Introduction to Spatial Databases and SQL

Dylan E. Beaudette

Natural Resources Conservation Service
U.S. Deptartment of Agriculture

—

California Soil Resource Lab (UC Davis)
http://casoilresource.lawr.ucdavis.edu/

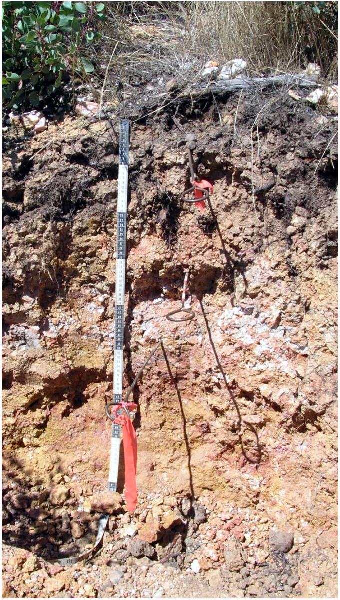
13 April, 2011

United States Department of Agriculture
NRCS Natural Resources Conservation Service
Soils

California Soil Resource Lab

Talk Outline

1. SQL Review
   - History
   - Basic Syntax
   - Table Operations
   - Joins
2. Spatial Databases: PostGIS
   - OGC Simple Features
   - GiST Spatial Index
   - PostGIS Metadata Tables
3. Spatial SQL
   - Points
   - Lines
   - Polygons
4. Data Import/Export
5. Examples
   - Soil Texture
   - Soil Water Storage



SQL[1] "Structured Query Language"

History

- developed by IBM in the '70s
- interactive vocabulary for database queries
- most modern systems are built on the 'SQL-92' standard

Modern Uses

- general purpose question *asking* vehicle
- SQL-based interfaces to many types of data: filesystem elements, GIS data, etc.
- often abstracted behind an interface of some kind: i.e. Google, etc.

[1] http://en.wikipedia.org/wiki/SQL

Flavors of SQL / Portability Issues

Many Vendors / Projects

- client/server: Oracle, MS SQL, Informix, IBM, MySQL, PostGreSQL
- file-based: Access, SQLite, BerkeleyDB

...but all support a subset of the SQL standards

Backwards Compatibility = Not Portable

- standard is vague on actual syntax
- complex & large standard → only subset implemented
- historic deviations from standard preserved

...in most cases the differences are slight

SQL Extensions

Why Bother?

The SQL language is great for simple set operations, but lacks many of the convenient functions found in other langauges. Extensions provide the ability to "call" external functions from other common programming languages, entirely within the context of the database.

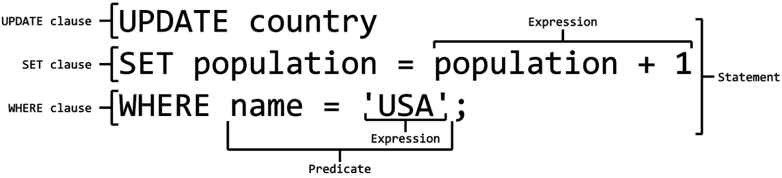
Some Examples

- "procedural SQL": PL/SQL, SQL PL, PGPLSQL, etc.
- SQL/XML: parsing of XML (extensible markup language documents)
- SQL/R: use of R language commands (numerical algorithms, statistics, etc.)
- SQL/Perl: use of perl language commands libraries (pattern mataching, etc.)
- SQL/Python: use of python language commands and libraries

SQL "Structured Query Language"

Syntax Notes

- set-based, declarative computer language
  i.e. a program that describes *what* to do, not *how* to do it
- 3-value logic: TRUE, FALSE, NULL
- several language elements:
  - statements: SQL code that has a persistent effect on tables, etc.
  - queries: SQL code that returns data
  - expressions: operations on or tests against a column's contents
  - clauses: logical 'chunks' of statements / queries



[2]

