IF EXISTS ozone_plot;
CREATE TABLE ozone_plot
 (
	cn			varchar (68),
	srv_cn			varchar (68),
	cty_cn			varchar (68),
	invyr			integer,
	statecd			integer,
	unitcd			integer,
	countycd		integer,
	o3plot			double precision,
	field_id		integer,
	split_plotid		integer,
	measyear		integer,
	measmon			integer,
	measday			integer,
	lat			double precision,
	lon			double precision,
	elevation		double precision,
	manual			real,
	qa_status		integer,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance	varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance	varchar (12),
	cycle			integer,
	subcycle		integer
);

-- create indexes ...

DROP TABLE IF EXISTS ozone_plot_summary;
CREATE TABLE ozone_plot_summary
 (
	cn			varchar (68),
	invyr			integer,
	statecd			integer,
	countycd		integer,
```

```
    o3plot                  double precision,
    field_id                integer,
    split_plotid            integer,
    measyear                integer,
    species_eval_cnt        double precision,
    biosite_index           double precision,
    elev                    integer,
    pltsize                 double precision,
    aspect                  integer,
    terrpos                 double precision,
    soildpth                double precision,
    soildrn                 double precision,
    plotwet                 double precision,
    pltdstrb                double precision,
    biosite_index_multiplier    double precision,
    lat                     double precision,
    lon                     double precision,
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance     varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance    varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS ozone_species_summary;
CREATE TABLE ozone_species_summary
 (
    cn                      varchar (68),
    invyr                   integer,
    statecd                 integer,
    countycd                integer,
    o3plot                  double precision,
    field_id                integer,
    split_plotid            integer,
    ground_loc_cd           integer,
    measyear                integer,
    biospcd                 double precision,
    amnt_max                double precision,
    amnt_min                double precision,
    amnt_mean               double precision,
    svrty_max               double precision,
```

```sql
    svrty_min              double precision,
    svrty_mean             double precision,
    plant_inj_cnt          double precision,
    plant_eval_cnt         double precision,
    plant_ratio            double precision,
    biospcd_sum            double precision,
    biospcd_index          double precision,
    elev                   integer,
    pltsize                double precision,
    aspect                 integer,
    terrpos                double precision,
    soildpth               double precision,
    soildrn                double precision,
    plotwet                double precision,
    pltdstrb               double precision,
    created_by             varchar (60),
    created_date           timestamp without time zone,
    created_in_instance    varchar (12),
    modified_by            varchar (60),
    modified_date          timestamp without time zone,
    modified_in_instance   varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS ozone_validation;
CREATE TABLE ozone_validation
 (
    cn                     varchar (68),
    plt_cn                 varchar (68),
    invyr                  integer,
    statecd                integer,
    countycd               integer,
    o3plot                 double precision,
    field_id               integer,
    split_plotid           integer,
    biospcd                double precision,
    qastatcd               integer,
    crwtypcd               integer,
    leafvchr               double precision,
    injvalid               double precision,
    o3_statcd              double precision,
    measyear               integer,
    created_by             varchar (60),
```

```
        created_date                timestamp without time zone,
        created_in_instance         varchar (12),
        modified_by                 varchar (60),
        modified_date               timestamp without time zone,
        modified_in_instance        varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS ozone_visit;
CREATE TABLE ozone_visit
 (
        cn                          varchar (68),
        plt_cn                      varchar (68),
        invyr                       integer,
        statecd                     integer,
        countycd                    integer,
        o3plot                      double precision,
        field_id                    integer,
        split_plotid                integer,
        smpkndcd                    integer,
        measday                     integer,
        measmon                     integer,
        measyear                    integer,
        pltsize                     double precision,
        aspect                      integer,
        terrpos                     double precision,
        soildpth                    double precision,
        soildrn                     double precision,
        pltdstrb                    double precision,
        crwtypcd                    integer,
        plotwet                     double precision,
        injcheck                    integer,
        gridden                     integer,
        created_by                  varchar (60),
        created_date                timestamp without time zone,
        created_in_instance         varchar (12),
        modified_by                 varchar (60),
        modified_date               timestamp without time zone,
        modified_in_instance        varchar (12)
);

-- create indexes ...
```

```sql
DROP TABLE IF EXISTS p2veg_subp_structure;
CREATE TABLE p2veg_subp_structure
 (
	cn			varchar (68),
	plt_cn			varchar (68),
	statecd			integer,
	unitcd			integer,
	countycd		integer,
	plot			integer,
	invyr			integer,
	subp			integer,
	condid			integer,
	growth_habit_cd		varchar (4),
	layer			integer,
	cover_pct		integer,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance	varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance	varchar (12),
	cycle			integer,
	subcycle		integer
);

-- create indexes ...
ALTER TABLE p2veg_subp_structure ADD CONSTRAINT p2veg_subp_structure_pkey
PRIMARY KEY (cn);
DROP TABLE IF EXISTS p2veg_subplot_spp;
CREATE TABLE p2veg_subplot_spp
 (
	cn			varchar (68),
	plt_cn			varchar (68),
	invyr			integer,
	statecd			integer,
	unitcd			integer,
	countycd		integer,
	plot			integer,
	subp			integer,
	condid			integer,
	veg_fldspcd		varchar (20),
	unique_sp_nbr		integer,
	veg_spcd		varchar (20),
	growth_habit_cd		varchar (4),
```

```
	layer		integer,
	cover_pct			integer,
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12),
	cycle		integer,
	subcycle			integer
);

-- create indexes ...
ALTER TABLE p2veg_subplot_spp ADD CONSTRAINT p2veg_subplot_spp_pkey
PRIMARY KEY (cn);
--ALTER TABLE p2veg_subplot_spp ADD CONSTRAINT p2veg_subplot_spp_ukey UNIQUE
(plt_cn, veg_fldspcd, unique_sp_nbr, subp, condid);

DROP TABLE IF EXISTS plot CASCADE;
CREATE TABLE plot
 (
	cn		varchar (68),
	srv_cn			varchar (68),
	cty_cn			varchar (68),
	prev_plt_cn			varchar (68),
	invyr		integer,
	statecd			integer,
	unitcd			integer,
	countycd			integer,
	plot		integer,
	plot_status_cd			integer,
	plot_nonsample_reasn_cd			integer,
	measyear			integer,
	measmon			integer,
	measday			integer,
	remper			numeric(3,1),
	kindcd			integer,
	designcd			integer,
	rddistcd			integer,
	watercd			integer,
	lat		numeric(8,6),
	lon		numeric(9,6),
	elev		integer,
	grow_typ_cd			integer,
```

```
        mort_typ_cd                 integer,
        p2panel                     integer,
        p3panel                     integer,
        ecosubcd                    varchar (14),
        congcd                      integer,
        manual                      numeric(3,1),
        subpanel                    integer,
        kindcd_nc                   integer,
        qa_status                   integer,
        created_by                  varchar (60),
        created_date                timestamp without time zone,
        created_in_instance         varchar (12),
        modified_by                 varchar (60),
        modified_date               timestamp without time zone,
        modified_in_instance        varchar (12),
        microplot_loc               varchar (24),
        declination                 numeric(4,1),
        emap_hex                    integer,
        samp_method_cd              integer,
        subp_examine_cd             integer,
        macro_breakpoint_dia        integer,
        intensity                   varchar (4),
        cycle                       integer,
        subcycle                    integer,
        eco_unit_pnw                varchar (20),
        topo_position_pnw           varchar (4),
        nf_sampling_status_cd       integer,
        nf_plot_status_cd           integer,
        nf_plot_nonsample_reasn_cd  integer,
        p2veg_sampling_status_cd    integer,
        p2veg_sampling_level_detail_cd  integer,
        invasive_sampling_status_cd integer,
        invasive_specimen_rule_cd   integer,
        designcd_p2a                integer
);

-- create indexes ...
ALTER TABLE plot ADD CONSTRAINT plot_pkey PRIMARY KEY (cn);
--ALTER TABLE plot ADD CONSTRAINT plot_ukey UNIQUE (statecd, invyr, unitcd,
countycd, plot);

DROP TABLE IF EXISTS plotgeom;
CREATE TABLE plotgeom
 (
```

```
	cn			varchar (68),
	statecd			integer,
	invyr			integer,
	unitcd			integer,
	countycd		integer,
	plot			integer,
	lat			double precision,
	lon			double precision,
	congcd			integer,
	ecosubcd		varchar (14),
	huc			integer,
	emap_hex		integer,
	fipscounty		integer,
	roadlesscd		varchar (8),
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance	integer,
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance	integer,
	adforcd			integer
);

-- create indexes ...
ALTER TABLE plotgeom ADD CONSTRAINT plotgeom_pkey PRIMARY KEY (cn);

DROP TABLE IF EXISTS plotsnap;
CREATE TABLE plotsnap
 (
	cn			varchar (68),
	srv_cn			varchar (68),
	cty_cn			varchar (68),
	prev_plt_cn		varchar (68),
	invyr			integer,
	statecd			integer,
	unitcd			integer,
	countycd		integer,
	plot			integer,
	plot_status_cd		integer,
	plot_nonsample_reasn_cd	integer,
	measyear		integer,
	measmon			integer,
	measday			integer,
	remper			real,
```

```
kindcd                  integer,
designcd                integer,
rddistcd                integer,
watercd                 integer,
lat            double precision,
lon            double precision,
elev           integer,
grow_typ_cd             integer,
mort_typ_cd             integer,
p2panel                 integer,
p3panel                 integer,
ecosubcd                varchar (14),
congcd                  integer,
manual                  real,
subpanel                integer,
kindcd_nc               integer,
qa_status               integer,
created_by              varchar (60),
created_date            timestamp without time zone,
created_in_instance             varchar (12),
modified_by             varchar (60),
modified_date           timestamp without time zone,
modified_in_instance            varchar (12),
microplot_loc           varchar (24),
declination             real,
emap_hex                integer,
samp_method_cd                  integer,
subp_examine_cd                 integer,
macro_breakpoint_dia            integer,
intensity               varchar (4),
cycle          integer,
subcycle                integer,
eco_unit_pnw            varchar (20),
topo_position_pnw               varchar (4),
eval_grp_cn             varchar (68),
eval_grp                integer,
expall                  double precision,
expcurr                 double precision,
expvol                  double precision,
expgrow                 double precision,
expmort                 double precision,
expremv                 double precision,
adj_expall              double precision,
adj_expcurr             double precision,
```

```
    adj_expvol_macr                 double precision,
    adj_expvol_subp                 double precision,
    adj_expvol_micr                 double precision,
    adj_expgrow_macr                double precision,
    adj_expgrow_subp                double precision,
    adj_expgrow_micr                double precision,
    adj_expmort_macr                double precision,
    adj_expmort_subp                double precision,
    adj_expmort_micr                double precision,
    adj_expremv_macr                double precision,
    adj_expremv_subp                double precision,
    adj_expremv_micr                double precision
);

-- create indexes ...
ALTER TABLE plotsnap ADD CONSTRAINT plotsnap_pkey PRIMARY KEY (cn,
eval_grp_cn);
--CREATE INDEX plotsnap_plotsnap_plot_idx_idx ON plotsnap (eval_grp, cn);

DROP TABLE IF EXISTS pop_estn_unit cascade;
CREATE TABLE pop_estn_unit
 (
    cn                  varchar (68),
    eval_cn             varchar (68),
    rscd                integer,
    evalid              integer,
    estn_unit           integer,
    estn_unit_descr     varchar (510),
    statecd             integer,
    arealand_eu         double precision,
    areatot_eu          double precision,
    area_used           double precision,
    area_source         varchar (100),
    p1pntcnt_eu         double precision,
    p1source            varchar (60),
    created_by          varchar (60),
    created_date        timestamp without time zone,
    created_in_instance varchar (12),
    modified_by         varchar (60),
    modified_date       timestamp without time zone,
    modified_in_instance varchar (12)
);

-- create indexes ...
```

```sql
ALTER TABLE pop_estn_unit ADD CONSTRAINT pop_estn_unit_pkey PRIMARY KEY
(cn);
--ALTER TABLE pop_estn_unit ADD CONSTRAINT pop_estn_unit_ukey UNIQUE (rscd,
evalid, estn_unit);
CREATE INDEX pop_estn_unit_eval_cn_idx ON pop_estn_unit (eval_cn);

DROP TABLE IF EXISTS pop_eval CASCADE;
CREATE TABLE pop_eval
 (
	cn			varchar (68),
	eval_grp_cn			varchar (68),
	rscd			integer,
	evalid			integer,
	eval_descr			varchar (510),
	statecd			integer,
	location_nm			varchar (510),
	report_year_nm			varchar (510),
	start_invyr			integer,
	end_invyr			integer,
	land_only			varchar (510),
	timberland_only			varchar (2),
	growth_acct			varchar (2),
	estn_method			varchar (80),
	notes			text,
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...
ALTER TABLE pop_eval ADD CONSTRAINT pop_eval_pkey PRIMARY KEY (cn);
--ALTER TABLE pop_eval ADD CONSTRAINT pop_eval_ukey UNIQUE (rscd, evalid);
CREATE INDEX pop_eval_timberland_only_idx ON pop_eval (timberland_only);

DROP TABLE IF EXISTS pop_eval_attribute;
CREATE TABLE pop_eval_attribute
 (
	cn			varchar (68),
	eval_cn			varchar (68),
	attribute_nbr			integer,
	statecd			integer,
```

```
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance             varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance            varchar (12)
);

-- create indexes ...
ALTER TABLE pop_eval_attribute ADD CONSTRAINT pop_eval_attribute_pkey
PRIMARY KEY (cn);
--ALTER TABLE pop_eval_attribute ADD CONSTRAINT pop_eval_attribute_ukey UNIQUE
(eval_cn, attribute_nbr);

DROP TABLE IF EXISTS pop_eval_grp;
CREATE TABLE pop_eval_grp
 (
    cn              varchar (68),
    rscd            integer,
    eval_grp                integer,
    eval_grp_descr                  varchar (510),
    statecd                 integer,
    notes           text,
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance             varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance            varchar (12)
);

-- create indexes ...
ALTER TABLE pop_eval_grp ADD CONSTRAINT pop_eval_grp_pkey PRIMARY KEY
(cn);
ALTER TABLE pop_eval_grp ADD CONSTRAINT pop_eval_grp_ukey UNIQUE (rscd,
eval_grp);
CREATE INDEX pop_eval_grp_eval_grp_idx ON pop_eval_grp (eval_grp, statecd);

DROP TABLE IF EXISTS pop_eval_typ;
CREATE TABLE pop_eval_typ
 (
    cn              varchar (68),
    eval_grp_cn             varchar (68),
    eval_cn                 varchar (68),
```

```
    eval_typ                varchar (30),
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance            varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance           varchar (12)
);

-- create indexes ...
ALTER TABLE pop_eval_typ ADD CONSTRAINT pop_eval_typ_pkey PRIMARY KEY
(cn);
ALTER TABLE pop_eval_typ ADD CONSTRAINT pop_eval_typ_ukey UNIQUE
(eval_grp_cn, eval_cn, eval_typ);

DROP TABLE IF EXISTS pop_plot_stratum_assgn;
CREATE TABLE pop_plot_stratum_assgn
 (
    cn              varchar (68),
    stratum_cn              varchar (68),
    plt_cn              varchar (68),
    statecd             integer,
    invyr           integer,
    unitcd              integer,
    countycd                integer,
    plot            integer,
    rscd            integer,
    evalid              integer,
    estn_unit               integer,
    stratumcd               integer,
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance            varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance           varchar (12)
);

-- create indexes ...
ALTER TABLE pop_plot_stratum_assgn ADD CONSTRAINT pop_plot_stratum_assgn_pkey
PRIMARY KEY (cn);
--ALTER TABLE pop_plot_stratum_assgn ADD CONSTRAINT pop_plot_stratum_assgn_ukey
UNIQUE (rscd, evalid, statecd, countycd, plot);
CREATE INDEX pop_plot_stratum_assgn_stratum_cn_idx ON pop_plot_stratum_assgn
```

