Title: **SQL-Based Analysis of the Sensitivity of Soil Water Holding Capacity to Methods of Spatial Interpolation**
Name: **Michael Felzan**
Course: **GIS 5577**
Date: **5/12/22**
GitHub Repository: https://github.com/MGIS-UMN/class-project-fezfelzan

Introduction

There are many variables which control the input and output of water in an agricultural system. We may think of soil as a 'bucket' in these kinds of systems, as soil has the ability to store water. Transpiration, evaporation, runoff, and deep percolation displace water from the bucket, while rain, upward flow of groundwater, and irrigation put water back into the bucket. Irrigation is the only variable we as humans are able to control, as a means for making sure the water in the bucket never gets too low. The goal of irrigation is to apply the right amount of water to keep soil-water levels in the optimum range, considering the effects of all other variables at play. The optimum range of soil-water levels is above the "trigger point" where plants become stressed or wilt, but not so far above the field capacity point where water is being wasted due to complete soil saturation.

The composition of a soil (silt, sand, loam, clay, etc.) affects its ability to retain water. Soils are generally discretized based on composition differences or consistencies. Although changes in soil composition are most likely more gradual in nature, soils are conventionally broken down into discrete horizons in 3D space, based on notable changes in soil composition. The vertical (depth) and horizontal (XY-dimension) bounding lines that subdivide horizons are inconsistent, which complicates the summarization of Available Water Storage (AWS) for an entire agricultural field. Because soil AWS is determined at the horizon-level (as it is a function of soil composition), there are two interpolation operations required to move from soil-horizon-AWS to field-level-AWS at a given depth range: one in the vertical "Z-dimension," and one in the horizontal "XY-dimension."

The US Department of Agriculture (USDA) Natural Resources Conservation Service's (NRCS) Soil Survey Geographic Database (abbreviated as "SSURGO") is one of the most widely used soil datasets in studies concerned with the spatial distribution of soil types. The data model for this database (**Appendix B.1**) is a robust and complicated collection of tables and shapefiles, which are interconnected through primary/foreign keys. However, it seems that many people who this dataset end up working with only a few of the tables in the database, as confirmed in the User Guide for this dataset. These tables are the horizon table ("chorizon"), the component table ("component), and the map unit table ("mapunit"), and they are fundamental to the dataset's structure and theoretical framework.

The SSURGO database revolves around a "Map Unit" multipolygon layer (but also provided in raster form), where each of the map units represent distinct soil regions across geographic space. Each of these polygons has a distinct map unit key ("MUKEY") in its attribute table, which is linked to various other tables. These map units are a coarse discretization of soil types, as there are multiple "Components" (most easily thought of as phases of the soil series) within each of these map units. Each component has its own distinct key, and is linked to the map unit key (MUKEY) by which it is contained. Additionally, there are multiple different and distinct soil *Horizon* layers within a single soil Component – however, these soil horizons occur on *the Z-dimension* with accompanying depth attributes. The Map Unit layer is the only layer described here which has inherent spatial attributes (geom), besides the

tabular depth attribute data provided in the horizon table. Because of this, in order to use the SSURGO datasets to summarize *field*-level Available Water Storage (AWS), via soil horizon AWS values at a given depth interval, available water storage must first be aggregated in Z-dimension (horizons→components/map units), and then spatially interpolated in the XY-dimension (moving from SSURGO map unit boundaries to parcel boundaries).

Project Objective

This project is an extension of internship work I'm completing for the MN LCCMR. The LCCMR is working to build an Irrigation Management Assistant tool, which aims to programmatically approximate the amount irrigation that should be applied to fields throughout growing seasons. This project requires the reporting of soil-water holding capacity at the parcel-level, in order to make judgements about the amount of water a given field should receive. The LCCMR is particularly concerned with soil-water holding capacity metrics at four distinct soil-depth intervals: 0-12inches, 12-24inches, 24-36inches, and 36-48inches. SSURGO provides a pre-aggregated map unit table (MUAGGATT) in their dataset which offers map-unit-level soil-water holding capacity values, though the table only provides values for the depth intervals of 0-25cm, 0-50cm, 0-100cm, and 0-150cm.