[2]image c/o wikipedia

SQL Syntax

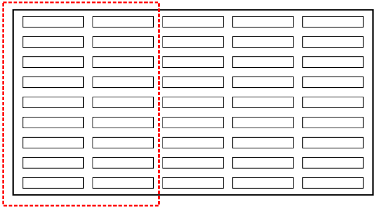
Syntax Notes

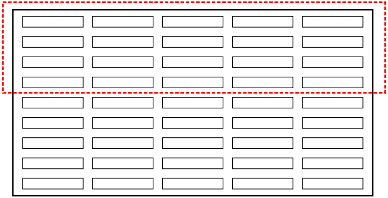
SELECT [columns]
  [**FROM** from_item]
  [**WHERE** condition]
  [**GROUP BY** expression]
  [**HAVING** condition]
  [{ **UNION** | **INTERSECT** | **EXCEPT** } **SELECT** [...]]
  [**ORDER BY** expression [**ASC** | **DESC**]]
  [**LIMIT count**] ;

Example Query

```
SELECT column_x, column_y, column_z
FROM table_x
WHERE column_x = 'something'
-- optional
GROUP BY column_x
ORDER BY column_x ; -- semi-colon denotes end of SQL statement
```

Operations on a Single Table





- filtering by column: SELECT a, b, c, ...
- filtering by row: WHERE
- ordering: ORDER BY
- aggregating: GROUP BY
- aggregating *then* filtering: GROUP BY, HAVING

INSERT/UPDATE/DELETE Statements

INSERT records into a table

```
INSERT INTO chorizon      -- table name
(cokey, hzname, hzdept_r, hzdepb_r)  -- record template
VALUES        -- SQL keyword 'here comes the data'
('new_cokey', 'Ap', 0, 10)      -- a new record
```

UPDATE existing records in a table

```
UPDATE chorizon       -- table to modify some records in
SET hzname = 'E'       -- update horizon names to modern conventions
WHERE hzname = 'A2' ;      -- but only the matching historic names
```

DELETE records FROM a table (be careful!)

```
DELETE FROM chorizon      -- table to delete records from
WHERE hzname IS NULL ;     -- records that are missing a horizon name
```

Table Modification Statements[3]

Altering Table Structure

```
-- add a new column
ALTER TABLE chorizon ADD COLUMN hydrophobicity_index integer;
-- now remove the column
ALTER TABLE chorizon DROP COLUMN hydrophobicity_index integer;
```

Altering Column Definitions

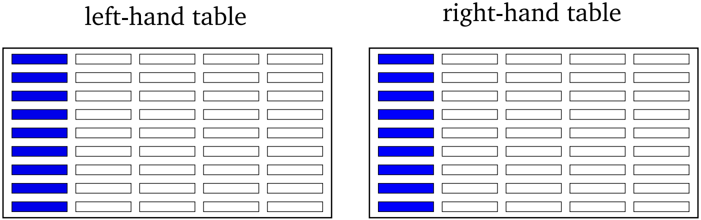
```
-- rename a column
ALTER TABLE chorizon RENAME COLUMN claytotal_r TO clay;
-- change the column's datatype (be careful!)
ALTER TABLE chorizon ALTER COLUMN clay TYPE numeric;
-- do not allow NULL values in a column
ALTER TABLE chorizon ALTER COLUMN clay SET NOT NULL;
-- do not allow values over 100%
ALTER TABLE chorizon ALTER COLUMN clay CHECK (clay <= 100) ;
```

[3] http://www.postgresql.org/docs/8.3/static/sql-altertable.html

Joining Tables

Generic pattern for joining 2 tables

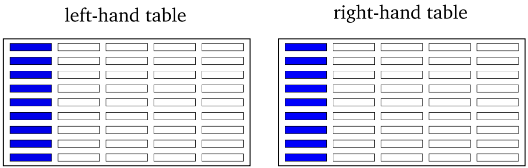
```
SELECT [left−hand_table.columns], [right−hand_table.columns]
FROM left−hand_table JOIN right−hand_table

−− rule for aligning data from each table
ON [join condition]

−− optionaly do more stuff after the join is complete
[WHERE clause]
[GROUP BY clause]
[ORDER BY clause] ;
```