```
(stratum_cn);
CREATE INDEX pop_plot_stratum_assgn_plt_cn_idx ON pop_plot_stratum_assgn (plt_cn);

DROP TABLE IF EXISTS pop_stratum;
CREATE TABLE pop_stratum
 (
	cn		varchar (68),
	estn_unit_cn		varchar (68),
	rscd		integer,
	evalid		integer,
	estn_unit		integer,
	stratumcd		integer,
	stratum_descr		varchar (510),
	statecd		integer,
	p1pointcnt		double precision,
	p2pointcnt		double precision,
	expns		double precision,
	adj_factor_macr		real,
	adj_factor_subp		real,
	adj_factor_micr		real,
	adj_factor_cwd		real,
	adj_factor_fwd_sm		real,
	adj_factor_fwd_lg		real,
	adj_factor_duff		real,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...
ALTER TABLE pop_stratum ADD CONSTRAINT pop_stratum_pkey PRIMARY KEY (cn);
CREATE INDEX pop_stratum_estn_unit_cn_idx ON pop_stratum (estn_unit_cn);

DROP TABLE IF EXISTS ref_citation;
CREATE TABLE ref_citation
 (
	citation_nbr		integer,
	citation		text,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
```

```
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance            varchar (12)
);

-- create indexes ...
ALTER TABLE ref_citation ADD CONSTRAINT ref_citation_pkey PRIMARY KEY
(citation_nbr);

DROP TABLE IF EXISTS ref_fiadb_version;
CREATE TABLE ref_fiadb_version
 (
    version             varchar (80),
    install_type            varchar (20),
    descr           text,
    created_by          varchar (60),
    created_date            timestamp without time zone,
    created_in_instance             varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance            varchar (12)
);

-- create indexes ...
ALTER TABLE ref_fiadb_version ADD CONSTRAINT ref_fiadb_version_pkey PRIMARY
KEY (version);

DROP TABLE IF EXISTS ref_forest_type;
CREATE TABLE ref_forest_type
 (
    value           integer,
    meaning             varchar (160),
    typgrpcd            integer,
    manual_start            real,
    manual_end          real,
    allowed_in_field            varchar (2),
    created_by          varchar (60),
    created_date            timestamp without time zone,
    created_in_instance             varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance            varchar (12)
);
```

```
-- create indexes ...
ALTER TABLE ref_forest_type ADD CONSTRAINT ref_forest_type_pkey PRIMARY KEY
(value);

DROP TABLE IF EXISTS ref_forest_type_group;
CREATE TABLE ref_forest_type_group
 (
	value		integer,
	meaning			varchar (160),
	abbr		varchar (80),
	duff_density		double precision,
	duff_carbon_ratio		double precision,
	litter_density		double precision,
	litter_carbon_ratio		double precision,
	pile_density		double precision,
	pile_carbon_ratio		double precision,
	pile_decay_ratio		double precision,
	fwd_density		double precision,
	fwd_carbon_ratio		double precision,
	fwd_decay_ratio		double precision,
	fwd_small_qmd		double precision,
	fwd_medium_qmd			double precision,
	fwd_large_qmd		double precision,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...
ALTER TABLE ref_forest_type_group ADD CONSTRAINT ref_forest_type_group_pkey
PRIMARY KEY (value);

DROP TABLE IF EXISTS ref_habtyp_description;
CREATE TABLE ref_habtyp_description
 (
	cn		varchar (68),
	habtypcd		varchar (20),
	pub_cd		varchar (20),
	scientific_name		varchar (230),
	common_name		varchar (510),
	valid		varchar (2),
```

```
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance             varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance            varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS ref_habtyp_publication;
CREATE TABLE ref_habtyp_publication
 (
    cn              varchar (68),
    pub_cd                  varchar (20),
    title           varchar (400),
    author                  varchar (400),
    type            varchar (20),
    valid           varchar (2),
    created_by              varchar (60),
    created_date            timestamp without time zone,
    created_in_instance             varchar (12),
    modified_by             varchar (60),
    modified_date           timestamp without time zone,
    modified_in_instance            varchar (12)
);

-- create indexes ...
ALTER TABLE ref_habtyp_publication ADD CONSTRAINT ref_habtyp_publication_pkey
PRIMARY KEY (cn);
--ALTER TABLE ref_habtyp_publication ADD CONSTRAINT ref_habtyp_publication_ukey
UNIQUE (pub_cd);

DROP TABLE IF EXISTS ref_invasive_species;
CREATE TABLE ref_invasive_species
 (
    cn              varchar (68),
    statecd                 integer,
    symbol                  varchar (32),
    inv_group_cd            integer,
    unitcd_list             varchar (40),
    start_date              timestamp without time zone,
    end_date                timestamp without time zone,
    manual_start            real,
```

```
	manual_end			real,
	notes			text,
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...
ALTER TABLE ref_invasive_species ADD CONSTRAINT ref_invasive_species_pkey
PRIMARY KEY (cn);
--ALTER TABLE ref_invasive_species ADD CONSTRAINT ref_invasive_species_ukey
UNIQUE (statecd, symbol);

DROP TABLE IF EXISTS ref_lichen_species;
CREATE TABLE ref_lichen_species
 (
	lich_sppcd			integer,
	yearstart			integer,
	yearend			integer,
	spp_acronym			varchar (12),
	genus			varchar (80),
	species			varchar (100),
	cn			varchar (68),
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS ref_lichen_spp_comments;
CREATE TABLE ref_lichen_spp_comments
 (
	lich_sppcd			integer,
	spp_name			varchar (160),
	yearend			integer,
	yearstart			integer,
	spp_comments			text,
```

```
    cn varchar (68),
    created_by varchar (60),
    created_date timestamp without time zone,
    created_in_instance varchar (12),
    modified_by varchar (60),
    modified_date timestamp without time zone,
    modified_in_instance varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS ref_plant_dictionary;
CREATE TABLE ref_plant_dictionary
 (
    cn varchar (68),
    symbol_type varchar (40),
    symbol varchar (32),
    scientific_name varchar (200),
    new_symbol varchar (32),
    new_scientific_name varchar (200),
    common_name varchar (200),
    category varchar (30),
    family varchar (50),
    growth_habit varchar (100),
    duration varchar (100),
    us_nativity varchar (200),
    state_distribution text,
    state_and_province text,
    scientific_name_w_author text,
    genera_binomial_author varchar (200),
    trinomial_author varchar (200),
    quadrinomial_author varchar (200),
    xgenus varchar (2),
    genus varchar (80),
    xspecies varchar (2),
    species varchar (100),
    ssp varchar (8),
    xsubspecies varchar (2),
    subspecies varchar (60),
    var varchar (8),
    xvariety varchar (2),
    variety varchar (60),
    subvar varchar (14),
    subvariety varchar (60),
```

```
	f		varchar (4),
	forma		varchar (60),
	notes		text,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...
ALTER TABLE ref_plant_dictionary ADD CONSTRAINT ref_plant_dictionary_pkey
PRIMARY KEY (cn);
--ALTER TABLE ref_plant_dictionary ADD CONSTRAINT ref_plant_dictionary_ukey
UNIQUE (symbol_type, symbol, new_symbol);

DROP TABLE IF EXISTS ref_pop_attribute;
CREATE TABLE ref_pop_attribute
 (
	cn		varchar (68),
	attribute_nbr		integer,
	attribute_descr		varchar (510),
	timberland		varchar (2),
	eval_typ		varchar (30),
	expression		text,
	where_clause		text,
	footnote		text,
	attribute_glossary		text,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...
ALTER TABLE ref_pop_attribute ADD CONSTRAINT ref_pop_attribute_pkey PRIMARY
KEY (cn);
--ALTER TABLE ref_pop_attribute ADD CONSTRAINT ref_pop_attribute_ukey UNIQUE
(attribute_nbr);

DROP TABLE IF EXISTS ref_pop_eval_typ_descr;
```

```
CREATE TABLE ref_pop_eval_typ_descr
 (
	cn			varchar (510),
	label_order			integer,
	eval_typ			varchar (510),
	eval_typ_label			varchar (510),
	change_eval_typ			varchar (510),
	eval_typ_descr			varchar (510),
	created_by			varchar (510),
	created_date			timestamp without time zone,
	created_in_instance			varchar (510),
	modified_by			varchar (510),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (510)
);

-- create indexes ...
ALTER TABLE ref_pop_eval_typ_descr ADD CONSTRAINT ref_pop_eval_typ_descr_pkey
PRIMARY KEY (cn);
--ALTER TABLE ref_pop_eval_typ_descr ADD CONSTRAINT ref_pop_eval_typ_descr_ukey
UNIQUE (eval_typ);

DROP TABLE IF EXISTS ref_research_station CASCADE;
CREATE TABLE ref_research_station
 (
	statecd			integer,
	rscd			integer,
	rs			varchar (10),
	state_name			varchar (80),
	state_abbr			varchar (8),
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...
ALTER TABLE ref_research_station ADD CONSTRAINT ref_research_station_pkey
PRIMARY KEY (statecd);

DROP TABLE IF EXISTS ref_species CASCADE;
CREATE TABLE ref_species
```

```
(
    spcd                integer,
    common_name                 varchar (200),
    genus               varchar (80),
    species                 varchar (100),
    variety                 varchar (100),
    subspecies                  varchar (100),
    species_symbol                  varchar (16),
    e_spgrpcd                   integer,
    w_spgrpcd                   integer,
    c_spgrpcd                   integer,
    p_spgrpcd                   integer,
    major_spgrpcd                   integer,
    stocking_spgrpcd                    integer,
    forest_type_spgrpcd                 integer,
    exists_in_ncrs                  varchar (2),
    exists_in_ners                  varchar (2),
    exists_in_pnwrs                 varchar (2),
    exists_in_rmrs                  varchar (2),
    exists_in_srs               varchar (2),
    sitetree                varchar (2),
    sftwd_hrdwd                 varchar (2),
    st_exists_in_ncrs                   varchar (2),
    st_exists_in_ners                   varchar (2),
    st_exists_in_pnwrs                  varchar (2),
    st_exists_in_rmrs                   varchar (2),
    st_exists_in_srs                    varchar (2),
    core                varchar (2),
    east                varchar (2),
    west                varchar (2),
    caribbean                   varchar (2),
    pacific                 varchar (2),
    woodland                    varchar (2),
    manual_start                    real,
    manual_end                  real,
    jenkins_spgrpcd                 integer,
    jenkins_total_b1                    double precision,
    jenkins_total_b2                    double precision,
    jenkins_stem_wood_ratio_b1                  double precision,
    jenkins_stem_wood_ratio_b2                  double precision,
    jenkins_stem_bark_ratio_b1                  double precision,
    jenkins_stem_bark_ratio_b2                  double precision,
    jenkins_foliage_ratio_b1                    double precision,
    jenkins_foliage_ratio_b2                    double precision,
```

```
    jenkins_root_ratio_b1               double precision,
    jenkins_root_ratio_b2               double precision,
    jenkins_sapling_adjustment                  double precision,
    wood_spgr_greenvol_drywt                    double precision,
    wood_spgr_greenvol_drywt_cit                integer,
    bark_spgr_greenvol_drywt                    double precision,
    bark_spgr_greenvol_drywt_cit                integer,
    mc_pct_green_bark                   double precision,
    mc_pct_green_bark_cit               double precision,
    mc_pct_green_wood                   double precision,
    mc_pct_green_wood_cit               integer,
    wood_spgr_mc12vol_drywt                     double precision,
    wood_spgr_mc12vol_drywt_cit                 integer,
    bark_vol_pct                double precision,
    bark_vol_pct_cit                integer,
    raile_stump_dob_b1                  double precision,
    raile_stump_dib_b1                  double precision,
    raile_stump_dib_b2                  double precision,
    cwd_decay_ratio1                    double precision,
    cwd_decay_ratio2                    double precision,
    cwd_decay_ratio3                    double precision,
    cwd_decay_ratio4                    double precision,
    cwd_decay_ratio5                    double precision,
    dwm_carbon_ratio                    double precision,
    standing_dead_decay_ratio1                  double precision,
    standing_dead_decay_ratio2                  double precision,
    standing_dead_decay_ratio3                  double precision,
    standing_dead_decay_ratio4                  double precision,
    standing_dead_decay_ratio5                  double precision,
    created_by                  varchar (60),
    created_date                timestamp without time zone,
    created_in_instance                 varchar (12),
    modified_by                 varchar (60),
    modified_date               timestamp without time zone,
    modified_in_instance                varchar (12)
);

-- create indexes ...
ALTER TABLE ref_species ADD CONSTRAINT ref_species_pkey PRIMARY KEY (spcd);
--ALTER TABLE ref_species ADD CONSTRAINT ref_species_ukey UNIQUE
(species_symbol);

DROP TABLE IF EXISTS ref_species_group CASCADE;
CREATE TABLE ref_species_group
```

```
(
    spgrpcd                 integer,
    name                varchar (80),
    region                  varchar (16),
    class               varchar (16),
    created_by                  varchar (60),
    created_date                timestamp without time zone,
    created_in_instance                 varchar (12),
    modified_by                 varchar (60),
    modified_date               timestamp without time zone,
    modified_in_instance                varchar (12)
);

-- create indexes ...
ALTER TABLE ref_species_group ADD CONSTRAINT ref_species_group_pkey PRIMARY
KEY (spgrpcd);

DROP TABLE IF EXISTS ref_state_elev CASCADE;
CREATE TABLE ref_state_elev
(
    statecd                 integer,
    min_elev                    integer,
    max_elev                    integer,
    lowest_point                varchar (60),
    highest_point               varchar (60),
    created_by                  varchar (60),
    created_date                timestamp without time zone,
    created_in_instance                 varchar (12),
    modified_by                 varchar (60),
    modified_date               timestamp without time zone,
    modified_in_instance                varchar (12)
);

-- create indexes ...
ALTER TABLE ref_state_elev ADD CONSTRAINT ref_state_elev_pkey PRIMARY KEY
(statecd);

DROP TABLE IF EXISTS ref_unit CASCADE;
CREATE TABLE ref_unit
(
    statecd                 integer,
    value               integer,
    meaning                 varchar (160),
    created_by                  varchar (60),
```

```
    created_date        timestamp without time zone,
    created_in_instance     varchar (12),
    modified_by         varchar (60),
    modified_date       timestamp without time zone,
    modified_in_instance    varchar (12)
);

-- create indexes ...
ALTER TABLE ref_unit ADD CONSTRAINT ref_unit_pkey PRIMARY KEY (statecd, value);

DROP TABLE IF EXISTS seedling CASCADE;
CREATE TABLE seedling
 (
    cn          varchar (68),
    plt_cn          varchar (68),
    invyr           integer,
    statecd         integer,
    unitcd          integer,
    countycd        integer,
    plot            integer,
    subp            integer,
    condid          integer,
    spcd            integer,
    spgrpcd         integer,
    stocking        real,
    treecount       integer,
    totage          integer,
    created_by          varchar (60),
    created_date        timestamp without time zone,
    created_in_instance     varchar (12),
    modified_by         varchar (60),
    modified_date       timestamp without time zone,
    modified_in_instance    varchar (12),
    treecount_calc      integer,
    tpa_unadj           double precision,
    cycle           integer,
    subcycle        integer
);

-- create indexes ...
ALTER TABLE seedling ADD CONSTRAINT seedling_pkey PRIMARY KEY (cn);
--ALTER TABLE seedling ADD CONSTRAINT seedling_ukey UNIQUE (plt_cn, subp, condid, spcd);
```

```
DROP TABLE IF EXISTS sitetree;
CREATE TABLE sitetree
 (
	cn			varchar (68),
	plt_cn			varchar (68),
	prev_sit_cn		varchar (68),
	invyr			integer,
	statecd			integer,
	unitcd			integer,
	countycd		integer,
	plot			integer,
	condid			integer,
	tree			integer,
	spcd			integer,
	dia			real,
	ht			integer,
	agedia			integer,
	spgrpcd			integer,
	sitree			integer,
	sibase			integer,
	subp			integer,
	azimuth			integer,
	dist			real,
	method			integer,
	sitree_est		integer,
	validcd			integer,
	condlist		integer,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12),
	cycle			integer,
	subcycle		integer
);