The objective of this project may be broken into two research questions. The first of these questions is to determine, with the documentation available, *how* SSURGO arrived at their map-unit-level available water storage (AWS) values, and to write a script that may (approximately) reproduce their results – so that *custom* depth intervals may be inputted into the script. The second objective of this project is to determine, after having used the script described above to calculate map-unit AWS at 0-12in/12-24in/24-36in/36-48in, the extent which different spatial interpolation algorithms affect final field-level AWS calculations, as one moves from SSURGO map-unit geometries to parcel geometries. The algorithms in question here are area-weighted average, inverse distance weighting (IDW) and area-weighted averaging using two different heuristics: 1. If a soil type takes up >80% of a parcel's area, then use that soil type's AWS for the whole parcel; 2. If a soil type takes up <20% of a parcel's area, do not include that soil type in the area-weighted averaging.

Database Description

The database model proposed in this project is designed to present data and results to the LCCMR as an organized package, allowing for easy interpretation of results and future analysis using the data collected. **Figure 1** provides an entity-relationship model for this database. The tables on the left portion of **Figure 1** are built from data generated by SQL scripts I have written for this project, and the tables on the right are the source data I used to generate my results. I chose to include the source data in my database because I believe it improves data quality, or rather the ability to verify/reproduce my generated results. However, the source data included has been considerably trimmed down (especially the SSURGO data; **Appendix B.1**). My hope here is that through providing *only* the essential data needed to perform the analyses in this study, this database may serve a utility to anyone pursuing similar analyses.

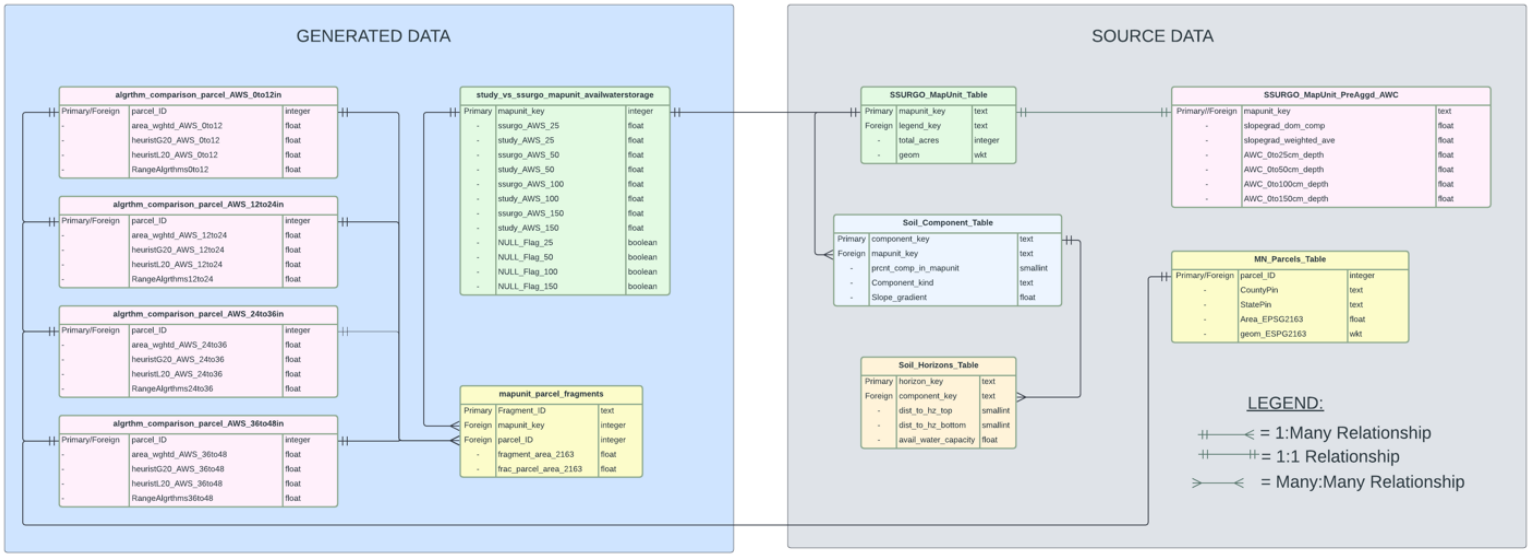


Figure 1. *Database entity-relationship diagram (full-size PNG included in GitHub repository)*

The organization of my generated data may be viewed in the context of the two research objectives in this project. The first objective, to replicate the algorithm SSURGO used to aggregate soil horizon Available Water Storage (AWS) values to the map-unit level, requires comparison of the map-unit AWS values SSURGO reported for the soil depth intervals of 0-25cm, 0-50cm, 0-100cm, and 0-150, and the values my own script generated at those same depth intervals. Packaging all of these values together in the "*study vs ssurgo mapunit availwaterstorage*" table allows for the calculation of R^2 and RMSE between the values generated by my script and SSURGO's, using the Map Unit ID as the primary key which joins the two sets of data together. Additional fields signaling NULL values within a map unit table entry may allow for interpretations of why discrepancies exist between the calculated values.