Joins

Types of Joins

- Cartiesian Join: not generally useful, returns all permutation of input rows
- Inner Join: most commonly used, returns rows that occur in **both** tables
  - 1:1 $\rightarrow$ rows missing from either table ommitted
  - 1:many $\rightarrow$ rows in the left-hand table repeated
  - many:1 $\rightarrow$ rows in the right-hand table repeated (LU table)
- Left Outer Join: returns all records from the left-hand table, despite missing records in the right-hand table
  - 1:1,1:many,many:1 $\rightarrow$ rows missing from right-hand table padded with NULL
- Right Outer Join: same as left outer join, but reversed
  - 1:1,1:many,many:1 $\rightarrow$ rows missing from left-hand table padded with NULL



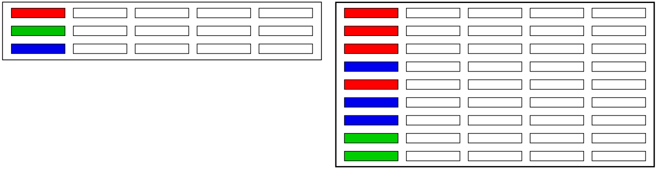
Inner Join

Join map unit data to component data (1:many)

```
SELECT substr(muname, 0, 30) as muname, mapunit.mukey, cokey,
compname, comppct_r
FROM mapunit JOIN component
ON mapunit.mukey = component.mukey
WHERE mapunit.mukey = '464463'
ORDER BY comppct_r DESC;
```

Results

```
           muname               | mukey  |     cokey     |    compname     | comppct_r
--------------------------------+--------+---------------+-----------------+-----------
 San Joaquin sandy loam, shall | 464443 | 464443:641360 | San Joaquin     |        85
 San Joaquin sandy loam, shall | 464443 | 464443:641362 | Exeter          |        14
 San Joaquin sandy loam, shall | 464443 | 464443:641361 | Unnamed, ponded |         1
```

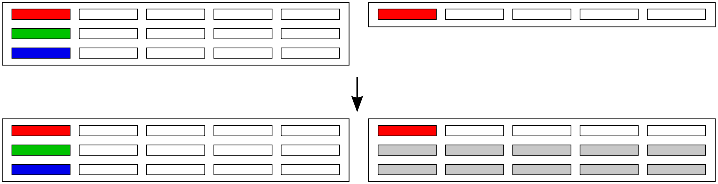


Left-Outer Join

Generate a listing of restrictive features for a single map unit

SELECT mukey, component.cokey, compname, comppct_r, reskind, reshard
FROM
component **LEFT JOIN** corestrictions
ON component.cokey = corestrictions.cokey
WHERE mukey = '464443'
ORDER BY comppct_r **DESC**;

```
 mukey  |     cokey     |     compname     | comppct_r | reskind |  reshard
--------+---------------+------------------+-----------+---------+-----------
 464443 | 464443:641360 | San Joaquin      |        85 | Duripan | Indurated
 464443 | 464443:641362 | Exeter           |        14 |         |
 464443 | 464443:641361 | Unnamed, ponded  |         1 |         |
```



Joins to Nested Sub-Queries

```
SELECT mukey, mu_area_frac, taxgrtgroup, hd.cokey as id, top, bottom, prop
FROM
(
    -- component weights, in the form of area fractions
    SELECT cd.mukey, cokey, taxgrtgroup, (comppct_r::numeric / 100.0) * mu_area as mu_area_frac
    FROM
    (
        -- component keys and percentages
        SELECT mukey, cokey, comppct_r, taxgrtgroup
        from component
        where areasymbol = 'ca654'
        and taxgrtgroup is not NULL
        ) as cd
    JOIN
    (
        -- map unit areas by mukey
        SELECT mukey, sum(ST_Area(wkb_geometry)) as mu_area
        from mapunit_poly
        where areasymbol = 'ca654'
        group by mukey
        ) as mu_areas
    on cd.mukey = mu_areas.mukey
) as comp_wts
-- regular join will throw out all components without horizon data
JOIN
(
    -- horizon level data
    SELECT cokey, hzdept_r as top, hzdepb_r as bottom, claytotal_r as prop
    from chorizon
    where om_r is not null
    and areasymbol = 'ca654'
) as hd
on comp_wts.cokey = hd.cokey
ORDER BY mukey, id, top ASC;
```

What is PostGIS?