-- create indexes ...
ALTER TABLE sitetree ADD CONSTRAINT sitetree_pkey PRIMARY KEY (cn);
--ALTER TABLE sitetree ADD CONSTRAINT sitetree_ukey UNIQUE (plt_cn, condid, tree);

DROP TABLE IF EXISTS soils_erosion;
CREATE TABLE soils_erosion
```

```
(
    cn                  varchar (68),
    plt_cn              varchar (68),
    invyr               integer,
    statecd             integer,
    countycd            integer,
    plot                integer,
    subp                integer,
    measyear            integer,
    soilspct            double precision,
    compcpct            double precision,
    typrtdcd            double precision,
    typcmpcd            double precision,
    typareacd           double precision,
    typothrcd           double precision,
    created_by          varchar (60),
    created_date        timestamp without time zone,
    created_in_instance varchar (12),
    modified_by         varchar (60),
    modified_date       timestamp without time zone,
    modified_in_instance varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS soils_lab;
CREATE TABLE soils_lab
 (
    cn                  varchar (68),
    plt_cn              varchar (68),
    invyr               integer,
    statecd             integer,
    countycd            integer,
    plot                integer,
    smplnnbr            double precision,
    vstnbr              integer,
    layer_type          varchar (20),
    sampler_type        varchar (4),
    qastatcd            integer,
    sample_date         timestamp without time zone,
    lab_id              varchar (20),
    sample_id           varchar (24),
    field_moist_soil_wt double precision,
    air_dry_soil_wt     double precision,
```

```
    oven_dry_soil_wt                    double precision,
    field_moist_water_content_pct                double precision,
    residual_water_content_pct                double precision,
    total_water_content_pct                double precision,
    bulk_density                double precision,
    coarse_fraction_pct                double precision,
    c_org_pct                double precision,
    c_inorg_pct                double precision,
    c_total_pct                double precision,
    n_total_pct                double precision,
    ph_h2o                double precision,
    ph_cacl2                double precision,
    exchng_na                double precision,
    exchng_k                double precision,
    exchng_mg                double precision,
    exchng_ca                double precision,
    exchng_al                double precision,
    ecec            double precision,
    exchng_mn                double precision,
    exchng_fe                double precision,
    exchng_ni                double precision,
    exchng_cu                double precision,
    exchng_zn                double precision,
    exchng_cd                double precision,
    exchng_pb                double precision,
    exchng_s                double precision,
    bray1_p                double precision,
    olsen_p                double precision,
    measyear                integer,
    modified_by                varchar (60),
    modified_date                timestamp without time zone,
    modified_in_instance                varchar (12),
    created_by                varchar (60),
    created_date                timestamp without time zone,
    created_in_instance                varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS soils_sample_loc;
CREATE TABLE soils_sample_loc
 (
    cn              varchar (68),
    plt_cn                varchar (68),
```

```
    invyr            integer,
    statecd          integer,
    countycd         integer,
    plot             integer,
    smplnnbr         integer,
    measyear         integer,
    forflthk         double precision,
    ltrlrthk         double precision,
    forflthkn        double precision,
    ltrlrthkn        double precision,
    forflthks        double precision,
    ltrlrthks        double precision,
    forflthke        double precision,
    ltrlrthke        double precision,
    forflthkw        double precision,
    ltrlrthkw        double precision,
    condid           integer,
    vstnbr           integer,
    txtrlyr1         double precision,
    txtrlyr2         double precision,
    dpthsbsl         double precision,
    soils_statcd     integer,
    created_by       varchar (60),
    created_date     timestamp without time zone,
    created_in_instance      varchar (12),
    modified_by      varchar (60),
    modified_date    timestamp without time zone,
    modified_in_instance     varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS soils_visit;
CREATE TABLE soils_visit
 (
    cn               varchar (68),
    plt_cn           varchar (68),
    invyr            integer,
    statecd          integer,
    countycd         integer,
    plot             integer,
    measday          integer,
    measmon          integer,
    measyear         integer,
```

```
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS subp_cond CASCADE;
CREATE TABLE subp_cond
 (
	cn		varchar (68),
	plt_cn		varchar (68),
	invyr		integer,
	statecd		integer,
	unitcd		integer,
	countycd		integer,
	plot		integer,
	subp		integer,
	condid		integer,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12),
	micrcond_prop		real,
	subpcond_prop		real,
	macrcond_prop		real,
	nonfr_incl_pct_subp		integer,
	nonfr_incl_pct_macro		integer,
	cycle		integer,
	subcycle		integer
);

-- create indexes ...
ALTER TABLE subp_cond ADD CONSTRAINT subp_cond_pkey PRIMARY KEY (cn);
--ALTER TABLE subp_cond ADD CONSTRAINT subp_cond_ukey UNIQUE (plt_cn, subp,
condid);

DROP TABLE IF EXISTS subp_cond_chng_mtrx;
CREATE TABLE subp_cond_chng_mtrx
```

```
(
    cn                  varchar (68),
    statecd             integer,
    subp                integer,
    subptyp             integer,
    plt_cn              varchar (68),
    condid              integer,
    prev_plt_cn         varchar (68),
    prevcond            integer,
    subptyp_prop_chng   numeric(5,4),
    created_by          varchar (60),
    created_date        timestamp without time zone,
    created_in_instance varchar (12),
    modified_by         varchar (60),
    modified_date       timestamp without time zone,
    modified_in_instance varchar (12)
);

-- create indexes ...
ALTER TABLE subp_cond_chng_mtrx ADD CONSTRAINT subp_cond_chng_mtrx_pkey
PRIMARY KEY (cn);
--ALTER TABLE subp_cond_chng_mtrx ADD CONSTRAINT subp_cond_chng_mtrx_ukey
UNIQUE (plt_cn, prev_plt_cn, subp, subptyp, condid, prevcond);

DROP TABLE IF EXISTS subplot CASCADE;
CREATE TABLE subplot
(
    cn                  varchar (68),
    plt_cn              varchar (68),
    prev_sbp_cn         varchar (68),
    invyr               integer,
    statecd             integer,
    unitcd              integer,
    countycd            integer,
    plot                integer,
    subp                integer,
    subp_status_cd      integer,
    point_nonsample_reasn_cd integer,
    micrcond            integer,
    subpcond            integer,
    macrcond            integer,
    condlist            integer,
    slope               integer,
    aspect              integer,
```

```
        waterdep                    double precision,
        p2a_grm_flg                 varchar (2),
        created_by                  varchar (60),
        created_date                timestamp without time zone,
        created_in_instance                 varchar (12),
        modified_by                 varchar (60),
        modified_date               timestamp without time zone,
        modified_in_instance                varchar (12),
        cycle           integer,
        subcycle                integer,
        root_dis_sev_cd_pnwrs               integer,
        nf_subp_status_cd               integer,
        nf_subp_nonsample_reasn_cd                  integer,
        p2veg_subp_status_cd                integer,
        p2veg_subp_nonsample_reasn_cd               integer,
        invasive_subp_status_cd                 integer,
        invasive_nonsample_reasn_cd                 integer
);

-- create indexes ...
ALTER TABLE subplot ADD CONSTRAINT subplot_pkey PRIMARY KEY (cn);
--ALTER TABLE subplot ADD CONSTRAINT subplot_ukey UNIQUE (plt_cn, subp);

DROP TABLE IF EXISTS survey CASCADE;
CREATE TABLE survey
 (
        cn              varchar (68),
        invyr           integer,
        p3_ozone_ind                varchar (2),
        statecd                 integer,
        stateab                 varchar (4),
        statenm                 varchar (56),
        rscd            integer,
        ann_inventory               varchar (2),
        notes           text,
        created_by                  varchar (60),
        created_date                timestamp without time zone,
        created_in_instance                 varchar (12),
        modified_by                 varchar (60),
        modified_date               timestamp without time zone,
        modified_in_instance                varchar (12),
        cycle           integer,
        subcycle                integer
);
```

```
-- create indexes ...
ALTER TABLE survey ADD CONSTRAINT survey_pkey PRIMARY KEY (cn);
--ALTER TABLE survey ADD CONSTRAINT survey_ukey UNIQUE (statecd, invyr,
p3_ozone_ind, cycle);

DROP TABLE IF EXISTS tree CASCADE;
CREATE TABLE tree
 (
	cn		varchar (68),
	plt_cn		varchar (68),
	prev_tre_cn	varchar (68),
	invyr		integer,
	statecd		integer,
	unitcd		integer,
	countycd	integer,
	plot		integer,
	subp		integer,
	tree		integer,
	condid		integer,
	azimuth		integer,
	dist		real,
	prevcond	integer,
	statuscd	integer,
	spcd		integer,
	spgrpcd		integer,
	dia		real,
	diahtcd		integer,
	ht		integer,
	htcd		integer,
	actualht	integer,
	treeclcd	integer,
	cr		integer,
	cclcd		integer,
	treegrcd	integer,
	agentcd		integer,
	cull		integer,
	damloc1		integer,
	damtyp1		integer,
	damsev1		integer,
	damloc2		integer,
	damtyp2		integer,
	damsev2		integer,
	decaycd		integer,
```

stocking	real,
wdldstem	integer,
volcfnet	double precision,
volcfgrs	double precision,
volcsnet	double precision,
volcsgrs	double precision,
volbfnet	double precision,
volbfgrs	double precision,
volcfsnd	double precision,
growcfgs	double precision,
growbfsl	double precision,
growcfal	double precision,
mortcfgs	double precision,
mortbfsl	double precision,
mortcfal	double precision,
remvcfgs	double precision,
remvbfsl	double precision,
remvcfal	double precision,
diacheck	integer,
mortyr	integer,
salvcd	integer,
uncrcd	integer,
cposcd	integer,
clightcd	integer,
cvigorcd	integer,
cdencd	integer,
cdiebkcd	integer,
transcd	integer,
treehistcd	integer,
diacalc	real,
bhage	integer,
totage	integer,
culldead	integer,
cullform	integer,
cullmstop	integer,
cullbf	integer,
cullcf	integer,
bfsnd	integer,
cfsnd	integer,
sawht	integer,
boleht	integer,
formcl	integer,
htcalc	integer,
hrdwd_clump_cd	integer,

```
sitree                  integer,
created_by              varchar (60),
created_date            timestamp without time zone,
created_in_instance     varchar (12),
modified_by             varchar (60),
modified_date           timestamp without time zone,
modified_in_instance    varchar (12),
mortcd                  integer,
htdmp                   real,
roughcull               integer,
mist_cl_cd              integer,
cull_fld                integer,
reconcilecd             integer,
prevdia                 real,
fgrowcfgs               double precision,
fgrowbfsl               double precision,
fgrowcfal               double precision,
fmortcfgs               double precision,
fmortbfsl               double precision,
fmortcfal               double precision,
fremvcfgs               double precision,
fremvbfsl               double precision,
fremvcfal               double precision,
p2a_grm_flg             varchar (2),
treeclcd_ners           integer,
treeclcd_srs            integer,
treeclcd_ncrs           integer,
treeclcd_rmrs           integer,
standing_dead_cd        integer,
prev_status_cd          integer,
prev_wdldstem           integer,
tpa_unadj               double precision,
tpamort_unadj           double precision,
tparemv_unadj           double precision,
tpagrow_unadj           double precision,
drybio_bole             double precision,
drybio_top              double precision,
drybio_stump            double precision,
drybio_sapling          double precision,
drybio_wdld_spp         double precision,
drybio_bg               double precision,
carbon_ag               double precision,
carbon_bg               double precision,
cycle                   integer,
```

```
    subcycle                integer,
    bored_cd_pnwrs                  integer,
    damloc1_pnwrs           integer,
    damloc2_pnwrs           integer,
    diacheck_pnwrs                  integer,
    dmg_agent1_cd_pnwrs             integer,
    dmg_agent2_cd_pnwrs             integer,
    dmg_agent3_cd_pnwrs             integer,
    mist_cl_cd_pnwrs                integer,
    severity1_cd_pnwrs              integer,
    severity1a_cd_pnwrs             integer,
    severity1b_cd_pnwrs             integer,
    severity2_cd_pnwrs              integer,
    severity2a_cd_pnwrs             integer,
    severity2b_cd_pnwrs             integer,
    severity3_cd_pnwrs              integer,
    unknown_damtyp1_pnwrs           integer,
    unknown_damtyp2_pnwrs           integer,
    prev_pntn_srs           integer,
    disease_srs             integer,
    dieback_severity_srs            integer,
    damage_agent_cd1                integer,
    damage_agent_cd2                integer,
    damage_agent_cd3                integer,
    centroid_dia double precision,
    centroid_dia_ht double precision,
    centroid_dia_ht_actual double precision,
    upper_dia double precision,
    upper_dia_ht double precision
);

-- create indexes ...
ALTER TABLE tree ADD CONSTRAINT tree_pkey PRIMARY KEY (cn);
ALTER TABLE tree ADD CONSTRAINT tree_ukey UNIQUE (plt_cn, subp, tree);
-- keep tree_cond_idx:
CREATE INDEX tree_cond_idx ON tree (plt_cn, condid);
CREATE INDEX tree_plt_cn_idx ON tree (plt_cn);

DROP TABLE IF EXISTS tree_grm_estn;
CREATE TABLE tree_grm_estn
(
  cn                    varchar(34),
  statecd                       integer,
  invyr   integer,
```

```
plt_cn                        varchar(34),
tre_cn                        varchar(34),
land_basis varchar(10),
estimate varchar(20),
estn_type varchar(10),
estn_units varchar(3),
component varchar(15),
subtyp_grm integer,
remper numeric(3,1),
tpagrow_unadj numeric(11,6),
tparemv_unadj numeric(11,6),
tpamort_unadj numeric(11,6),
ann_net_growth numeric(11,6),
removals NUMERIC(13,6),
mortality NUMERIC(13,6),
est_begin NUMERIC(13,6),
est_begin_recalc VARCHAR(1),
est_end NUMERIC(13,6),
est_midpt NUMERIC(13,6),
est_threshold NUMERIC(13,6),
dia_begin NUMERIC(5,2),
dia_begin_recalc VARCHAR(1),
dia_end NUMERIC(5,2),
dia_midpt NUMERIC(5,2),
dia_threshold NUMERIC(5,2),
g_s NUMERIC(13,6),
i NUMERIC(13,6),
g_i NUMERIC(13,6),
m NUMERIC(13,6),
g_m NUMERIC(13,6),
c NUMERIC(13,6),
g_c NUMERIC(13,6),
r NUMERIC(13,6),
g_r NUMERIC(13,6),
d NUMERIC(13,6),
g_d NUMERIC(13,6),
cd NUMERIC(13,6),
g_cd NUMERIC(13,6),
ci NUMERIC(13,6),
g_ci NUMERIC(13,6),
created_by varchar(30),
created_date timestamp without time zone,
created_in_instance varchar(6),
modified_by varchar(30),
```

```
    modified_date timestamp without time zone,
    modified_in_instance varchar(6)
);

-- create indexes ...
ALTER TABLE tree_grm_estn ADD CONSTRAINT tree_grm_estn_pkey PRIMARY KEY
(cn);
--ALTER TABLE tree_grm_estn ADD CONSTRAINT tree_grm_estn_ukey UNIQUE (tre_cn,
land_basis, estimate, estn_type, estn_units);

DROP TABLE IF EXISTS tree_regional_biomass;
CREATE TABLE tree_regional_biomass
 (
        tre_cn                  varchar (68),
        statecd                 integer,
        regional_drybiot                    double precision,
        regional_drybiom                    double precision,
        created_by              varchar (60),
        created_date            timestamp without time zone,
        created_in_instance                 varchar (12),
        modified_by             varchar (60),
        modified_date           timestamp without time zone,
        modified_in_instance                varchar (12)
);