The leftmost tables, labeled *"algrthm comparison parcel AWS [Z_1] to [Z_2] in,"* reflect the objective of Research Question 2 – to assess the variation in reported *parcel*-level AWS, imparted by the use of differing interpolation algorithms. Here, the Parcel ID is the primary key for each of these tables, and the other columns describe parcel AWS values generated by the different algorithms. An additional column is appended to these tables which provides the range (min and max) of all of the algorithms for an easier assessment of variance. Originally, these tables were all joined together as one large table, though to improve legibility or understanding of the data, I decided to break them out into four separate tables.

The last table in the subset of 'generated data,' *"mapunit parcel fragments,"* is a table I had to create for my analysis, and therefore thought was worth including – to provide anyone interested in a similar study (or verifying my results) with the means to do so. The data in this table is the result of an ST_Intersection() function run on the SSURGO map unit polygons and the Minnesota parcels polygon layers, so it is essentially a mosaic of fragments defined the areas where these two layers intersect (**Figure 4**). Each of these fragments contains both the map unit ID and parcel ID which defines the fragment polygon, along with the area of fragment, and the total area of its "parent" parcel. This table is further explained in the Research Question 2 section.

Data

	Title	Purpose in Analysis	Link to Source
1	Statewide soil composition (XY&Z dimensions) (via gSSURGO dataset)	Parsing out various soil component percentages across the extent of each gSSURGO "map unit" and at varying depths within units (0-25, 0-50cm, etc.) to address the question of how SSURGO arrives at their available water storage calculations aggregated by map units (*research question 2*).	USDA-NRCS Soil Survey Geographic Database
2	Available Water Storage 0-25cm; 0-50cm; 0-100cm; 0-150cm (via gSSURGO dataset)	SSURGO provides Available Water Storage (AWS) values for each of their map units, aggregated at the depths of 0-25cm, 0-50cm, 0-100cm, and 0-150cm in their '*muggatt*' table. This table will be used to assess if the use of different algorithms in summarizing map unit AWS values to parcel boundaries results in substantial variance between results.	USDA-NRCS Soil Survey Geographic Database
3	MN County Parcel Data (Statewide)	Aggregating gSSURGO soil component data to agricultural fields	(Requested and received from UMN)

Table 1. *Source Data*

The NRCS gSSURGO dataset is the main input (source) data for this study. This dataset is made up of a series of tables which contain unique soil statistics for a series of map polygons comprising MN. Tables of particular interest include the 'MUAGGATT' table (contains summarized AWS values for each polygon at 0-25, 0-50, 0-100, 0-150cm), the 'chorizon' table (contains 'hztept'/'hzdepb,' distance from top/bottom of soil horizon to surface), and the 'component' table (contains 'comppct,' % of component with gSSURGO map unit). Additionally, the MN County Parcel dataset (available via the University of Minnesota) is used as source data to examine the effects of various algorithms in summarizing gSSURGO map unit AWS values to larger parcel divisions.

Data generated within this project include my script's reported SSURGO map unit AWS values at the depth intervals of 0-25cm, 0-50cm, 0-100cm, and 0-150cm, packaged alongside SSURGO's reported values for the same map units at the depth intervals in question. My script also identifies which SSURGO map units contain missing data. The other main "generated" data in this study is a set of AWS metrics for all *parcels* in Minnesota. Field-level available water storage is calculated for all parcels using four different spatial interpolation algorithms, and the resultant values from each of these algorithms are provided in a single table, linked by each parcel's ID (**Figure 1**).