PostgreSQL

- Relational Database Management System (RDBMS)
- Scaleable to *n* processors, across *m* computers
- Support for very large data types and tables
- Open Source

PostGIS

- Spatial Extension to PostgreSQL
- Based on C library functions
- OGC Simple Feature Model
- Open Source (compare with $60K+/CPU for Oracle Spatial)

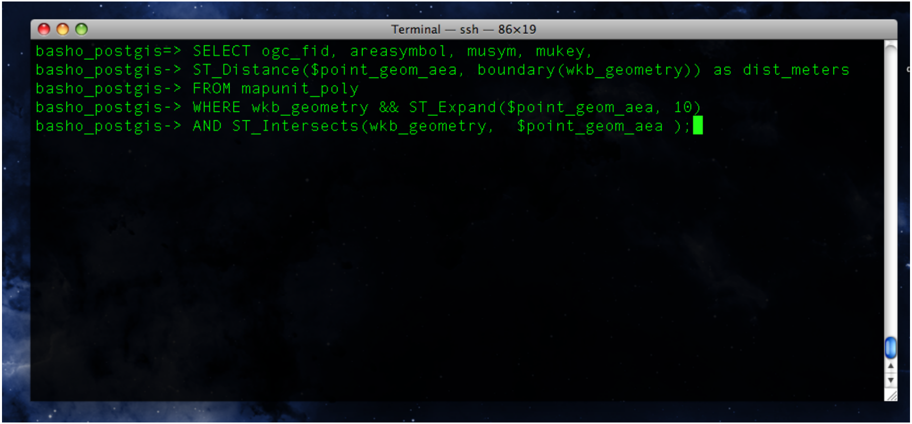
Why Should I Use PostGIS?

1. Scales well with massive datasets / file system objects
2. Familliar SQL-based manipulation of attribute and spatial features
3. Repeatable, transparent work-flow

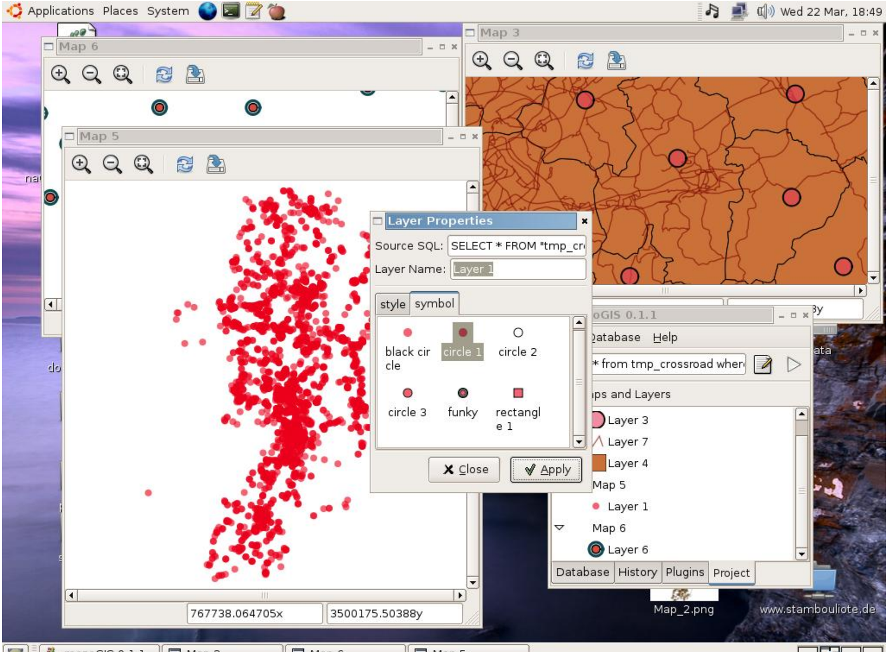
```
CREATE TABLE dwr.mapunit_dau as
SELECT m_polys.areasymbol, m_polys.mukey, d_polys.dau_id,
ST_Intersection(d_polys.wkb_geometry, m_polys.wkb_geometry) as wkb_geometry
FROM
    (
    -- subset map unit polygons to certain survey areas
    -- 6.540 s
    SELECT wkb_geometry, areasymbol, mukey
    FROM mapunit_poly
    -- results in 21682 map unit polygons
    WHERE areasymbol in ('ca653','ca654', 'ca113')
    ) as m_polys
JOIN
    (
    -- subset DAU polygons that overlap with specific survey areas
    --  512 ms
    SELECT dwr.dau.wkb_geometry, dau.dau3_d_id as dau_id
    FROM dwr.dau JOIN mapunit_bound_poly
    -- results in 58 DAU polygons
    ON mapunit_bound_poly.areasymbol in ('ca653','ca654', 'ca113')
    and dwr.dau.wkb_geometry && mapunit_bound_poly.wkb_geometry
    ) as d_polys
-- join condition: only those polygons which completely intersect
ON ST_Intersects(d_polys.wkb_geometry, m_polys.wkb_geometry);
```