-- create indexes ...
ALTER TABLE tree_regional_biomass ADD CONSTRAINT tree_regional_biomass_pkey
PRIMARY KEY (tre_cn);

DROP TABLE IF EXISTS veg_plot_species CASCADE;
CREATE TABLE veg_plot_species
 (
        cn              varchar (68),
        plt_cn                  varchar (68),
        vvt_cn                  varchar (68),
        invyr           integer,
        statecd                 integer,
        countycd                integer,
        plot            integer,
        veg_fldspcd             varchar (32),
        unique_sp_nbr               integer,
        veg_spcd                varchar (32),
        specimen_collected                  varchar (2),
        specimen_label_nbr                  double precision,
```

```sql
	specimen_not_collected_reason		integer,
	specimen_resolved		varchar (2),
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS veg_quadrat CASCADE;
CREATE TABLE veg_quadrat
 (
	cn		varchar (68),
	plt_cn		varchar (68),
	vvt_cn		varchar (68),
	vsb_cn		varchar (68),
	invyr		integer,
	statecd		integer,
	countycd		integer,
	plot		integer,
	subp		integer,
	quadrat		integer,
	condid		integer,
	quadrat_status		integer,
	quadrat_status_pre2004		integer,
	trampling		integer,
	created_by		varchar (60),
	created_date		timestamp without time zone,
	created_in_instance		varchar (12),
	modified_by		varchar (60),
	modified_date		timestamp without time zone,
	modified_in_instance		varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS veg_subplot CASCADE;
CREATE TABLE veg_subplot
 (
	cn		varchar (68),
	plt_cn		varchar (68),
```

```
    vvt_cn                  varchar (68),
    invyr               integer,
    statecd                 integer,
    countycd                 integer,
    plot                integer,
    subp                integer,
    veg_subp_status_cd                  integer,
    veg_subp_nonsample_reasn_cd                 integer,
    subp_accessible_forest_pct                  integer,
    detailed_nonforest_land_use                 integer,
    total_canopy_cover_layer_1                  integer,
    total_canopy_cover_layer_2                  integer,
    total_canopy_cover_layer_3                  integer,
    total_canopy_cover_layer_4                  integer,
    crypto_crust_cover_pct                  integer,
    lichen_cover_pct                integer,
    litter_duff_cover_pct                integer,
    mineral_soil_cover_pct                  integer,
    moss_cover_pct                  integer,
    road_trail_cover_pct                integer,
    rock_cover_pct                  integer,
    standing_water_cover_pct                    integer,
    stream_lake_cover_pct               integer,
    trash_junk_cover_pct                integer,
    wood_cover_pct                  integer,
    veg_subp_status_cd_pre2004                  integer,
    veg_subp_nonsmp_rsn_cd_pre2004                      integer,
    created_by                  varchar (60),
    created_date                timestamp without time zone,
    created_in_instance                 varchar (12),
    modified_by                 varchar (60),
    modified_date               timestamp without time zone,
    modified_in_instance                varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS veg_subplot_spp CASCADE;
CREATE TABLE veg_subplot_spp
 (
    cn                  varchar (68),
    plt_cn                  varchar (68),
    vvt_cn                  varchar (68),
    vsb_cn                  varchar (68),
```

```
	vps_cn			varchar (68),
	invyr		integer,
	statecd			integer,
	countycd			integer,
	plot		integer,
	subp		integer,
	veg_fldspcd			varchar (32),
	unique_sp_nbr			integer,
	veg_spcd			varchar (32),
	sp_canopy_cover_total			real,
	sp_canopy_cover_layer_1_2			real,
	sp_canopy_cover_layer_3			real,
	sp_canopy_cover_layer_4			real,
	quad_1_presence			integer,
	quad_2_presence			integer,
	quad_3_presence			integer,
	dummy_subp_cover_pre2004			integer,
	max_cover_layer_nbr_pre2004			integer,
	created_by			varchar (60),
	created_date			timestamp without time zone,
	created_in_instance			varchar (12),
	modified_by			varchar (60),
	modified_date			timestamp without time zone,
	modified_in_instance			varchar (12)
);

-- create indexes ...

DROP TABLE IF EXISTS veg_visit CASCADE;
CREATE TABLE veg_visit
 (
	cn		varchar (68),
	plt_cn			varchar (68),
	invyr		integer,
	statecd			integer,
	countycd			integer,
	plot		integer,
	veg_qa_status			integer,
	veg_kindcd			integer,
	veg_manual			varchar (16),
	trace_cover_allowed			integer,
	veg_measyear			integer,
	veg_measmon			integer,
	veg_measday			integer,
```

```
        veg_sample_basis                    integer,
        created_by                  varchar (60),
        created_date                 timestamp without time zone,
        created_in_instance                 varchar (12),
        modified_by                  varchar (60),
        modified_date                 timestamp without time zone,
        modified_in_instance                 varchar (12)
);

-- create indexes ...


-- foreign keys do not create indexes, so there is no real benefit to having them here
-- some unique constraints and natural keys are also not useful even though they create indexes

-- delete relationships ...
-- ALTER TABLE cond DROP CONSTRAINT cond_plt_cn_fk;
-- ALTER TABLE tree DROP CONSTRAINT tree_plt_cn_fk;
-- ALTER TABLE pop_stratum DROP CONSTRAINT pop_stratum_estn_unit_cn_fk;
-- ALTER TABLE pop_estn_unit DROP CONSTRAINT pop_estn_unit_eval_cn_fk;
-- ALTER TABLE pop_plot_stratum_assgn DROP CONSTRAINT
pop_plot_stratum_assgn_stratum_cn_fk;

-- create relationships ...
-- relationship from tree (plt_cn) to cond(plt_cn) does not enforce integrity.
-- ALTER TABLE cond ADD CONSTRAINT cond_plt_cn_fk FOREIGN KEY (plt_cn)
REFERENCES plot(cn);
-- relationship from pop_plot_stratum_assgn (plt_cn) to plot(cn) does not enforce integrity.
-- ALTER TABLE tree ADD CONSTRAINT tree_plt_cn_fk FOREIGN KEY (plt_cn)
REFERENCES plot(cn);
-- ALTER TABLE pop_stratum ADD CONSTRAINT pop_stratum_estn_unit_cn_fk FOREIGN
KEY (estn_unit_cn) REFERENCES pop_estn_unit(cn);
-- ALTER TABLE pop_estn_unit ADD CONSTRAINT pop_estn_unit_eval_cn_fk FOREIGN
KEY (eval_cn) REFERENCES pop_eval(cn);
-- ALTER TABLE pop_plot_stratum_assgn ADD CONSTRAINT
pop_plot_stratum_assgn_stratum_cn_fk FOREIGN KEY (stratum_cn) REFERENCES
pop_stratum(cn);
```

Appendix 18 – SQL program to add spatial columns to FIAD data

Filename: fiad_geography.sql

```
CREATE OR REPLACE FUNCTION fiad.fiad_geography()
   RETURNS void AS
$BODY$
DECLARE
   rows_affected integer := 0;
BEGIN
   SET search_path TO fiad, public;
   RAISE NOTICE 'Total expected runtime: ~110 seconds.';

   ALTER TABLE fiad.plot
      DROP COLUMN IF EXISTS geom CASCADE,
      DROP COLUMN IF EXISTS region_cd CASCADE;

   -- NOTICE:  drop cascades to 3 other objects
   -- DETAIL:  drop cascades to view summary."vFIAD_landscape_correction"
   -- drop cascades to view summary."vFIAD_by_region2"
   -- drop cascades to view summary."vFIAD_by_region3"

   -- EPSG:2163 - US National Atlas Lambert Azimuthal Equal Area
   -- http://spatialreference.org/ref/epsg/2163/
   ALTER TABLE fiad.plot
      ADD COLUMN geom GEOMETRY(POINT,2163),
      ADD COLUMN region_cd integer;

   -- Convert plot lat/lon to geometry and transform from WGS84 to US National Atlas Lambert
Azimuthal Equal Area
   -- ~75 seconds
   UPDATE fiad.plot
      SET geom = ST_Transform(ST_GeomFromText('POINT(' || lon || ' ' || lat || ')',4326),2163);

   -- Index the geometry
   -- ~31 seconds
   CREATE INDEX plot_gix
      ON fiad.plot USING GIST ( geom );

   -- Vaccum to ensure that the spatial index will be used.
   -- ~48 seconds
   --VACUUM FULL ANALYZE fiad.plot;

   -- set plot region_cd to region_cd from mrla polygons
```

```sql
    -- full update using PostGIS only
    -- ~10 minutes
    UPDATE
        fiad.plot AS p
    SET
        region_cd = m.region_cd
    FROM
        public.mlra_v42 AS m
    WHERE
        ST_Covers(m.geom, p.geom);


    -- catch points outside of mlra polygons, e.g. near coastlines, islands
    -- only null lat/lon remain
    -- ~74 seconds
    UPDATE
        fiad.plot AS p
    SET
        region_cd = n.region_cd
    FROM
        (
        SELECT DISTINCT ON
            (p.cn)
            p.cn,
            m.region_cd
        FROM
            fiad.plot AS p,
            public.mlra_v42 AS m
        WHERE
            p.region_cd IS NULL
            AND ST_DWithin(m.geom, p.geom, 80000)
        ORDER BY
            p.cn,
            ST_Distance(m.geom, p.geom)
        ) AS n
    WHERE
        p.cn = n.cn;

END;
$BODY$
    LANGUAGE plpgsql VOLATILE
    COST 100;
```

Appendix 19 – SQL program to create summaries of FIAD data

Filename: fiad_summaries.sql

```
CREATE OR REPLACE FUNCTION fiad.fiad_summaries()
   RETURNS void AS
$BODY$
DECLARE
   rows_affected integer := 0;
BEGIN
   SET search_path TO fiad;
   RAISE NOTICE 'Total expected runtime: ~523 seconds.';

   -- add formatted taxon to ref_species
   DROP VIEW IF EXISTS v_ref_species CASCADE;
   CREATE VIEW v_ref_species AS
   SELECT
      *,
      CONCAT_WS(' ',
         genus,
         species
      ) AS taxon
   FROM
      ref_species;

   -- most recent evaluations by evaluation type
   DROP VIEW IF EXISTS v_latest_eval;
   CREATE VIEW v_latest_eval AS
   SELECT
      pet.eval_typ,
      peg.statecd,
      MAX(peg.eval_grp) AS eval_grp
   FROM
      pop_eval pev,
      pop_eval_grp peg,
      pop_eval_typ pet
   WHERE
      MOD(peg.eval_grp, 10000) <= EXTRACT('year' FROM CURRENT_DATE) -- filter out
invyr like 9999
      AND pev.timberland_only <> 'Y'
      AND pev.cn = pet.eval_cn
      AND pet.eval_grp_cn = peg.cn
   GROUP BY
      pet.eval_typ,
```

```
    peg.statecd
ORDER by
    pet.eval_typ,
    peg.statecd;

DROP TABLE IF EXISTS trees_by_region CASCADE;
CREATE TABLE trees_by_region
(
region_cd integer,
taxon VARCHAR,
genus VARCHAR,
species VARCHAR,
common_name VARCHAR,
wetland_cd VARCHAR(3),
trees NUMERIC
);

-- count trees by region, wetland code
-- ~340 seconds
INSERT INTO trees_by_region
SELECT
    p.region_cd,
    spp.taxon,
    spp.genus,
    spp.species,
    '' AS com_name, --need a unique name here, don't use common_name
    CASE
        WHEN c.physclcd IS NULL THEN
            'N/A'
        WHEN c.physclcd IN (31, 32, 33, 34, 35, 39) THEN --hydric codes
            'w'
        ELSE
            'u'
    END AS wetland_cd,
    ROUND(
        SUM(
            COALESCE(
                t.tpa_unadj * --Trees per acre unadjusted
                CASE t.dia
                    WHEN NULL THEN
                        psm.adj_factor_subp --Adjusts the SUBplot population estimates to account for
partially nonsampled plots.
                    ELSE
                        CASE least(t.dia, 5-0.001)
```

```
                WHEN t.dia THEN
                  psm.adj_factor_micr --Adjusts the MICROplot population estimates to
account for partially nonsampled plots.
                ELSE
                  CASE least(t.dia, COALESCE(p.macro_breakpoint_dia,9999) - 0.001)
--diameter above which trees are measured on the (macro)plot
                    WHEN t.dia THEN
                      psm.adj_factor_subp
                    ELSE
                      psm.adj_factor_macr --Adjusts the MACROplot population estimates
to account for partially nonsampled plots.
                  END
              END
          END,
          0
        ) *
        psm.expns
      )
    ) AS trees
  FROM
    cond c,
    tree t,
    plot p,
    pop_stratum psm,
    pop_plot_stratum_assgn ppsa,
    pop_estn_unit peu,
    pop_eval pev,
    pop_eval_grp peg,
    pop_eval_typ pet,
    v_latest_eval lev,
    v_ref_species AS spp
  WHERE
    c.plt_cn = p.cn
    AND pet.eval_typ = 'EXPVOL'
    and t.plt_cn = c.plt_cn
    and t.condid = c.condid
    and t.statuscd = 1 --live trees
    and t.dia >= 1.0
    and c.cond_status_cd = 1 --Accessible forest land
    AND ppsa.plt_cn = p.cn --Stratum information is assigned to a plot by overlaying the plot's
location on the Phase 1 imagery
    AND ppsa.stratum_cn = psm.cn
    and peu.cn = psm.estn_unit_cn
    AND pet.eval_grp_cn = peg.cn
```

```
    and pev.cn = pet.eval_cn
    and pev.cn = peu.eval_cn
    AND peg.eval_grp = lev.eval_grp
    AND peg.statecd = lev.statecd
    AND pet.eval_typ = lev.eval_typ
    AND t.spcd = spp.spcd
  GROUP BY
    p.region_cd,
    spp.taxon,
    spp.genus,
    spp.species,
    com_name,
    wetland_cd
  ORDER BY
    p.region_cd,
    spp.taxon,
    wetland_cd;

  -- get common name from species with no variety and no subspecies
  UPDATE trees_by_region t
  SET
    common_name = spp.common_name
  FROM
    v_ref_species AS spp
  WHERE
    t.genus = spp.genus
    AND t.species = spp.species
    AND spp.subspecies = ''
    AND spp.variety = '';

  -- add pkey AFTER insert to speed up insert
  ALTER TABLE trees_by_region ADD CONSTRAINT trees_by_region_pkey PRIMARY
KEY (region_cd, taxon, wetland_cd);

  --VACUUM FULL ANALYZE trees_by_region;


  DROP TABLE IF EXISTS trees_arid_west_minus_ca CASCADE;
  CREATE TABLE trees_arid_west_minus_ca
  (
  region_cd integer,
  taxon VARCHAR,
  genus VARCHAR,
  species VARCHAR,
```

```
common_name VARCHAR,
wetland_cd VARCHAR(3),
trees NUMERIC
);

-- count trees by Arid West region minus CA, wetland code
-- ~6 seconds, 90 rows
INSERT INTO trees_arid_west_minus_ca
SELECT
    p.region_cd,
    spp.taxon,
    spp.genus,
    spp.species,
    '' AS com_name, --need a unique name here, don't use common_name
    CASE
        WHEN c.physclcd IS NULL THEN
            'N/A'
        WHEN c.physclcd IN (31, 32, 33, 34, 35, 39) THEN --hydric codes
            'w'
        ELSE
            'u'
    END AS wetland_cd,
    ROUND(
        SUM(
            COALESCE(
                t.tpa_unadj * --Trees per acre unadjusted
                CASE t.dia
                    WHEN NULL THEN
                        psm.adj_factor_subp --Adjusts the SUBplot population estimates to account for
partially nonsampled plots.
                    ELSE
                        CASE least(t.dia, 5-0.001)
                            WHEN t.dia THEN
                                psm.adj_factor_micr --Adjusts the MICROplot population estimates to
account for partially nonsampled plots.
                            ELSE
                                CASE least(t.dia, COALESCE(p.macro_breakpoint_dia,9999) - 0.001)
--diameter above which trees are measured on the (macro)plot
                                    WHEN t.dia THEN
                                        psm.adj_factor_subp
                                    ELSE
                                        psm.adj_factor_macr --Adjusts the MACROplot population estimates
to account for partially nonsampled plots.
                                END
```