Methods & Data Analysis

Loading data

The SSURGO data used in this study was downloaded from the USDA-NRCS website in the form of an Esri geodatabase. The only data from the SSURGO database used in this study were the map unit multipolygon shapefile (and associated attribute table), the components table, the horizons table, and the pre-aggregated map unit values table. The geodatabase was opened in ArcGIS Pro, and a copy of

the map unit multipolygon layer was exported as a shapefile; the rest of the necessary tables were exported as .CSV's.

All of the tables (component, horizon, pre-aggregated) were imported to the classroom database using Create Table and Insert statements, assigning the data types and aliases for each table. The map unit shapefile was loaded into the classroom database using the shp2pgsql function included in the PGAdmin package. **Appendix C.1** provides an example of calling this tool at the command line to load a shapefile into a SQL database row-by-row. The MN Countywide parcels shapefile was similarly loaded using shp2pgsql.

Research Question 1.

My first step in approaching Research Question 1 (creating a script which mimics SSURGO's process for aggregating soil horizon AWS at a given depth interval) was to join the mapunit, component, and horizon tables together, based on the keys that define each of their one-to-many relationships (**Appendix A.1**). These joins are necessary because soil Available Water Storage (AWS) is reported by SSURGO only at the *horizon* level. The available SSURGO documentation implies that an area-weighted average was used to summarize all horizons (and their respective AWS values) in a given depth interval, using horizon *thickness* as the "weight." In order to write a SQL script that summarizes AWS for all horizons in a given depth bracket based on thickness, I first needed to establish a method for "updating" the thickness of layers which fell across the bounding lines of the depth bracket(s), as a horizon's full thickness would not be represented in such case. **Figure 2** provides an illustration of this: if a horizon extends from 9-32cm (thickness=23cm), but the depth interval only extends to the 25cm line, the thickness of this horizon needs to be updated to only reflect the area enclosed within the interval (9cm-25cm; thickness=16). To address this, I created a SQL script which queries all horizon layers which fall along interval bounding lines, emplaces boolean 'flags' for rows that have on-bounding-line contenders, calculates the *updated* thickness of contending horizons, and appends these new thickness values to the original dataset. A 'SQL CASE' statement was then used to determine the "working thickness" for all horizons (eg. if the horizon falls on a bounding line in the depth interval, use the updated thickness; if not, use the original thickness). The area-weighted average calculation was carried out using 'GROUP BY' statements; eg. available water storage = SUM(horizon thickness fraction of whole * horizon AWS), grouped by the soil component each horizon falls within.

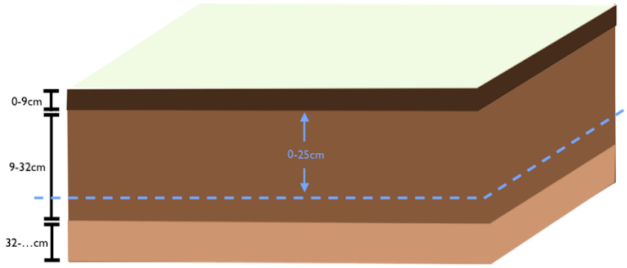


Figure 2. *Illustration depicting a case when a horizon layer extends across the soil depth interval bounding line.*

Research Question 2.

The motivation driving the first research question was to establish a script which could summarize SSURGO map-unit soil Available Water Storage (AWS) at *any* user-specified depth interval, because the LCCMR is particularly interested in the soil depth intervals of 0-12in, 12-24in, 24-36in, and 36-48in – which SSURGO does not provide values for in their pre-aggregated "MUAGGATT" table.

After using the script written for Research Question 1. to obtain map-unit AWS values at these four depth intervals, the objective of Research Question 2 is to summarize map-unit-AWS, at each of the four depth intervals, to the *parcel/field*-level, using different spatial interpolation algorithms (area-weighted average, area weighted average using "heuristics;" at the time of writing this report, IDW has not yet been considered). The generation of this data may be used to assess how different interpolation methods affect the final calculation of field-level AWS.