What Does PostGIS Look Like?

`psql` command-line client



What Does PostGIS Look Like?



OGC Simple Feature Model

The Standard

- "Simple Features" - not a topological format (i.e. GRASS or Coverage)
- 2D and 3D geometric primitives, without self-intersection
- http://www.opengeospatial.org/standards/sfs

Basic Geometries

- POINT $(x\ y)$
- LINESTRING $(x_1\ y_1,\ x_2\ y_2,\ x_3\ y_3,\ \ldots,\ x_n\ y_n)$
- POLYGON $(x_1\ y_1,\ x_2\ y_2,\ x_3\ y_3,\ \ldots,\ x_{n-1}\ y_{n-1},\ x_1\ y_1)$;

"Multi-" Geometries

- MULTIPOINT $((POINT_1),\ \ldots,\ (POINT_n)\ )$
- MULTILINESTRING $((LINESTRING_1),\ \ldots,\ (LINESTRING_n)\ )$
- MULTIPOLYGON $((POLYGON_1),\ \ldots,\ (POLYGON_n)\ )$;

SQL Review

ANSI SQL Examples

- SELECTion **SELECT** a **from** b **where** c = d
- sorting **SELECT** a **from** b **order by** a **desc**
- join **SELECT** t1.a, t2.b **from** t1 **join** t2 **on** ...
- aggregation **SELECT** **sum**(a)**from** b **group by** a

OGC "Spatial" SQL Examples

- feature extraction **SELECT** ST_X(point_geom), ST_Y(point_geom)**from** ...
- feature extraction **SELECT** PointN(geom)**from** ...
- spatial join **SELECT** * **from** t1 **join** t2 **on** ST_Distance(t1.geom, t2.geom)< 100 ...
- feature manipulation **SELECT** ST_Transform(geom, SRID)**from** ...
- feature analysis **SELECT** ST_Buffer(geom, distance)**from** ...
- GIS overlay functions **SELECT** ST_Intersection(geom_1, geom_2)**from** ...

Spatial SQL: Point Geometry

Construction

- **SELECT** ST_MakePoint(x,y)
- **SELECT** ST_Centroid(polygon_geom)
- **SELECT** ST_PointOnSurface(polygon_geom)

Measurement

- **SELECT** ST_X(geom), ST_Y(geom)
- **SELECT** ST_Distance(geom1, geom2)
- **SELECT** ST_Distance_Sphere(geom1, geom2)

Geometric Operation

- **SELECT** ST_Buffer(geom, distance)
- **SELECT** ST_Expand(geom, distance)
- **SELECT** ST_Touches(geom1, geom2)

Spatial SQL: Line Geometry

Construction

- **SELECT** ST_MakeLine(geometry collection)
- **SELECT** ST_MakeLine(geom1, geom2)

Measurement

- **SELECT** ST_Length(geom)
- **SELECT** ST_Length_Spheroid(geom, spheroid)
- **SELECT** ST_Length3d(geom)

Geometric Operation

- **SELECT** ST_Crosses(geom1, geom2)
- **SELECT** ST_Overlaps(geom, distance)
- **SELECT** ST_Intersection(geom1, geom2)

Spatial SQL: Polygon Geometry

Construction

- **SELECT** ST_ConvexHull(geometry collection)
- **SELECT** ST_BuildArea(line_geom)

Measurement

- **SELECT** ST_Area(geom)
- **SELECT** ST_Perimeter(geom)

Geometric Operation

- **SELECT** ST_Intersection(geom, geom)
- **SELECT** ST_Intersects(geom, geom)
- **SELECT** ST_Contains(geom1, geom2)

Getting Data into and out of PostGIS

OGR tools: recall order of OGR data sources: *output input*

- Import

```
ogr2ogr -f "PostgreSQL" \
PG:'dbname=ssurgo_combined user=xxxx password=xxxx host=host' input_file.shp
```

- Export

```
ogr2ogr output_file.shp \
PG:'dbname=ssurgo_combined user=xxxx password=xxxx host=postgis.server.edu' tablename
```

→ Note the OGR syntax for specifying a PostGIS DSN.