```
                END
            END,
            0
          ) *
          psm.expns
        )
      ) AS trees
    FROM
      cond c,
      tree t,
      plot p,
      pop_stratum psm,
      pop_plot_stratum_assgn ppsa,
      pop_estn_unit peu,
      pop_eval pev,
      pop_eval_grp peg,
      pop_eval_typ pet,
      v_latest_eval lev,
      v_ref_species AS spp
    WHERE
      c.plt_cn = p.cn
      AND pet.eval_typ = 'EXPVOL'
      and t.plt_cn = c.plt_cn
      and t.condid = c.condid
      and t.statuscd = 1 --live trees
      and t.dia >= 1.0
      and c.cond_status_cd = 1 --Accessible forest land
      AND ppsa.plt_cn = p.cn --Stratum information is assigned to a plot by overlaying the plot's
location on the Phase 1 imagery
      AND ppsa.stratum_cn = psm.cn
      and peu.cn = psm.estn_unit_cn
      AND pet.eval_grp_cn = peg.cn
      and pev.cn = pet.eval_cn
      and pev.cn = peu.eval_cn
      AND peg.eval_grp = lev.eval_grp
      AND peg.statecd = lev.statecd
      AND pet.eval_typ = lev.eval_typ
      AND p.region_cd = 5 --'aw'
      AND p.statecd <> 6
      AND t.spcd = spp.spcd
    GROUP BY
      p.region_cd,
      spp.taxon,
      spp.genus,
```

```
    spp.species,
    com_name,
    wetland_cd
  ORDER BY
    p.region_cd,
    spp.taxon,
    wetland_cd;

  -- get common name from species with no variety and no subspecies
  UPDATE trees_arid_west_minus_ca t
  SET
    common_name = spp.common_name
  FROM
    v_ref_species AS spp
  WHERE
    t.genus = spp.genus
    AND t.species = spp.species
    AND spp.subspecies = ''
    AND spp.variety = '';

  -- add pkey AFTER insert to speed up insert
  ALTER TABLE trees_arid_west_minus_ca ADD CONSTRAINT
trees_arid_west_minus_ca_pkey PRIMARY KEY (region_cd, taxon, wetland_cd);


  DROP TABLE IF EXISTS trees_by_state CASCADE;
  CREATE TABLE trees_by_state
  (
    statecd integer,
    taxon VARCHAR,
    genus VARCHAR,
    species VARCHAR,
    common_name VARCHAR,
    wetland_cd VARCHAR(3),
    trees NUMERIC
  );

  -- count trees by state, wetland code
  -- ~114 seconds
  INSERT INTO trees_by_state
  SELECT
    p.statecd,
    spp.taxon,
    spp.genus,
```

```
        spp.species,
        '' AS com_name,
        CASE
          WHEN c.physclcd IS NULL THEN
            'N/A'
          WHEN c.physclcd IN (31, 32, 33, 34, 35, 39) THEN --hydric codes
            'w'
          ELSE
            'u'
        END AS wetland_cd,
        ROUND(
          SUM(
            COALESCE(
              t.tpa_unadj * --Trees per acre unadjusted
              CASE t.dia
                WHEN NULL THEN
                  psm.adj_factor_subp --Adjusts the SUBplot population estimates to account for
partially nonsampled plots.
                ELSE
                  CASE least(t.dia, 5-0.001)
                    WHEN t.dia THEN
                      psm.adj_factor_micr --Adjusts the MICROplot population estimates to
account for partially nonsampled plots.
                    ELSE
                      CASE least(t.dia, COALESCE(p.macro_breakpoint_dia,9999) - 0.001)
--diameter above which trees are measured on the (macro)plot
                        WHEN t.dia THEN
                          psm.adj_factor_subp
                        ELSE
                          psm.adj_factor_macr --Adjusts the MACROplot population estimates
to account for partially nonsampled plots.
                      END
                  END
              END,
              0
            ) *
            psm.expns
          )
        ) AS trees
      FROM
        cond c,
        tree t,
        plot p,
        pop_stratum psm,
```

```
    pop_plot_stratum_assgn ppsa,
    pop_estn_unit peu,
    pop_eval pev,
    pop_eval_grp peg,
    pop_eval_typ pet,
    v_latest_eval lev,
    v_ref_species AS spp
  WHERE
    c.plt_cn = p.cn
    AND pet.eval_typ = 'EXPVOL'
    and t.plt_cn = c.plt_cn
    and t.condid = c.condid
    and t.statuscd = 1 --live trees
    and t.dia >= 1.0
    and c.cond_status_cd = 1 --Accessible forest land
    AND ppsa.plt_cn = p.cn --Stratum information is assigned to a plot by overlaying the plot's
location on the Phase 1 imagery
    AND ppsa.stratum_cn = psm.cn
    and peu.cn = psm.estn_unit_cn
    AND pet.eval_grp_cn = peg.cn
    and pev.cn = pet.eval_cn
    and pev.cn = peu.eval_cn
    AND peg.eval_grp = lev.eval_grp
    AND peg.statecd = lev.statecd
    AND pet.eval_typ = lev.eval_typ
    AND t.spcd = spp.spcd
  GROUP BY
    p.statecd,
    spp.taxon,
    spp.genus,
    spp.species,
    com_name,
    wetland_cd
  ORDER BY
    p.statecd,
    spp.taxon,
    wetland_cd;

  -- get common name from species with no variety and no subspecies
  UPDATE trees_by_state t
  SET
    common_name = spp.common_name
  FROM
    v_ref_species AS spp
```

```
WHERE
    t.genus = spp.genus
    AND t.species = spp.species
    AND spp.subspecies = ''
    AND spp.variety = '';

-- add pkey AFTER insert to speed up insert
ALTER TABLE trees_by_state ADD CONSTRAINT trees_by_state_pkey PRIMARY
KEY (statecd, taxon, wetland_cd);

--VACUUM FULL ANALYZE trees_by_state;


DROP TABLE IF EXISTS area_by_region CASCADE;
CREATE TABLE area_by_region
(
region_cd integer,
wetland_cd VARCHAR(3),
area NUMERIC
);

-- count area by region, wetland code
-- ~7 seconds
INSERT INTO area_by_region
SELECT
    p.region_cd,
    CASE
        WHEN c.physclcd IS NULL THEN
            'N/A'
        WHEN c.physclcd IN (31, 32, 33, 34, 35, 39) THEN --hydric codes
            'w'
        ELSE
            'u'
    END AS wetland_cd,
    SUM(
    psm.expns * c.condprop_unadj *
        CASE c.prop_basis
        WHEN 'MACR' THEN
            psm.adj_factor_macr
        ELSE
            psm.adj_factor_subp
        END
    ) AS area
FROM
```

```
    cond c,
    plot p,
    pop_plot_stratum_assgn ppsa,
    pop_stratum psm,
    pop_estn_unit peu,
    pop_eval pev,
    pop_eval_typ pet,
    pop_eval_grp peg,
    v_latest_eval lev
  WHERE
    p.cn = c.plt_cn
    AND pet.eval_typ = 'EXPCURR'
    AND c.cond_status_cd = 1
    AND ppsa.plt_cn = p.cn
    AND ppsa.stratum_cn = psm.cn
    AND peu.cn = psm.estn_unit_cn
    AND pev.cn = peu.eval_cn
    AND pev.cn = pet.eval_cn
    AND pet.eval_grp_cn = peg.cn
    AND peg.eval_grp = lev.eval_grp
    AND peg.statecd = lev.statecd
    AND pet.eval_typ = lev.eval_typ
  GROUP BY
    p.region_cd,
    wetland_cd
  ORDER BY
    p.region_cd,
    wetland_cd;

  -- add pkey AFTER insert to speed up insert
  ALTER TABLE area_by_region ADD CONSTRAINT area_by_region_pkey PRIMARY
KEY (region_cd, wetland_cd);

  --VACUUM FULL ANALYZE area_by_region;


  DROP TABLE IF EXISTS area_arid_west_minus_ca CASCADE;
  CREATE TABLE area_arid_west_minus_ca
  (
  region_cd integer,
  wetland_cd VARCHAR(3),
  area NUMERIC
  );
```

```sql
-- count area by Arid West region minus CA, wetland code
-- ~5 seconds
INSERT INTO area_arid_west_minus_ca
SELECT
   p.region_cd,
   CASE
      WHEN c.physclcd IS NULL THEN
         'N/A'
      WHEN c.physclcd IN (31, 32, 33, 34, 35, 39) THEN --hydric codes
         'w'
      ELSE
         'u'
   END AS wetland_cd,
   SUM(
   psm.expns * c.condprop_unadj *
      CASE c.prop_basis
      WHEN 'MACR' THEN
         psm.adj_factor_macr
      ELSE
         psm.adj_factor_subp
      END
   ) AS area
FROM
   cond c,
   plot p,
   pop_plot_stratum_assgn ppsa,
   pop_stratum psm,
   pop_estn_unit peu,
   pop_eval pev,
   pop_eval_typ pet,
   pop_eval_grp peg,
   v_latest_eval lev
WHERE
   p.cn = c.plt_cn
   AND pet.eval_typ = 'EXPCURR'
   AND c.cond_status_cd = 1
   AND ppsa.plt_cn = p.cn
   AND ppsa.stratum_cn = psm.cn
   AND peu.cn = psm.estn_unit_cn
   AND pev.cn = peu.eval_cn
   AND pev.cn = pet.eval_cn
   AND pet.eval_grp_cn = peg.cn
   AND peg.eval_grp = lev.eval_grp
   AND peg.statecd = lev.statecd
```

```
    AND pet.eval_typ = lev.eval_typ
    AND p.region_cd = 5 --'aw'
    AND p.statecd <> 6
  GROUP BY
    p.region_cd,
    wetland_cd
  ORDER BY
    p.region_cd,
    wetland_cd;

  -- add pkey AFTER insert to speed up insert
  ALTER TABLE area_arid_west_minus_ca ADD CONSTRAINT
area_arid_west_minus_ca_pkey PRIMARY KEY (region_cd, wetland_cd);


  DROP TABLE IF EXISTS area_by_state CASCADE;
  CREATE TABLE area_by_state
  (
  statecd integer,
  wetland_cd VARCHAR(3),
  area NUMERIC
  );

  -- count area by state, wetland code
  -- ~7 seconds
  INSERT INTO area_by_state
  SELECT
    p.statecd,
    CASE
      WHEN c.physclcd IS NULL THEN
        'N/A'
      WHEN c.physclcd IN (31, 32, 33, 34, 35, 39) THEN --hydric codes
        'w'
      ELSE
        'u'
    END AS wetland_cd,
    SUM(
    psm.expns * c.condprop_unadj *
      CASE c.prop_basis
      WHEN 'MACR' THEN
        psm.adj_factor_macr
      ELSE
        psm.adj_factor_subp
      END
```

```
    ) AS area
  FROM
    cond c,
    plot p,
    pop_plot_stratum_assgn ppsa,
    pop_stratum psm,
    pop_estn_unit peu,
    pop_eval pev,
    pop_eval_typ pet,
    pop_eval_grp peg,
    v_latest_eval lev
  WHERE
    p.cn = c.plt_cn
    AND pet.eval_typ = 'EXPCURR'
    AND c.cond_status_cd = 1
    AND ppsa.plt_cn = p.cn
    AND ppsa.stratum_cn = psm.cn
    AND peu.cn = psm.estn_unit_cn
    AND pev.cn = peu.eval_cn
    AND pev.cn = pet.eval_cn
    AND pet.eval_grp_cn = peg.cn
    AND peg.eval_grp = lev.eval_grp
    AND peg.statecd = lev.statecd
    AND pet.eval_typ = lev.eval_typ
  GROUP BY
    p.statecd,
    wetland_cd
  ORDER BY
    p.statecd,
    wetland_cd;

  -- add pkey AFTER insert to speed up insert
  ALTER TABLE area_by_state ADD CONSTRAINT area_by_state_pkey PRIMARY KEY
(statecd, wetland_cd);

  --VACUUM FULL ANALYZE area_by_state;

  -- views to simplify analysis queries and improve efficiency

  DROP VIEW IF EXISTS v_trees_by_region_flat CASCADE;
  CREATE VIEW v_trees_by_region_flat AS
  SELECT
    taxon,
    genus,
```

```
    species,
    common_name,
    region_cd,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            trees
        ELSE
            0
        END
    ) AS npw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            trees
        ELSE
            0
        END
    ) AS npu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            trees
        ELSE
            0
        END
    ) AS npother
FROM
    trees_by_region
GROUP BY
    taxon,
    genus,
    species,
    common_name,
    region_cd;


DROP VIEW IF EXISTS v_trees_arid_west_minus_ca_flat CASCADE;
CREATE VIEW v_trees_arid_west_minus_ca_flat AS
SELECT
    taxon,
    genus,
    species,
    common_name,
```

```
    region_cd,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            trees
        ELSE
            0
        END
    ) AS npw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            trees
        ELSE
            0
        END
    ) AS npu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            trees
        ELSE
            0
        END
    ) AS npother
FROM
    trees_arid_west_minus_ca
GROUP BY
    taxon,
    genus,
    species,
    common_name,
    region_cd;


DROP VIEW IF EXISTS v_trees_by_state_flat CASCADE;
CREATE VIEW v_trees_by_state_flat AS
SELECT
    taxon,
    genus,
    species,
    common_name,
    statecd,
    SUM(
```

```
      CASE
      WHEN wetland_cd = 'w' THEN
        trees
      ELSE
        0
      END
    ) AS npw,
    SUM(
      CASE
      WHEN wetland_cd = 'u' THEN
        trees
      ELSE
        0
      END
    ) AS npu,
    SUM(
      CASE
      WHEN wetland_cd NOT IN ('w', 'u') THEN
        trees
      ELSE
        0
      END
    ) AS npother
FROM
    trees_by_state
GROUP BY
    taxon,
    genus,
    species,
    common_name,
    statecd;


DROP VIEW IF EXISTS v_area_by_region_flat CASCADE;
CREATE VIEW v_area_by_region_flat AS
SELECT
    region_cd,
    SUM(
      CASE
      WHEN wetland_cd = 'w' THEN
        area
      ELSE
        0
      END
```

```
    ) AS ndotw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            area
        ELSE
            0
        END
    ) AS ndotu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            area
        ELSE
            0
        END
    ) AS ndotother
FROM
    area_by_region
GROUP BY
    region_cd;


DROP VIEW IF EXISTS v_area_arid_west_minus_ca_flat CASCADE;
CREATE VIEW v_area_arid_west_minus_ca_flat AS
SELECT
    region_cd,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            area
        ELSE
            0
        END
    ) AS ndotw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            area
        ELSE
            0
        END
    ) AS ndotu,
    SUM(
```

```
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            area
        ELSE
            0
        END
    ) AS ndotother
FROM
    area_arid_west_minus_ca
GROUP BY
    region_cd;


DROP VIEW IF EXISTS v_area_by_state_flat CASCADE;
CREATE VIEW v_area_by_state_flat AS
SELECT
    statecd,
    SUM(
        CASE
        WHEN wetland_cd = 'w' THEN
            area
        ELSE
            0
        END
    ) AS ndotw,
    SUM(
        CASE
        WHEN wetland_cd = 'u' THEN
            area
        ELSE
            0
        END
    ) AS ndotu,
    SUM(
        CASE
        WHEN wetland_cd NOT IN ('w', 'u') THEN
            area
        ELSE
            0
        END
    ) AS ndotother
FROM
    area_by_state
GROUP BY
```

```sql
    statecd;