The area-weighted average calculation was performed using a set of SQL queries which mirror how ArcGIS's "Union" tool works (**Figure 3**). The ST_Intersect**ion**() function was used on both the SSURRGO map unit polygon layer and the MN countywide polygon layer. This resulted in a mosaic of small fragments, created from the areas where polygon borders of both layers intersected (**Figure 4**). **Appendix C.3** provides the code used for this operation. In this query, the IDs of both the parcels and map units which define each fragment are saved in their corresponding attribute table via the ST_Intersect**s**() function. In doing so, parcel-level AWS may be calculated via "grouping by" parcel ID, where parcel AWS is equal to the sum of all *fragment*-AWS values multiplied by the fraction area each fragment occupies in the total parcel area. To calculate area for these layers, I first projected the data into an equal area projection (EPSG:2163) using ST_Transform(). The original spatial and attribute information for the "fragments" layer is stored in the "mapunit_parcel_fragments" table in my database (**Figure 1**).

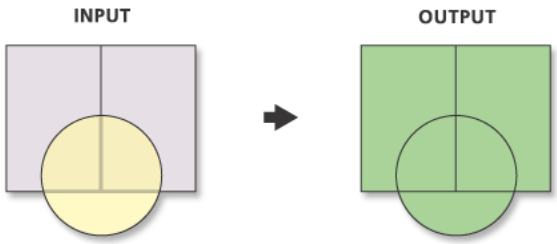


Figure 3. *Illustration describing the functionality of ArcGIS's "Union" tool (source: Esri).*

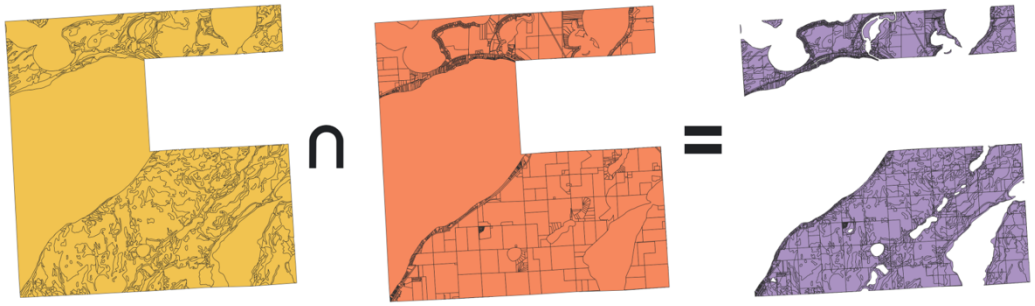


Figure 4. *Layer created via an St_Intersection() on the map unit polygon layer (left) and the MN countywide parcels layer (middle). Script used to create layer in Appendix C.3*

The LCCMR is interested in how applying certain heuristics during the area-weighted averaging of map unit AWS affects the arrived values of field-level AWS. The two heuristics in question are:

1. If a soil type occupies >80% of a parcel, use that soil's AWS for the whole parcel.
2. If a soil type occupies <20% of a parcel, do not consider soil type in the area-weighted average calculation.

For the ">80%" heuristic described above, I queried all parcels which had fragments (same layer from previous section) which took up >80% of the parcel's total area, and created a new field in the 'fragments table' which flagged these parcel IDs as such ("has >80% fragment" field; boolean 'true'/'false'). I also queried all fragments that took up >80% of a parcel's area, and mapped their AWS values to all

fragments sharing the same parcel IDs in a separate field ('aws_greaterthan80'). By doing this, I was able to utilize a SQL 'CASE' statement during the area-weighted average operation, where the same area-weighted average operation seen in the previous section is performed, *unless* the parcel is flagged as having a fragment which takes up >80% of the area. In this case that a parcel is flagged, the aggregate function used in the GROUP BY is not SUM(), but instead an AVE() of all of the "aws_greaterthan80" values in the parcel. Because all of the "aws_greaterthan80" values for fragments in a parcel where one takes up >80% are identical, the average is essentially a means for selecting a single value during the GROUP BY.

For the "<20%" heuristic described above, a different set of logic-based SQL queries were needed. This is due to the fact that if soil types are "dropped" from the calculation of area-weighted average, the 'total' area of the parcel is altered, which changes the fractional area each fragment takes up in a parcel (its weight). To address this, I first queried all fragments that took up <20% of a parcels area and saved the entries in a temporary table. All parcel ID's occurring in this temporary table were given a 'flag=true' attribute in the original dataset, similar to the above heuristic operation. I calculated the new 'total' area of the parcels for the contending parcels with a <20% fragment, along with the updated fractional area each fragment occupies in their parent parcel. By doing so, when conducting the final area-weighted average operation, if a parcel ID is flagged as having a <20% area fragment, the updated fractional areas are selected instead of the originals.