PostGIS Loader/Dumper

- Import

```
shp2pgsql -s SRID -c -g wkb_geometry -I shapefile.shp schema.table \
| psql -U username -h host database
```

- Export

```
pgsql2shp -f shapefile.shp -h host -u username -P password -k -g wkb_geometry \
database schema.table
```

→ See the manual page for pgsql2shp for a complete list of arguments.

Getting Data into and out of PostGIS

Text Files

- CSV format, from within the psql client

  \copy tablename TO 'filename.csv'CSV HEADER

- CSV format, via psql client

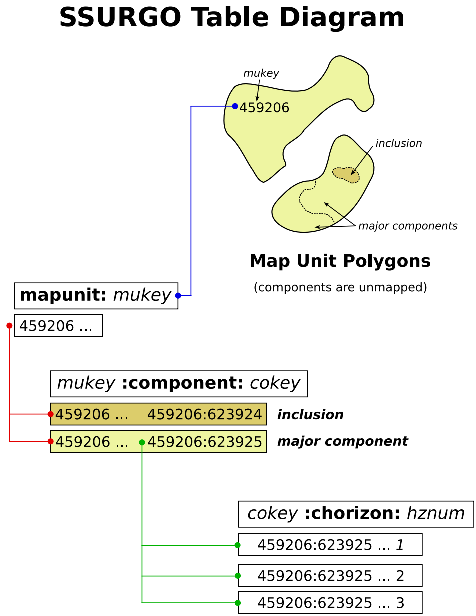
  echo "select column_list from table_list "| psql --tuples --no-align -F "," database > file.csv

- Tabular data to HTML format, via psql client See output below:

  echo "select column_list from table_list "| psql --html database > file.html

```
area compname
132472.230854819 Hilmar variant
322819.967391312 Oneil
362729.418301135 Carranza
431948.171760353 Tuff rockland
448784.927049035 Gravel pits
500763.225267798 Snelling variant
518860.954990617 Foster
571640.132661382 Alamo
648973.748756059 Toomes
924327.631201791 Dumps
```

SSURGO Database Structure



Aggregation of SSURGO Geometry

Query: extract SSURGO geom. from arbitrary bbox, compute area weights

```
-- define a transformed bounding box for later use
\set bbox ST_Transform(
ST_SetSRID(
ST_MakeBox2D( ST_MakePoint(-122.25, 39.28), ST_MakePoint(-122.20, 39.30) )
, 4326)
, 9001)

-- select map unit keys, map unit symbols, and computed areas for the intersecting polygons
SELECT mukey,
sum(ST_Area(ST_Intersection(wkb_geometry, :bbox))) / ST_Area(:bbox) as mu_area_wt
from mapunit_poly
WHERE ST_Intersects(wkb_geometry, :bbox)
GROUP BY mukey;
```

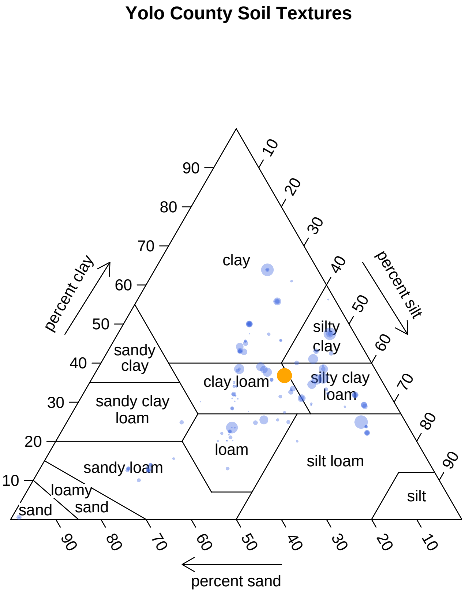
```
 mukey  |     mu_area_wt
--------+-------------------
 461544 |  0.562595368617999
 461571 |  0.347993963186697
 461595 | 0.0748614412770969
 461667 | 0.0145492269180839
```