-----------------------------------------------------------
DROP VIEW IF EXISTS v_trees_freq_region CASCADE;
CREATE VIEW v_trees_freq_region AS
SELECT
    t.*,
    a.ndotw,
    a.ndotu,
    a.ndotother,
    CASE
    WHEN t.npw + t.npu = 0 THEN
        NULL
    ELSE
        t.npw / (t.npw + t.npu)
    END AS wetland_freq_unadj,
    CASE
    WHEN t.npw + t.npu = 0 THEN
        NULL
    WHEN t.npw = 0 THEN
        0
    ELSE
        1 / (1 + (t.npu * a.ndotw / (t.npw * a.ndotu)))
    END AS wetland_freq_adj
FROM
    v_trees_by_region_flat AS t,
    v_area_by_region_flat AS a
WHERE
    t.region_cd = a.region_cd;


DROP VIEW IF EXISTS v_trees_freq_arid_west_minus_ca CASCADE;
CREATE VIEW v_trees_freq_arid_west_minus_ca AS
SELECT
    t.*,
    a.ndotw,
    a.ndotu,
    a.ndotother,
    CASE
    WHEN t.npw + t.npu = 0 THEN
        NULL
    ELSE
        t.npw / (t.npw + t.npu)
    END AS wetland_freq_unadj,
```

```sql
    CASE
    WHEN t.npw + t.npu = 0 THEN
        NULL
    WHEN t.npw = 0 THEN
        0
    ELSE
        1 / (1 + (t.npu * a.ndotw / (t.npw * a.ndotu)))
    END AS wetland_freq_adj
FROM
    v_trees_arid_west_minus_ca_flat AS t,
    v_area_arid_west_minus_ca_flat AS a
WHERE
    t.region_cd = a.region_cd;


------------------------------------------------------------
DROP VIEW IF EXISTS v_trees_freq_state CASCADE;
CREATE VIEW v_trees_freq_state AS
SELECT
    t.*,
    a.ndotw,
    a.ndotu,
    a.ndotother,
    CASE
    WHEN t.npw + t.npu = 0 THEN
        NULL
    ELSE
        t.npw / (t.npw + t.npu)
    END AS wetland_freq_unadj,
    CASE
    WHEN t.npw + t.npu = 0 THEN
        NULL
    WHEN t.npw = 0 THEN
        0
    ELSE
        1 / (1 + (t.npu * a.ndotw / (t.npw * a.ndotu)))
    END AS wetland_freq_adj
FROM
    v_trees_by_state_flat AS t,
    v_area_by_state_flat AS a
WHERE
    t.statecd = a.statecd;
```

```
------------------------------------------------------------
DROP VIEW IF EXISTS v_trees_ind_status_region CASCADE;
CREATE VIEW v_trees_ind_status_region AS
SELECT
   t.*,
   r.region_abbr,
   r.region_name,
   CASE
      WHEN wetland_freq_unadj >= 0.99 THEN
         'OBL'
      WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
         'FACW'
      WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
         'FAC'
      WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
         'FACU'
      WHEN wetland_freq_unadj <= 0.01 THEN
         'UPL'
      ELSE
         '?'
   END AS wetland_ind_unadj,
   CASE
      WHEN wetland_freq_adj >= 0.99 THEN
         'OBL'
      WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
         'FACW'
      WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
         'FAC'
      WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
         'FACU'
      WHEN wetland_freq_adj <= 0.01 THEN
         'UPL'
      ELSE
         '?'
   END AS wetland_ind_adj,
   CASE
      WHEN wetland_freq_unadj >= 0.99 THEN
         5
      WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
         4
      WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
         3
      WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
         2
```

```
            WHEN wetland_freq_unadj <= 0.01 THEN
               1
            ELSE
               NULL
         END AS wetland_ind_unadj_rank,
         CASE
            WHEN wetland_freq_adj >= 0.99 THEN
               5
            WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
               4
            WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
               3
            WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
               2
            WHEN wetland_freq_adj <= 0.01 THEN
               1
            ELSE
               NULL
         END AS wetland_ind_adj_rank
      FROM
         v_trees_freq_region t,
         public.regions as r
      WHERE
         t.region_cd = r.region_cd
      ORDER BY
         taxon,
         t.region_cd;


      ---------------------------------------------------------
      DROP VIEW IF EXISTS v_trees_ind_status_region_variance CASCADE;
      CREATE VIEW v_trees_ind_status_region_variance AS
      SELECT
         taxon,
         MAX(wetland_ind_unadj_rank) - MIN(wetland_ind_unadj_rank) AS
   wetland_ind_unadj_rank_range,
         VAR_POP(wetland_ind_unadj_rank) AS wetland_ind_unadj_rank_variance,
         MAX(wetland_ind_adj_rank) - MIN(wetland_ind_adj_rank) AS
   wetland_ind_adj_rank_range,
         VAR_POP(wetland_ind_adj_rank) AS wetland_ind_adj_rank_variance
      FROM
         v_trees_ind_status_region t
      GROUP BY
         taxon;
```

```sql
------------------------------------------------------------
DROP VIEW IF EXISTS v_trees_ind_status_arid_west_minus_ca CASCADE;
CREATE VIEW v_trees_ind_status_arid_west_minus_ca AS
SELECT
   t.*,
   r.region_abbr,
   r.region_name,
   CASE
      WHEN wetland_freq_unadj >= 0.99 THEN
         'OBL'
      WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
         'FACW'
      WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
         'FAC'
      WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
         'FACU'
      WHEN wetland_freq_unadj <= 0.01 THEN
         'UPL'
      ELSE
         '?'
   END AS wetland_ind_unadj,
   CASE
      WHEN wetland_freq_adj >= 0.99 THEN
         'OBL'
      WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
         'FACW'
      WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
         'FAC'
      WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
         'FACU'
      WHEN wetland_freq_adj <= 0.01 THEN
         'UPL'
      ELSE
         '?'
   END AS wetland_ind_adj,
   CASE
      WHEN wetland_freq_unadj >= 0.99 THEN
         5
      WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
         4
      WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
         3
```

```
      WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
         2
      WHEN wetland_freq_unadj <= 0.01 THEN
         1
      ELSE
         NULL
   END AS wetland_ind_unadj_rank,
   CASE
      WHEN wetland_freq_adj >= 0.99 THEN
         5
      WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
         4
      WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
         3
      WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
         2
      WHEN wetland_freq_adj <= 0.01 THEN
         1
      ELSE
         NULL
   END AS wetland_ind_adj_rank
FROM
   v_trees_freq_arid_west_minus_ca t,
   public.regions as r
WHERE
   t.region_cd = r.region_cd
ORDER BY
   t.taxon,
   t.region_cd;


--------------------------------------------------------
DROP VIEW IF EXISTS v_trees_ind_status_state;
CREATE VIEW v_trees_ind_status_state AS
SELECT
   t.*,
   r.state_abbr,
   r.state_name,
   CASE
      WHEN wetland_freq_unadj >= 0.99 THEN
         'OBL'
      WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
         'FACW'
      WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
```

```
        'FAC'
    WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
        'FACU'
    WHEN wetland_freq_unadj <= 0.01 THEN
        'UPL'
    ELSE
        '?'
END AS wetland_ind_unadj,
CASE
    WHEN wetland_freq_adj >= 0.99 THEN
        'OBL'
    WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
        'FACW'
    WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
        'FAC'
    WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
        'FACU'
    WHEN wetland_freq_adj <= 0.01 THEN
        'UPL'
    ELSE
        '?'
END AS wetland_ind_adj,
CASE
    WHEN wetland_freq_unadj >= 0.99 THEN
        5
    WHEN wetland_freq_unadj >= 0.67 AND wetland_freq_unadj < 0.99 THEN
        4
    WHEN wetland_freq_unadj >= 0.33 AND wetland_freq_unadj < 0.67 THEN
        3
    WHEN wetland_freq_unadj > 0.01 AND wetland_freq_unadj < 0.33 THEN
        2
    WHEN wetland_freq_unadj <= 0.01 THEN
        1
    ELSE
        NULL
END AS wetland_ind_unadj_rank,
CASE
    WHEN wetland_freq_adj >= 0.99 THEN
        5
    WHEN wetland_freq_adj >= 0.67 AND wetland_freq_adj < 0.99 THEN
        4
    WHEN wetland_freq_adj >= 0.33 AND wetland_freq_adj < 0.67 THEN
        3
    WHEN wetland_freq_adj > 0.01 AND wetland_freq_adj < 0.33 THEN
```

```
      2
    WHEN wetland_freq_adj <= 0.01 THEN
      1
    ELSE
      NULL
  END AS wetland_ind_adj_rank
FROM
  v_trees_freq_state t,
  ref_research_station as r
WHERE
  t.statecd = r.statecd
ORDER BY
  t.taxon,
  t.statecd;


----------------------------------------------------------
DROP VIEW IF EXISTS v_trees_ind_status_region_flat CASCADE;
CREATE VIEW v_trees_ind_status_region_flat AS
SELECT DISTINCT ON (t1.taxon)
  t1.taxon,
  t1.genus,
  t1.species,
  t1.common_name,
  t_ncne.ndotw AS ncne_ndotw,
  t_ncne.ndotu AS ncne_ndotu,
  t_ncne.ndotother AS ncne_ndotother,
  t_ncne.npw AS ncne_npw,
  t_ncne.npu AS ncne_npu,
  t_ncne.npother AS ncne_npother,
  t_ncne.wetland_freq_unadj AS ncne_freq_unadj,
  t_ncne.wetland_freq_adj AS ncne_freq_adj,
  t_ncne.wetland_ind_unadj AS ncne_ind_unadj,
  t_ncne.wetland_ind_adj AS ncne_ind_adj,
  t_mw.ndotw AS mw_ndotw,
  t_mw.ndotu AS mw_ndotu,
  t_mw.ndotother AS mw_ndotother,
  t_mw.npw AS mw_npw,
  t_mw.npu AS mw_npu,
  t_mw.npother AS mw_npother,
  t_mw.wetland_freq_unadj AS mw_freq_unadj,
  t_mw.wetland_freq_adj AS mw_freq_adj,
  t_mw.wetland_ind_unadj AS mw_ind_unadj,
  t_mw.wetland_ind_adj AS mw_ind_adj,
```

t_emp.ndotw **AS** emp_ndotw,
t_emp.ndotu **AS** emp_ndotu,
t_emp.ndotother **AS** emp_ndotother,
t_emp.npw **AS** emp_npw,
t_emp.npu **AS** emp_npu,
t_emp.npother **AS** emp_npother,
t_emp.wetland_freq_unadj **AS** emp_freq_unadj,
t_emp.wetland_freq_adj **AS** emp_freq_adj,
t_emp.wetland_ind_unadj **AS** emp_ind_unadj,
t_emp.wetland_ind_adj **AS** emp_ind_adj,
t_gp.ndotw **AS** gp_ndotw,
t_gp.ndotu **AS** gp_ndotu,
t_gp.ndotother **AS** gp_ndotother,
t_gp.npw **AS** gp_npw,
t_gp.npu **AS** gp_npu,
t_gp.npother **AS** gp_npother,
t_gp.wetland_freq_unadj **AS** gp_freq_unadj,
t_gp.wetland_freq_adj **AS** gp_freq_adj,
t_gp.wetland_ind_unadj **AS** gp_ind_unadj,
t_gp.wetland_ind_adj **AS** gp_ind_adj,
t_aw.ndotw **AS** aw_ndotw,
t_aw.ndotu **AS** aw_ndotu,
t_aw.ndotother **AS** aw_ndotother,
t_aw.npw **AS** aw_npw,
t_aw.npu **AS** aw_npu,
t_aw.npother **AS** aw_npother,
t_aw.wetland_freq_unadj **AS** aw_freq_unadj,
t_aw.wetland_freq_adj **AS** aw_freq_adj,
t_aw.wetland_ind_unadj **AS** aw_ind_unadj,
t_aw.wetland_ind_adj **AS** aw_ind_adj,
t_agcp.ndotw **AS** agcp_ndotw,
t_agcp.ndotu **AS** agcp_ndotu,
t_agcp.ndotother **AS** agcp_ndotother,
t_agcp.npw **AS** agcp_npw,
t_agcp.npu **AS** agcp_npu,
t_agcp.npother **AS** agcp_npother,
t_agcp.wetland_freq_unadj **AS** agcp_freq_unadj,
t_agcp.wetland_freq_adj **AS** agcp_freq_adj,
t_agcp.wetland_ind_unadj **AS** agcp_ind_unadj,
t_agcp.wetland_ind_adj **AS** agcp_ind_adj,
t_wmvc.ndotw **AS** wmvc_ndotw,
t_wmvc.ndotu **AS** wmvc_ndotu,
t_wmvc.ndotother **AS** wmvc_ndotother,
t_wmvc.npw **AS** wmvc_npw,

```
    t_wmvc.npu AS wmvc_npu,
    t_wmvc.npother AS wmvc_npother,
    t_wmvc.wetland_freq_unadj AS wmvc_freq_unadj,
    t_wmvc.wetland_freq_adj AS wmvc_freq_adj,
    t_wmvc.wetland_ind_unadj AS wmvc_ind_unadj,
    t_wmvc.wetland_ind_adj AS wmvc_ind_adj,
    t_ak.ndotw AS ak_ndotw,
    t_ak.ndotu AS ak_ndotu,
    t_ak.ndotother AS ak_ndotother,
    t_ak.npw AS ak_npw,
    t_ak.npu AS ak_npu,
    t_ak.npother AS ak_npother,
    t_ak.wetland_freq_unadj AS ak_freq_unadj,
    t_ak.wetland_freq_adj AS ak_freq_adj,
    t_ak.wetland_ind_unadj AS ak_ind_unadj,
    t_ak.wetland_ind_adj AS ak_ind_adj,
    t_hi.ndotw AS hi_ndotw,
    t_hi.ndotu AS hi_ndotu,
    t_hi.ndotother AS hi_ndotother,
    t_hi.npw AS hi_npw,
    t_hi.npu AS hi_npu,
    t_hi.npother AS hi_npother,
    t_hi.wetland_freq_unadj AS hi_freq_unadj,
    t_hi.wetland_freq_adj AS hi_freq_adj,
    t_hi.wetland_ind_unadj AS hi_ind_unadj,
    t_hi.wetland_ind_adj AS hi_ind_adj,
    t_cb.ndotw AS cb_ndotw,
    t_cb.ndotu AS cb_ndotu,
    t_cb.ndotother AS cb_ndotother,
    t_cb.npw AS cb_npw,
    t_cb.npu AS cb_npu,
    t_cb.npother AS cb_npother,
    t_cb.wetland_freq_unadj AS cb_freq_unadj,
    t_cb.wetland_freq_adj AS cb_freq_adj,
    t_cb.wetland_ind_unadj AS cb_ind_unadj,
    t_cb.wetland_ind_adj AS cb_ind_adj,
    v.wetland_ind_unadj_rank_range,
    v.wetland_ind_unadj_rank_variance,
    v.wetland_ind_adj_rank_range,
    v.wetland_ind_adj_rank_variance
FROM
    v_trees_ind_status_region t1
    LEFT OUTER JOIN v_trees_ind_status_region t_ncne
        ON t1.taxon = t_ncne.taxon
```