Applications of Database

The structure of my database was mainly designed to aid the LCCMR's Irrigation Management Assistant project. The "*study vs ssurgo mapunit availwaterstorage*" table allows the databaser user to query SSURGO's pre-aggregated and my script's calculated map-unit AWS values for each of the four depth intervals, to understand the discrepancies in our approaches (variance/R^2/RMSE). The user could query data from a specific soil depth interval, and compare it to data from another interval. Using the four *"algrthm comparison parcel AWS [Z_1] to [Z_2] in,"* the database user may query data by parcel size, algorithm type, or by one of the four depth intervals, to further address how the interplay of these variables affects field-level available water storage reporting. My intention for including the source data used to generate the data in these tables was to allow for results reproducibility, but to also permit any novel queries the user may have.

Challenges

One of the biggest challenges of this project was understanding how to navigate the massive SSURGO database to accomplish a very specific task. Possibly due to the database's size and vast amount of possible applications, I found the available documentation surrounding my research objective to be generalized and best and non-existent at worst. Additionally, I had little prior knowledge of soil geology before this project, an issue was complicated by the fact that SSURGO created their own terminology/theoretical framework for the dataset (it took me a couple weeks to understand what a SSURGO soil "component" is, and I'm still not sure I fully do).

The size of the dataset also posed challenges in regards to loading data into our classroom database. There are 2,123,552 map units in SSURGO's MUPOLYGON layer, which, with my internet speed, makes for a ~20-hour upload into the database using shp2pgsql. This issue was magnified by the

fact that when I first imported the layer into the database, I assigned the wrong projection, and on my second try importing, the script errored out on one of the final rows. A 20-hour upload is not easy when you are in grad school and need to transport your computer everywhere (and thereby forfeiting WIFI connection with each new commute). Additionally, querying and performing spatial operations on this much data takes multiple minutes each run, which made it challenging to troubleshoot my code.

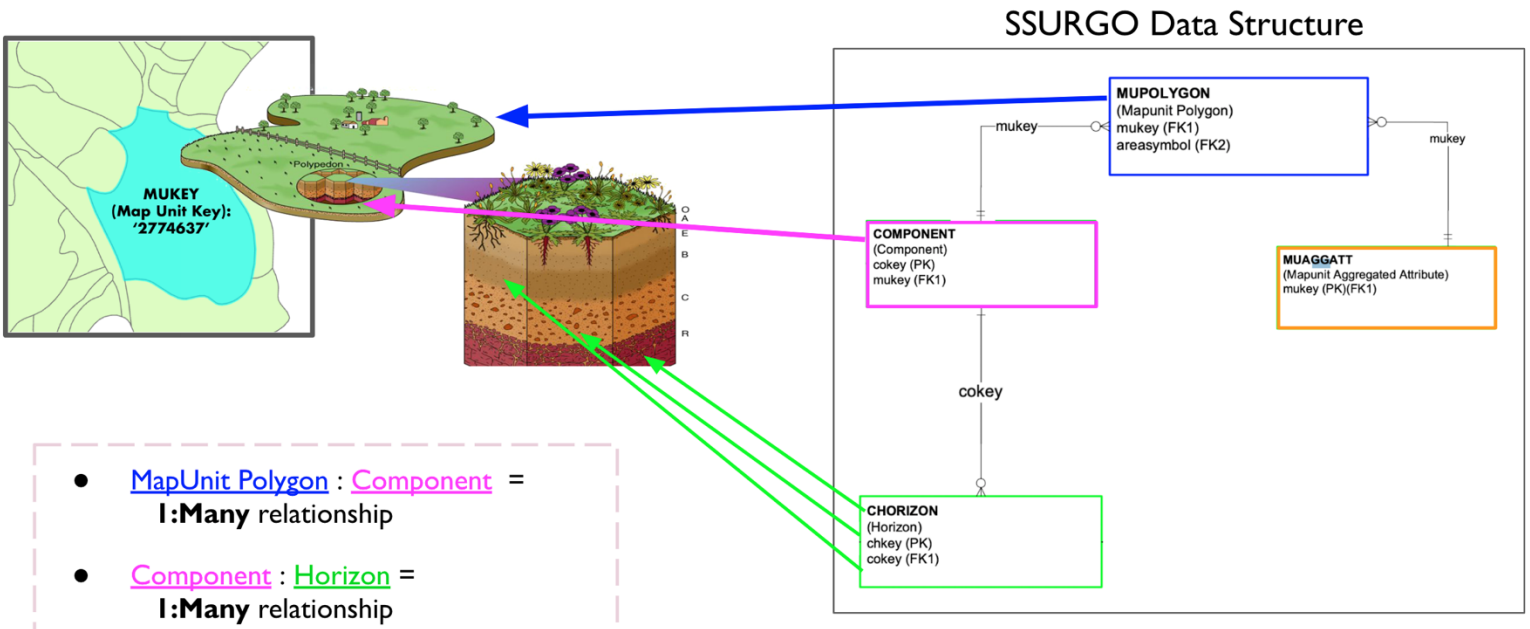
While completing Research Question 1., it took me some time to realize the reason there were inconsistencies between my calculated values and SSURGO's was because a large portion of the SSURGO soil tables have NULL entries for a variety of rows. Additionally, SSURGO has their own way of area-weighted averaging horizon layers when different combinations of adjacent horizons are NULL – a piece of documentation I wish I came across at the outset of this project.