Soil Texture Example

compute several weighted means of sand, silt, clay

```
-- join with polygons, and compute areas weights
SELECT mapunit_poly.mukey,
sum(ST_Area(wkb_geometry)) /
(SELECT ST_Area(wkb_geometry) FROM mapunit_bound_poly WHERE areasymbol = 'ca113') AS area_wt,
max(sand) as sand, max(silt) as silt, max(clay) as clay   -- fake aggregate functions applied
FROM
mapunit_poly
JOIN
        (
        -- compute component percent weighted mean
        SELECT mukey,
        sum(comppct_r * sand) / sum(comppct_r) AS sand,
        sum(comppct_r * silt) / sum(comppct_r) AS silt,
        sum(comppct_r * clay) / sum(comppct_r) AS clay
        FROM
        component
        JOIN
                (
                -- compute hz thickness weighted mean
                SELECT cokey,
                sum((hzdepb_r - hzdept_r) * sandtotal_r) / sum(hzdepb_r - hzdept_r) AS sand,
                sum((hzdepb_r - hzdept_r) * silttotal_r) / sum(hzdepb_r - hzdept_r) AS silt,
                sum((hzdepb_r - hzdept_r) * claytotal_r) / sum(hzdepb_r - hzdept_r) AS clay
                FROM chorizon
                GROUP BY cokey
                ) AS co_agg
        ON component.cokey = co_agg.cokey
        GROUP BY component.mukey
        ) AS mu_agg
ON mapunit_poly.mukey = mu_agg.mukey
GROUP BY mapunit_poly.mukey;
```

Soil Texture Example (cont.)



Soil Texture Example (cont.)

```
# load some libs:
library(plotrix)

# read in the data
x <- read.csv('yolo_texture.csv')

# simple soil texture, with symbol size weighted by area weight
soil.texture(x[,3:5], cex=sqrt(50*x$area_wt), pch=16, col.symbol=rgb(65,105,225, alpha=100, max=255),
show.lines=T, show.names=T, col.lines='black', col.names='black', main='Yolo County Soil Textures')

triax.points(cbind(weighted.mean(x$sand, x$area_wt), weighted.mean(x$silt, x$area_wt), weighted.mean(x
        $clay, x$area_wt)),
col.symbols='orange', pch=16, cex=2)
```

Soil Water Storage Computation

```
SELECT mukey, compname, comppct_r, a.* FROM component
JOIN
        (
        SELECT cokey, sum( (hzdepb_r − hzdept_r) * awc_r) AS component_whc, sum((hzdepb_r −
              hzdept_r)) AS depth
        FROM chorizon WHERE areasymbol = 'ca113'
        GROUP BY cokey
        ) AS a
ON component.cokey = a.cokey
WHERE component.areasymbol = 'ca113'
ORDER BY mukey ;
```

```
 mukey  |      compname       | comppct_r |      cokey       | component_whc | depth
--------+---------------------+-----------+------------------+---------------+-------
 459204 | Gravel pits         |       100 | 459204:659832    |          3.04 |   152
 459206 | Arbuckle            |        70 | 459206:623924    |         16.46 |   152
 459206 | Arbuckle            |        15 | 459206:1128332   |         16.46 |   152
 459207 | Arbuckle            |        85 | 459207:623932    |         16.46 |   152
 459208 | Balcom              |        85 | 459208:623933    |          9.69 |    61
 459209 | Balcom              |        85 | 459209:623937    |         17.25 |   104
 459210 | Balcom              |        85 | 459210:623942    |         17.25 |   104
 459211 | Balcom              |        85 | 459211:623949    |          9.44 |    61
 459212 | Balcom              |        45 | 459212:623950    |         13.93 |    86
```