```
      AND t_ncne.region_abbr = 'NCNE'
   LEFT OUTER JOIN v_trees_ind_status_region t_mw
      ON t1.taxon = t_mw.taxon
      AND t_mw.region_abbr = 'MW'
   LEFT OUTER JOIN v_trees_ind_status_region t_emp
      ON t1.taxon = t_emp.taxon
      AND t_emp.region_abbr = 'EMP'
   LEFT OUTER JOIN v_trees_ind_status_region t_gp
      ON t1.taxon = t_gp.taxon
      AND t_gp.region_abbr = 'GP'
   LEFT OUTER JOIN v_trees_ind_status_region t_aw
      ON t1.taxon = t_aw.taxon
      AND t_aw.region_abbr = 'AW'
   LEFT OUTER JOIN v_trees_ind_status_region t_agcp
      ON t1.taxon = t_agcp.taxon
      AND t_agcp.region_abbr = 'AGCP'
   LEFT OUTER JOIN v_trees_ind_status_region t_wmvc
      ON t1.taxon = t_wmvc.taxon
      AND t_wmvc.region_abbr = 'WMVC'
   LEFT OUTER JOIN v_trees_ind_status_region t_ak
      ON t1.taxon = t_ak.taxon
      AND t_ak.region_abbr = 'AK'
   LEFT OUTER JOIN v_trees_ind_status_region t_hi
      ON t1.taxon = t_hi.taxon
      AND t_hi.region_abbr = 'HI'
   LEFT OUTER JOIN v_trees_ind_status_region t_cb
      ON t1.taxon = t_cb.taxon
      AND t_cb.region_abbr = 'CB'
   LEFT OUTER JOIN v_trees_ind_status_region_variance v
      ON t1.taxon = v.taxon;
------------------------------------------------------------


------------------------------------------------------------
DROP VIEW IF EXISTS v_trees_compare_nwpl CASCADE;
CREATE VIEW v_trees_compare_nwpl AS
SELECT
   t.taxon AS fia_taxon,
   n.species AS nwpl_taxon,
   t.common_name AS fia_common_name,
   n.common_name AS nwpl_common_name,
   n.ncne AS nwpl_ind_ncne,
   t.ncne_ndotw AS fia_ncne_ndotw,
   t.ncne_ndotu AS fia_ncne_ndotu,
```

t.ncne_ndotother **AS** fia_ncne_ndotother,
t.ncne_npw **AS** fia_ncne_npw,
t.ncne_npu **AS** fia_ncne_npu,
t.ncne_npother **AS** fia_ncne_npother,
t.ncne_freq_adj **AS** fia_ncne_freq_adj,
t.ncne_freq_unadj **AS** fia_ncne_freq_unadj,
t.ncne_ind_adj **AS** fia_ncne_ind_adj,
t.ncne_ind_unadj **AS** fia_ncne_ind_unadj,
n.mw **AS** nwpl_ind_mw,
t.mw_ndotw **AS** fia_mw_ndotw,
t.mw_ndotu **AS** fia_mw_ndotu,
t.mw_ndotother **AS** fia_mw_ndotother,
t.mw_npw **AS** fia_mw_npw,
t.mw_npu **AS** fia_mw_npu,
t.mw_npother **AS** fia_mw_npother,
t.mw_freq_adj **AS** fia_mw_freq_adj,
t.mw_freq_unadj **AS** fia_mw_freq_unadj,
t.mw_ind_adj **AS** fia_mw_ind_adj,
t.mw_ind_unadj **AS** fia_mw_ind_unadj,
n.emp **AS** nwpl_ind_emp,
t.emp_ndotw **AS** fia_emp_ndotw,
t.emp_ndotu **AS** fia_emp_ndotu,
t.emp_ndotother **AS** fia_emp_ndotother,
t.emp_npw **AS** fia_emp_npw,
t.emp_npu **AS** fia_emp_npu,
t.emp_npother **AS** fia_emp_npother,
t.emp_freq_adj **AS** fia_freq_emp_adj,
t.emp_freq_unadj **AS** fia_emp_freq_unadj,
t.emp_ind_adj **AS** fia_ind_emp_adj,
t.emp_ind_unadj **AS** fia_emp_ind_unadj,
n.gp **AS** nwpl_ind_gp,
t.gp_ndotw **AS** fia_gp_ndotw,
t.gp_ndotu **AS** fia_gp_ndotu,
t.gp_ndotother **AS** fia_gp_ndotother,
t.gp_npw **AS** fia_gp_npw,
t.gp_npu **AS** fia_gp_npu,
t.gp_npother **AS** fia_gp_npother,
t.gp_freq_adj **AS** fia_freq_gp_adj,
t.gp_freq_unadj **AS** fia_gp_freq_unadj,
t.gp_ind_adj **AS** fia_ind_gp_adj,
t.gp_ind_unadj **AS** fia_gp_ind_unadj,
n.aw **AS** nwpl_ind_aw,
t.aw_ndotw **AS** fia_aw_ndotw,
t.aw_ndotu **AS** fia_aw_ndotu,

t.aw_ndotother **AS** fia_aw_ndotother,
t.aw_npw **AS** fia_aw_npw,
t.aw_npu **AS** fia_aw_npu,
t.aw_npother **AS** fia_aw_npother,
t.aw_freq_adj **AS** fia_freq_aw_adj,
t.aw_freq_unadj **AS** fia_aw_freq_unadj,
t.aw_ind_adj **AS** fia_ind_aw_adj,
t.aw_ind_unadj **AS** fia_aw_ind_unadj,
n.agcp **AS** nwpl_ind_agcp,
t.agcp_ndotw **AS** fia_agcp_ndotw,
t.agcp_ndotu **AS** fia_agcp_ndotu,
t.agcp_ndotother **AS** fia_agcp_ndotother,
t.agcp_npw **AS** fia_agcp_npw,
t.agcp_npu **AS** fia_agcp_npu,
t.agcp_npother **AS** fia_agcp_npother,
t.agcp_freq_adj **AS** fia_agcp_freq_adj,
t.agcp_freq_unadj **AS** fia_agcp_freq_unadj,
t.agcp_ind_adj **AS** fia_agcp_ind_adj,
t.agcp_ind_unadj **AS** fia_agcp_ind_unadj,
n.wmvc **AS** nwpl_ind_wmvc,
t.wmvc_ndotw **AS** fia_wmvc_ndotw,
t.wmvc_ndotu **AS** fia_wmvc_ndotu,
t.wmvc_ndotother **AS** fia_wmvc_ndotother,
t.wmvc_npw **AS** fia_wmvc_npw,
t.wmvc_npu **AS** fia_wmvc_npu,
t.wmvc_npother **AS** fia_wmvc_npother,
t.wmvc_freq_adj **AS** fia_wmvc_freq_adj,
t.wmvc_freq_unadj **AS** fia_wmvc_freq_unadj,
t.wmvc_ind_adj **AS** fia_wmvc_ind_adj,
t.wmvc_ind_unadj **AS** fia_wmvc_ind_unadj,
n.ak **AS** nwpl_ind_ak,
t.ak_ndotw **AS** fia_ak_ndotw,
t.ak_ndotu **AS** fia_ak_ndotu,
t.ak_ndotother **AS** fia_ak_ndotother,
t.ak_npw **AS** fia_ak_npw,
t.ak_npu **AS** fia_ak_npu,
t.ak_npother **AS** fia_ak_npother,
t.ak_freq_adj **AS** fia_ak_freq_adj,
t.ak_freq_unadj **AS** fia_ak_freq_unadj,
t.ak_ind_adj **AS** fia_ak_ind_adj,
t.ak_ind_unadj **AS** fia_ak_ind_unadj,
n.hi **AS** nwpl_ind_hi,
t.hi_ndotw **AS** fia_hi_ndotw,
t.hi_ndotu **AS** fia_hi_ndotu,

```sql
    t.hi_ndotother AS fia_hi_ndotother,
    t.hi_npw AS fia_hi_npw,
    t.hi_npu AS fia_hi_npu,
    t.hi_npother AS fia_hi_npother,
    t.hi_freq_adj AS fia_hi_freq_adj,
    t.hi_freq_unadj AS fia_hi_freq_unadj,
    t.hi_ind_adj AS fia_hi_ind_adj,
    t.hi_ind_unadj AS fia_hi_ind_unadj,
    n.cb AS nwpl_ind_cb,
    t.cb_ndotw AS fia_cb_ndotw,
    t.cb_ndotu AS fia_cb_ndotu,
    t.cb_ndotother AS fia_cb_ndotother,
    t.cb_npw AS fia_cb_npw,
    t.cb_npu AS fia_cb_npu,
    t.cb_npother AS fia_cb_npother,
    t.cb_freq_adj AS fia_cb_freq_adj,
    t.cb_freq_unadj AS fia_cb_freq_unadj,
    t.cb_ind_adj AS fia_cb_ind_adj,
    t.cb_ind_unadj AS fia_cb_ind_unadj,
    t.wetland_ind_unadj_rank_range AS fia_wetland_ind_unadj_rank_range,
    t.wetland_ind_unadj_rank_variance AS fia_wetland_ind_unadj_rank_variance,
    t.wetland_ind_adj_rank_range AS fia_wetland_ind_adj_rank_range,
    t.wetland_ind_adj_rank_variance AS fia_wetland_ind_adj_rank_variance
FROM
    v_trees_ind_status_region_flat t LEFT OUTER JOIN public.nwpl_2013 n
        ON t.taxon = n.species
WHERE
    t.species <> 'spp.'
ORDER BY
    t.taxon;


DROP VIEW IF EXISTS v_trees_compare_arid_west_ca CASCADE;
CREATE VIEW v_trees_compare_arid_west_ca AS
SELECT
    COALESCE(aw.taxon, ca.taxon) AS taxon,
    COALESCE(aw.common_name, ca.common_name) AS common_name,
    aw.npw AS aw_minus_ca_npw,
    aw.npu AS aw_minus_ca_npu,
    aw.npother AS aw_minus_ca_npother,
    aw.ndotw AS aw_minus_ca_ndotw,
    aw.ndotu AS aw_minus_ca_ndotu,
    aw.ndotother AS aw_minus_ca_ndotother,
    aw.wetland_freq_unadj AS aw_minus_ca_freq_unadj,
```

```
    aw.wetland_ind_unadj AS aw_minus_ca_ind_unadj,
    aw.wetland_freq_adj AS aw_minus_minus_ca_freq_adj,
    aw.wetland_ind_adj AS aw_minus_minus_ca_ind_adj,
    ca.npw AS ca_npw,
    ca.npu AS ca_npu,
    ca.npother AS ca_npother,
    ca.ndotw AS ca_ndotw,
    ca.ndotu AS ca_ndotu,
    ca.ndotother AS ca_ndotother,
    ca.wetland_freq_unadj AS ca_freq_unadj,
    ca.wetland_ind_unadj AS ca_ind_unadj,
    ca.wetland_freq_adj AS ca_freq_adj,
    ca.wetland_ind_adj AS ca_ind_adj
  FROM
    v_trees_ind_status_arid_west_minus_ca aw LEFT OUTER JOIN v_trees_ind_status_state
ca
      ON (aw.taxon = ca.taxon AND ca.statecd = 6)
      OR (aw.taxon IS NULL AND ca.statecd = 6)
      OR (ca.taxon IS NULL AND aw.taxon IS NOT NULL)
  ORDER BY
    aw.genus,
    ca.genus,
    aw.species,
    ca.species;


  ----------------------------------------------------------
  DROP VIEW IF EXISTS v_trees_ind_status_region_report CASCADE;
  CREATE VIEW v_trees_ind_status_region_report AS
  SELECT
    taxon,
    common_name,
    region_abbr,
    npw,
    npu,
    npother,
    ndotw,
    ndotu,
    ndotother,
    wetland_freq_unadj,
    wetland_freq_adj,
    wetland_ind_unadj,
    wetland_ind_adj
  FROM
```

```
        v_trees_ind_status_region
    WHERE
        species <> 'spp.'
    ORDER BY
        taxon,
        region_cd;

END;
$BODY$
    LANGUAGE plpgsql VOLATILE
    COST 100;
```

Appendix 20 – SQL program to display FIAD analyses

Filename: fiad_analysis.sql

```
SET search_path TO fiad;

-- SELECT fiad.fiad_summaries();

SELECT * FROM v_trees_ind_status_region_flat WHERE species <> 'spp.' ORDER BY
taxon;
SELECT * FROM v_trees_compare_nwpl;
SELECT * FROM v_trees_compare_arid_west_ca ORDER BY taxon;


-- FIA not found in NWPL
SELECT
   *
FROM
   v_trees_compare_nwpl
WHERE
   nwpl_taxon IS NULL;


-- current NWPL
SELECT
   *
FROM
   public.nwpl_2013 n

-- unadj/adj agree/diagree
SELECT
   'agree' AS agreement,
   *
FROM
   v_trees_ind_status_region
WHERE
   wetland_ind_unadj = wetland_ind_adj
UNION ALL
SELECT
   'disagree' AS agreement,
   *
FROM
   v_trees_ind_status_region
WHERE
```

```
    wetland_ind_unadj <> wetland_ind_adj;

-- what percent agree/disagree?
SELECT
    SUM(
        CASE
        WHEN wetland_ind_unadj = wetland_ind_adj THEN
            1
        ELSE
            0
        END
    ) AS n_agree,
    SUM(
        CASE
        WHEN wetland_ind_unadj <> wetland_ind_adj THEN
            1
        ELSE
            0
        END
    ) AS n_disagree,
    AVG(
        CASE
        WHEN wetland_ind_unadj = wetland_ind_adj THEN
            100
        ELSE
            0
        END
    ) AS pct_agree,
    AVG(
        CASE
        WHEN wetland_ind_unadj <> wetland_ind_adj THEN
            100
        ELSE
            0
        END
    ) AS pct_disagree
FROM
    v_trees_ind_status_region;

-- final output by state (4078 rows)
SELECT
    *
FROM
    v_trees_ind_status_state;
```

```sql
-- adj wetter or drier?
SELECT
  SUM(
    CASE
    WHEN wetland_ind_unadj_rank < wetland_ind_adj_rank THEN
      1
    ELSE
      0
    END
  ) AS n_adj_wetter,
  SUM(
    CASE
    WHEN wetland_ind_unadj_rank > wetland_ind_adj_rank THEN
      1
    ELSE
      0
    END
  ) AS n_adj_drier,
  SUM(
    CASE
    WHEN wetland_ind_unadj_rank = wetland_ind_adj_rank THEN
      1
    ELSE
      0
    END
  ) AS n_adj_same,
  AVG(
    CASE
    WHEN wetland_ind_unadj_rank < wetland_ind_adj_rank THEN
      100
    ELSE
      0
    END
  ) AS pct_adj_wetter,
  AVG(
    CASE
    WHEN wetland_ind_unadj_rank > wetland_ind_adj_rank THEN
      100
    ELSE
      0
    END
  ) AS pct_adj_drier,
```

```sql
    AVG(
        CASE
        WHEN wetland_ind_unadj_rank = wetland_ind_adj_rank THEN
            100
        ELSE
            0
        END
    ) AS pct_adj_same
FROM
    v_trees_ind_status_region;
```

Appendix 21 – SQL program to create summaries of combined FIAD and NPS data

Filename: public_combined_summaries.sql

```
CREATE OR REPLACE FUNCTION public.combined_summaries()
  RETURNS void AS
$BODY$
DECLARE
  rows_affected integer := 0;
BEGIN
  SET search_path TO public;
  RAISE NOTICE 'Total expected runtime: ~2 seconds.';

  DROP VIEW IF EXISTS v_ind_status_region_combined CASCADE;
  CREATE VIEW v_ind_status_region_combined AS
  SELECT
    n.taxon AS taxon,
    n.common_name AS nps_common_name,
    f.common_name AS fia_common_name,
    n.region_cd,
    r.region_abbr,
    r.region_name,
    n.wetland_freq_unadj AS nps_freq_unadj,
    f.wetland_freq_unadj AS fia_freq_unadj,
    (n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 AS mean_freq_unadj,
    n.wetland_freq_adj AS nps_freq_adj,
    f.wetland_freq_adj AS fia_freq_adj,
    (n.wetland_freq_adj + f.wetland_freq_adj) / 2 AS mean_freq_adj,
    CASE
    WHEN n.wetland_freq_unadj >= 0.99 THEN
      'OBL'
    WHEN n.wetland_freq_unadj >= 0.67 AND n.wetland_freq_unadj < 0.99 THEN
      'FACW'
    WHEN n.wetland_freq_unadj >= 0.33 AND n.wetland_freq_unadj < 0.67 THEN
      'FAC'
    WHEN n.wetland_freq_unadj > 0.01 AND n.wetland_freq_unadj < 0.33 THEN
      'FACU'
    WHEN n.wetland_freq_unadj <= 0.01 THEN
      'UPL'
    ELSE
      '?'
    END AS nps_ind_unadj,
    CASE
```