Due to time constraints, I was not able to consider IDW as an interpolation algorithm, but I plan on continuing work on this project into the summer.

Solutions

Although the datasets included in the SSURGO database are quite large (for my standards), I was able to perform most of my operations on a small study area (eg. Ottertail County), and extrapolate the code to the full dataset after I made sure the scripts were working. Anyone working with this database could query and analyze various subsets of data as well. Additionally, setting up indices for my table helped with querying speeds. While I was pursuing Research Question 1., I emailed somebody from the NRCS, and they passed along an ArcMap toolbox which serves the function of calculating map-unit available water storage at a user-specified depth interval. However, I believe the (blind) recreation of this algorithm, through a 400-line, commented-out, single-query SQL script ("*RS1_CustomSoilDepthAWS.sql*"; "Scripts" folder in project repository), may allow for viewers of this database to further understand the processes necessary for these kinds of interpolations. I believe these scripts also offer examples of multi-step, logic-based tasks using SQL, which some may think are only possible in Python. The code in this project repository provides an example of how CTEs may be 'daisy-chained' together in order accomplish complex tasks within a single query (all of my scripts were giant, single-query CTE strings, for better or worse). Similarly, the scripts in this project repository provide examples of how SQL "CASE" functions may be used as logic operators within aggregate functions. Although I did not have time to incorporate Inverse Distance Weighting in my assessment of interpolation algorithms, I am going attempt to incorporate SQL Stored Procedures as I work on this project into the summer – which I hope will add to the value of this database.