Soil Water Storage Computation

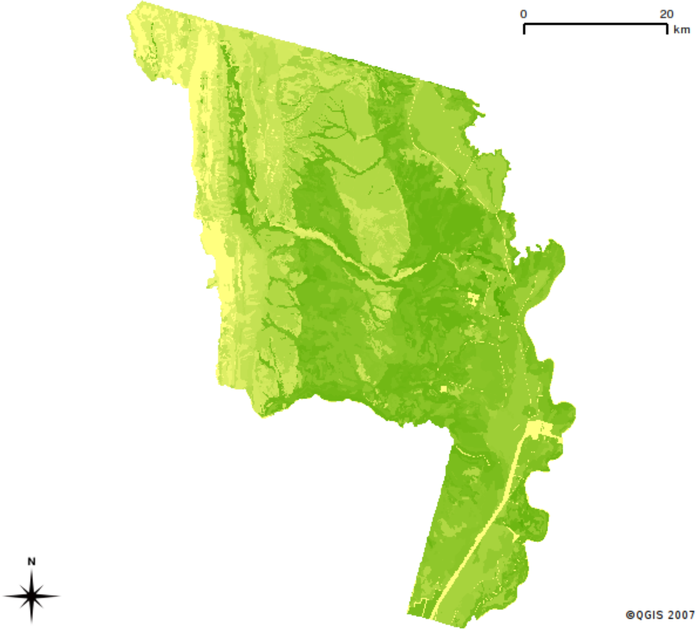
```
SELECT mukey,
-- note that weights from non-soil components must be removed
-- otherwise, weighted mean values will be too low
SUM(comppct_r * component_whc) / SUM(comppct_r) AS comppct_weighted_whc_cm
FROM component
JOIN
        (
        SELECT cokey, sum( (hzdepb_r - hzdept_r) * awc_r) AS component_whc,
        sum((hzdepb_r - hzdept_r)) AS depth
        FROM chorizon
        WHERE areasymbol = 'ca113'
        GROUP BY cokey
        ) AS a
USING (cokey)
WHERE component.areasymbol = 'ca113'
-- filter out components that are missing soils data
AND a.component_whc IS NOT NULL
GROUP BY mukey ;
```

```
mukey | comppct_weighted_whc_cm
-------+------------------------
459225 | 10
459226 | 10
459227 | 11
...
```

Soil Water Storage Computation

```
-- create the new table with both geometry and attributes
CREATE TABLE yolo_whc AS
SELECT ogc_fid, wkb_geometry AS wkb_geometry, b.mukey, b.comppct_weighted_whc_cm
FROM mapunit_poly
-- use LEFT JOIN to include non-soil polygons in the result set
-- alternatively use JOIN to ignore non-soil polygons
LEFT JOIN
        (
        SELECT mukey,
        -- note that weights from non-soil components must be removed
        -- otherwise, weighted mean values will be too low
        SUM(comppct_r * component_whc) / SUM(comppct_r) AS comppct_weighted_whc_cm
        FROM component
        JOIN
          (
          SELECT cokey, sum( (hzdepb_r - hzdept_r) * awc_r) AS component_whc,
          sum((hzdepb_r - hzdept_r)) AS depth
          FROM chorizon
          WHERE areasymbol = 'ca113'
          GROUP BY cokey
          ) AS a
        USING (cokey)
        WHERE component.areasymbol = 'ca113'
        -- filter out components that are missing soils data
        AND a.component_whc IS NOT NULL
        GROUP BY mukey
        ) AS b
-- JOIN constraint
USING (mukey)
-- optional constraint to limit geometry search in mapunit_poly table
WHERE mapunit_poly.areasymbol = 'ca113' ;

Create indexes and register the new geometry:
-- create attribute and spatial index:
CREATE UNIQUE INDEX yolo_whc_idx ON yolo_whc (ogc_fid) ;
CREATE INDEX yolo_whc_spatial_idx ON yolo_whc USING gist (wkb_geometry gist_geometry_ops);

-- register in geometry_columns table:
INSERT INTO geometry_columns VALUES ('','public','yolo_whc','wkb_geometry',2,9001,'POLYGON');
```

Soil Water Storage Computation



Additional Examples

1. simetaw work
2. soilweb
3. 1km gridded soils data
4. PostGIS In Action Book
5. interactive examples
6. HRCLIM data: maybe for friday?