```
WHEN f.wetland_freq_unadj >= 0.99 THEN
   'OBL'
WHEN f.wetland_freq_unadj >= 0.67 AND f.wetland_freq_unadj < 0.99 THEN
   'FACW'
WHEN f.wetland_freq_unadj >= 0.33 AND f.wetland_freq_unadj < 0.67 THEN
   'FAC'
WHEN f.wetland_freq_unadj > 0.01 AND f.wetland_freq_unadj < 0.33 THEN
   'FACU'
WHEN f.wetland_freq_unadj <= 0.01 THEN
   'UPL'
ELSE
   '?'
END AS fia_ind_unadj,
CASE
WHEN (n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 >= 0.99 THEN
   'OBL'
WHEN (n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 >= 0.67 AND
(n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 < 0.99 THEN
   'FACW'
WHEN (n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 >= 0.33 AND
(n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 < 0.67 THEN
   'FAC'
WHEN (n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 > 0.01 AND
(n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 < 0.33 THEN
   'FACU'
WHEN (n.wetland_freq_unadj + f.wetland_freq_unadj) / 2 <= 0.01 THEN
   'UPL'
ELSE
   '?'
END AS mean_ind_unadj,
CASE
WHEN n.wetland_freq_adj >= 0.99 THEN
   'OBL'
WHEN n.wetland_freq_adj >= 0.67 AND n.wetland_freq_adj < 0.99 THEN
   'FACW'
WHEN n.wetland_freq_adj >= 0.33 AND n.wetland_freq_adj < 0.67 THEN
   'FAC'
WHEN n.wetland_freq_adj > 0.01 AND n.wetland_freq_adj < 0.33 THEN
   'FACU'
WHEN n.wetland_freq_adj <= 0.01 THEN
   'UPL'
ELSE
   '?'
END AS nps_ind_adj,
```

```sql
CASE
WHEN f.wetland_freq_adj >= 0.99 THEN
   'OBL'
WHEN f.wetland_freq_adj >= 0.67 AND f.wetland_freq_adj < 0.99 THEN
   'FACW'
WHEN f.wetland_freq_adj >= 0.33 AND f.wetland_freq_adj < 0.67 THEN
   'FAC'
WHEN f.wetland_freq_adj > 0.01 AND f.wetland_freq_adj < 0.33 THEN
   'FACU'
WHEN f.wetland_freq_adj <= 0.01 THEN
   'UPL'
ELSE
   '?'
END AS fia_ind_adj,
CASE
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.99 THEN
   'OBL'
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.67 AND (n.wetland_freq_adj + f.wetland_freq_adj) / 2 < 0.99 THEN
   'FACW'
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.33 AND (n.wetland_freq_adj + f.wetland_freq_adj) / 2 < 0.67 THEN
   'FAC'
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 > 0.01 AND (n.wetland_freq_adj + f.wetland_freq_adj) / 2 < 0.33 THEN
   'FACU'
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 <= 0.01 THEN
   'UPL'
ELSE
   '?'
END AS mean_ind_adj,
CASE
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.99 THEN
   5
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.67 AND (n.wetland_freq_adj + f.wetland_freq_adj) / 2 < 0.99 THEN
   4
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.33 AND (n.wetland_freq_adj + f.wetland_freq_adj) / 2 < 0.67 THEN
   3
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 > 0.01 AND (n.wetland_freq_adj + f.wetland_freq_adj) / 2 < 0.33 THEN
   2
WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 <= 0.01 THEN
```

```
      1
    ELSE
      NULL
    END AS mean_ind_unadj_rank,
    CASE
    WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.99 THEN
      5
    WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.67 AND (n.wetland_freq_adj +
f.wetland_freq_adj) / 2 < 0.99 THEN
      4
    WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 >= 0.33 AND (n.wetland_freq_adj +
f.wetland_freq_adj) / 2 < 0.67 THEN
      3
    WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 > 0.01 AND (n.wetland_freq_adj +
f.wetland_freq_adj) / 2 < 0.33 THEN
      2
    WHEN (n.wetland_freq_adj + f.wetland_freq_adj) / 2 <= 0.01 THEN
      1
    ELSE
      NULL
    END AS mean_ind_adj_rank
  FROM
    nps.v_spp_freq_region n
    INNER JOIN fiad.v_trees_freq_region f
      ON n.taxon = f.taxon
      AND n.region_cd = f.region_cd
    INNER JOIN public.regions as r
      ON n.region_cd = r.region_cd
  ORDER BY
    n.taxon,
    n.region_cd;
  RAISE NOTICE 'Create view of indicator status of species by regions for FIA and NPS
combined.';


  DROP VIEW IF EXISTS v_ind_status_region_combined_variance CASCADE;
  CREATE VIEW v_ind_status_region_combined_variance AS
  SELECT
    taxon,
    MAX(mean_ind_unadj_rank) - MIN(mean_ind_unadj_rank) AS
mean_ind_unadj_rank_range,
    VAR_POP(mean_ind_unadj_rank) AS mean_ind_unadj_rank_variance,
    MAX(mean_ind_adj_rank) - MIN(mean_ind_adj_rank) AS mean_ind_adj_rank_range,
    VAR_POP(mean_ind_adj_rank) AS mean_ind_adj_rank_variance
```

```
FROM
    v_ind_status_region_combined
GROUP BY
    taxon;


DROP VIEW IF EXISTS v_ind_status_region_combined_flat CASCADE;
CREATE VIEW v_ind_status_region_combined_flat AS
SELECT DISTINCT ON (s1.taxon)
    s1.taxon,
    s1.nps_common_name,
    s1.fia_common_name,
    s_ncne.mean_freq_unadj AS ncne_mean_freq_unadj,
    s_ncne.mean_freq_adj AS ncne_mean_freq_adj,
    s_ncne.mean_ind_unadj AS ncne_mean_ind_unadj,
    s_ncne.mean_ind_adj AS ncne_mean_ind_adj,
    s_mw.mean_freq_unadj AS mw_mean_freq_unadj,
    s_mw.mean_freq_adj AS mw_mean_freq_adj,
    s_mw.mean_ind_unadj AS mw_mean_ind_unadj,
    s_mw.mean_ind_adj AS mw_mean_ind_adj,
    s_emp.mean_freq_unadj AS emp_mean_freq_unadj,
    s_emp.mean_freq_adj AS emp_mean_freq_adj,
    s_emp.mean_ind_unadj AS emp_mean_ind_unadj,
    s_emp.mean_ind_adj AS emp_mean_ind_adj,
    s_gp.mean_freq_unadj AS gp_mean_freq_unadj,
    s_gp.mean_freq_adj AS gp_mean_freq_adj,
    s_gp.mean_ind_unadj AS gp_mean_ind_unadj,
    s_gp.mean_ind_adj AS gp_mean_ind_adj,
    s_aw.mean_freq_unadj AS aw_mean_freq_unadj,
    s_aw.mean_freq_adj AS aw_mean_freq_adj,
    s_aw.mean_ind_unadj AS aw_mean_ind_unadj,
    s_aw.mean_ind_adj AS aw_mean_ind_adj,
    s_agcp.mean_freq_unadj AS agcp_mean_freq_unadj,
    s_agcp.mean_freq_adj AS agcp_mean_freq_adj,
    s_agcp.mean_ind_unadj AS agcp_mean_ind_unadj,
    s_agcp.mean_ind_adj AS agcp_mean_ind_adj,
    s_wmvc.mean_freq_unadj AS wmvc_mean_freq_unadj,
    s_wmvc.mean_freq_adj AS wmvc_mean_freq_adj,
    s_wmvc.mean_ind_unadj AS wmvc_mean_ind_unadj,
    s_wmvc.mean_ind_adj AS wmvc_mean_ind_adj,
    s_ak.mean_freq_unadj AS ak_mean_freq_unadj,
    s_ak.mean_freq_adj AS ak_mean_freq_adj,
    s_ak.mean_ind_unadj AS ak_mean_ind_unadj,
    s_ak.mean_ind_adj AS ak_mean_ind_adj,
```

```
    s_hi.mean_freq_unadj AS hi_mean_freq_unadj,
    s_hi.mean_freq_adj AS hi_mean_freq_adj,
    s_hi.mean_ind_unadj AS hi_mean_ind_unadj,
    s_hi.mean_ind_adj AS hi_mean_ind_adj,
    s_cb.mean_freq_unadj AS cb_mean_freq_unadj,
    s_cb.mean_freq_adj AS cb_mean_freq_adj,
    s_cb.mean_ind_unadj AS cb_mean_ind_unadj,
    s_cb.mean_ind_adj AS cb_mean_ind_adj,
    v.mean_ind_unadj_rank_range,
    v.mean_ind_unadj_rank_variance,
    v.mean_ind_adj_rank_range,
    v.mean_ind_adj_rank_variance
FROM
    v_ind_status_region_combined s1
    LEFT OUTER JOIN v_ind_status_region_combined s_ncne
        ON s1.taxon = s_ncne.taxon
        AND s_ncne.region_abbr = 'NCNE'
    LEFT OUTER JOIN v_ind_status_region_combined s_mw
        ON s1.taxon = s_mw.taxon
        AND s_mw.region_abbr = 'MW'
    LEFT OUTER JOIN v_ind_status_region_combined s_emp
        ON s1.taxon = s_emp.taxon
        AND s_emp.region_abbr = 'EMP'
    LEFT OUTER JOIN v_ind_status_region_combined s_gp
        ON s1.taxon = s_gp.taxon
        AND s_gp.region_abbr = 'GP'
    LEFT OUTER JOIN v_ind_status_region_combined s_aw
        ON s1.taxon = s_aw.taxon
        AND s_aw.region_abbr = 'AW'
    LEFT OUTER JOIN v_ind_status_region_combined s_agcp
        ON s1.taxon = s_agcp.taxon
        AND s_agcp.region_abbr = 'AGCP'
    LEFT OUTER JOIN v_ind_status_region_combined s_wmvc
        ON s1.taxon = s_wmvc.taxon
        AND s_wmvc.region_abbr = 'WMVC'
    LEFT OUTER JOIN v_ind_status_region_combined s_ak
        ON s1.taxon = s_ak.taxon
        AND s_ak.region_abbr = 'AK'
    LEFT OUTER JOIN v_ind_status_region_combined s_hi
        ON s1.taxon = s_hi.taxon
        AND s_hi.region_abbr = 'HI'
    LEFT OUTER JOIN v_ind_status_region_combined s_cb
        ON s1.taxon = s_cb.taxon
        AND s_cb.region_abbr = 'CB'
```

```
    LEFT OUTER JOIN v_ind_status_region_combined_variance v
      ON s1.taxon = v.taxon;


DROP VIEW IF EXISTS v_compare_combined_nwpl CASCADE;
CREATE VIEW v_compare_combined_nwpl AS
SELECT
    c.taxon AS nps_fia_taxon,
    n.species AS nwpl_taxon,
    c.nps_common_name AS nps_common_name,
    c.fia_common_name AS fia_common_name,
    n.common_name AS nwpl_common_name,
    n.ncne AS nwpl_ind_ncne,
    c.ncne_mean_freq_unadj AS nps_fia_ncne_mean_freq_unadj,
    c.ncne_mean_freq_adj AS nps_fia_ncne_mean_freq_adj,
    c.ncne_mean_ind_unadj AS nps_fia_ncne_mean_ind_unadj,
    c.ncne_mean_ind_adj AS nps_fia_ncne_mean_ind_adj,
    n.mw AS nwpl_ind_mw,
    c.mw_mean_freq_unadj AS nps_fia_mw_mean_freq_unadj,
    c.mw_mean_freq_adj AS nps_fia_mw_mean_freq_adj,
    c.mw_mean_ind_unadj AS nps_fia_mw_ind_unadj,
    c.mw_mean_ind_adj AS nps_fia_mw_mean_ind_adj,
    n.emp AS nwpl_ind_emp,
    c.emp_mean_freq_unadj AS nps_fia_emp_mean_freq_unadj,
    c.emp_mean_freq_adj AS nps_fia_mean_freq_emp_adj,
    c.emp_mean_ind_unadj AS nps_fia_emp_mean_ind_unadj,
    c.emp_mean_ind_adj AS nps_fia_mean_ind_emp_adj,
    n.gp AS nwpl_ind_gp,
    c.gp_mean_freq_unadj AS nps_fia_gp_mean_freq_unadj,
    c.gp_mean_freq_adj AS nps_fia_mean_freq_gp_adj,
    c.gp_mean_ind_unadj AS nps_fia_gp_mean_ind_unadj,
    c.gp_mean_ind_adj AS nps_fia_mean_ind_gp_adj,
    n.aw AS nwpl_ind_aw,
    c.aw_mean_freq_unadj AS nps_fia_aw_mean_freq_unadj,
    c.aw_mean_freq_adj AS nps_fia_mean_freq_aw_adj,
    c.aw_mean_ind_unadj AS nps_fia_aw_mean_ind_unadj,
    c.aw_mean_ind_adj AS nps_fia_mean_ind_aw_adj,
    n.agcp AS nwpl_ind_agcp,
    c.agcp_mean_freq_unadj AS nps_fia_agcp_mean_freq_unadj,
    c.agcp_mean_freq_adj AS nps_fia_agcp_mean_freq_adj,
    c.agcp_mean_ind_unadj AS nps_fia_agcp_mean_ind_unadj,
    c.agcp_mean_ind_adj AS nps_fia_agcp_mean_ind_adj,
    n.wmvc AS nwpl_ind_wmvc,
    c.wmvc_mean_freq_unadj AS nps_fia_wmvc_mean_freq_unadj,
```

```
        c.wmvc_mean_freq_adj AS nps_fia_wmvc_mean_freq_adj,
        c.wmvc_mean_ind_unadj AS nps_fia_wmvc_mean_ind_unadj,
        c.wmvc_mean_ind_adj AS nps_fia_wmvc_mean_ind_adj,
        n.ak AS nwpl_ind_ak,
        c.ak_mean_freq_unadj AS nps_fia_ak_mean_freq_unadj,
        c.ak_mean_freq_adj AS nps_fia_ak_mean_freq_adj,
        c.ak_mean_ind_unadj AS nps_fia_ak_mean_ind_unadj,
        c.ak_mean_ind_adj AS nps_fia_ak_mean_ind_adj,
        n.hi AS nwpl_ind_hi,
        c.hi_mean_freq_unadj AS nps_fia_hi_mean_freq_unadj,
        c.hi_mean_freq_adj AS nps_fia_hi_mean_freq_adj,
        c.hi_mean_ind_unadj AS nps_fia_hi_mean_ind_unadj,
        c.hi_mean_ind_adj AS nps_fia_hi_mean_ind_adj,
        n.cb AS nwpl_ind_cb,
        c.cb_mean_freq_unadj AS nps_fia_cb_mean_freq_unadj,
        c.cb_mean_freq_adj AS nps_fia_cb_mean_freq_adj,
        c.cb_mean_ind_unadj AS nps_fia_cb_mean_ind_unadj,
        c.cb_mean_ind_adj AS nps_fia_cb_mean_ind_adj,
        c.mean_ind_unadj_rank_range,
        c.mean_ind_unadj_rank_variance,
        c.mean_ind_adj_rank_range,
        c.mean_ind_adj_rank_variance
    FROM
        v_ind_status_region_combined_flat c LEFT OUTER JOIN public.nwpl_2013 n
            ON c.taxon = n.species
    ORDER BY
        c,taxon;


END;
$BODY$
    LANGUAGE plpgsql VOLATILE
    COST 100;
```

Appendix 22 – SQL program to display summaries of combined FIAD and NPS data

Filename: public_combined_analysis.sql

```
SET search_path TO public;

select * from v_ind_status_region_combined_flat ORDER BY taxon;
select * from v_compare_combined_nwpl ORDER BY nwpl_taxon;
```