Appendix

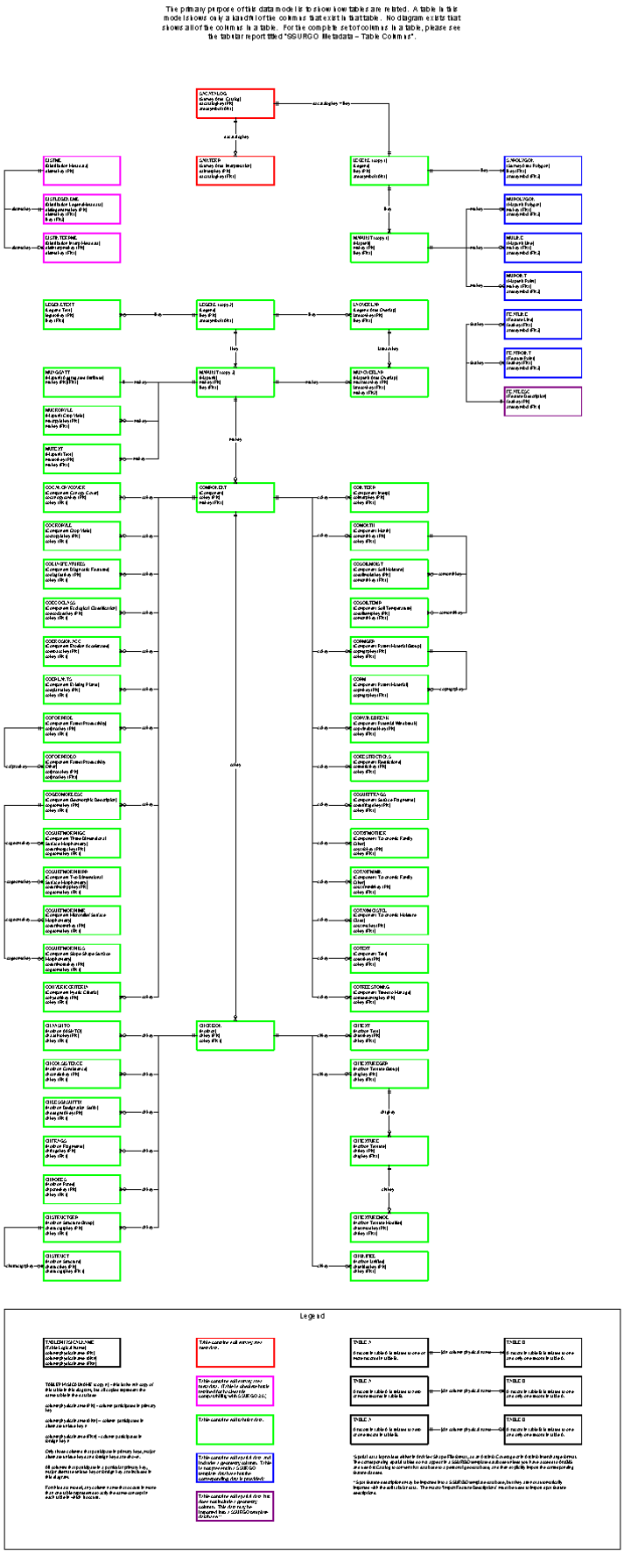


- MapUnit Polygon : Component = **1:Many** relationship
- Component : Horizon = **1:Many** relationship

Appendix A.1: *Model and illustration of how the SSURGO horizon, component, and map unit polygon tables relate to each other via one-to-many joins.*

SSURGO 2.2 Data Model – Diagram 1 of 2

Soil Tabular and Spatial Data Tables



Appendix B.1: *Full SSURGO Data Model.*
(view full-size image [here](#))

Appendix C.1: *Example of shp2pgsql command for loading a shapefile as a table (with geom attributes) into a SQL database.*

```
/Library/PostgreSQL/14/bin/shp2pgsql -s 4326
/Users/michaelfelzan/Desktop/LCCMR_Soils/mupolygon_copy.shp
public.mupolygoncopy | /Library/PostgreSQL/14/bin/psql -h
spatialdb.gisandbox.org -U felza001 -d felza001
```

Appendix C.2: *Code example for handling cases where horizons extend past soil depth interval bounding lines.*

```
working_thk_table AS
(
SELECT mupol_mukey,
            co_mukey,
            hz_cokey,
            chkey,
            avail_water_capacity,
            dist_to_hz_top,
            dist_to_hz_bottom,
            prcnt_comp_in_mapunit,
            fall_on_25_line,
            og_thk_r,
            thk_upto_25line_true,
            awc_notnull,
           -- if horizon doesnt fall on 25cm line, then use its original thickness.
           -- else: use the updated thickness calculated in this script.
       CASE WHEN fall_on_25_line = FALSE
            THEN og_thk_r ELSE thk_upto_25line_true
                END AS working_thk
FROM joined_true_false_fallonline
--only selecting horizons that are below or fall on the 25cm line:
WHERE dist_to_hz_bottom <= 25 OR fall_on_25_line
),
```

Appendix C.3: *Code example for the process of splitting parcel and map unit polygons into a mosaic of smaller polygons, based on the areas where the two polygon layers intersect. IDs from both layers are retained in the created "fragments" layer.*

```
WITH mapunit_parcel_intersect_fragments AS
(
--ID's for both parcel & map unit polygons:
SELECT p.gid, s.mukey,
       s.aws025wta, --available water storage of map unit @ 0-25cm depth
       s.aws050wta, --available water storage of map unit @ 0-50cm depth
       s.aws0100wta, --available water storage of map unit @ 0-100cm depth
       s.aws0150wta, --available water storage of map unit @ 0-150cm depth
       --splitting parcels & map units into smaller fragments by their borders:
       ST_Intersection(p.geom, s.geom) as splitgeom
FROM parcels_ott_4326 p,
     ottertail_w_muagg_aws s
WHERE ST_Intersects(p.geom, s.geom) --retain ID's of each (where they overlap)
```