Learning the FOSS4g Stack: Spatial SQL with Postgres

(aka Spatial is Not Special
Spatial SQL: A language for Geographers)

USING SQL TO LEVERAGE GEOGRAPHIC INFORMATION

gisadvisor.com, LLC.

What we'll cover

PART 1: LOADING SOFTWARE AND DATA
PART 2: OVERVIEW OF SQL
PART 3: SQL DATA TYPES
PART 4: TRADITIONAL SQL
PART 5: SPATIAL SQL FOR VECTOR GEOMETRY
PART 6: SPATIAL SQL FOR GEOGRAPHIC ANALYSIS

gisadvisor.com, LLC.

Course Goals

At the end of this course, you will:

- Understand what SQL is.
- Learn how to query databases using SQL.
- Learn how to leverage spatial constructs using spatial SQL.
- Develop GIS based applications with SQL on the back end.

gisadvisor.com, LLC.

Part I

LOADING SOFTWARE AND DATA

gisadvisor.com, LLC.

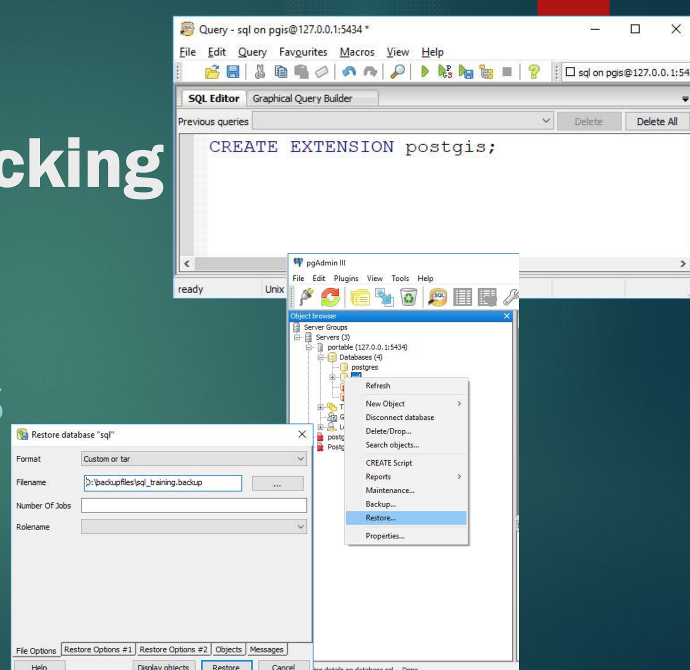
Software Installation

POSTGRES
POSTGIS
QGIS

gisadvisor.com, LLC.

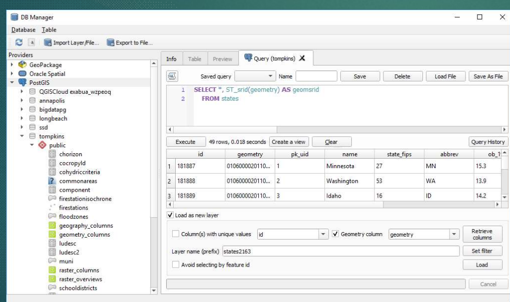
Loading and checking

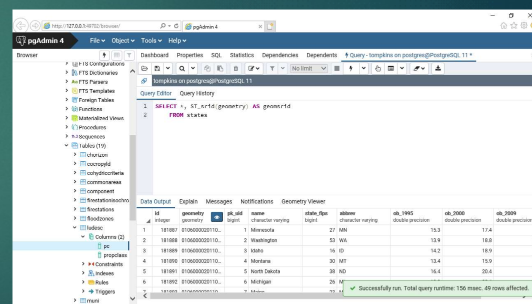
- RESTORE BACKUP FILE
- TEST CONNECTIONS



gisadvisor.com, LLC.

Interacting with Data: QGIS and PGAdmin





gisadvisor.com, LLC.

Part 2: Overview

WHY THIS COURSE
WHAT IS SQL
DATA AND SCENARIOS

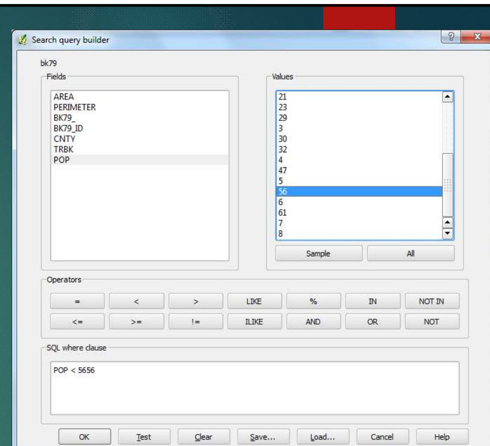
gisadvisor.com, LLC.

We are all special

- Sports
- Scouts
- Spelling
- Why not Spatial?
  - Special software
  - Special wizards
  - Special formats
  - Special tools
- Why not "strings are special"?
- Why not "integers are special"?

gisadvisor.com, LLC.

Why this course

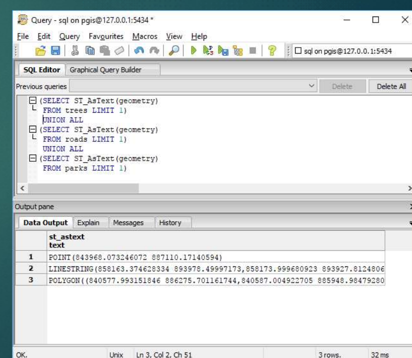


- **SQL is perceived as hard**
  - SQL is easy
  - SQL is powerful
- **Query wizards are *dumbed down***
  - We are raising a generation of GIS novices
  - Query wizards limit our ability of what we can really do with GIS

gisadvisor.com, LLC.

What if Spatial was just another data type?

- Just data in a table in the form of a *geometry*
  - If I can add two numbers, I can intersect two geometries
  - If I can trim a string of a few choice characters, I can clip a geometry with another geometry
  - If I can find a date between two other dates, I can find a geometry covered by another geometry



gisadvisor.com, LLC.

The case against SQL

*[using wizards].....this means just about any GIS professional can manipulate the model, and understand the workflow - without requiring a **highly paid, specialised consultant** to come in and tweak **obscure SQL statements**. Oracle Spatial, and the SQL that come with it, are not 'free' - there is **a huge learning curve**, and most of **it is not intuitive**.*

- quote from Directions Magazine thread

gisadvisor.com, LLC.

SQL is Obscure?

- Structured Query Language (SQL) is the most popular computer language used to create, modify and query databases.
- SQL is an industry standard database query language used to fetch records from tables and to present those records with the fields desired.
- SQL was adopted as a standard by ANSI (American National Standards Institute) in 1986 and ISO (International Organization for Standardization) in 1987.

gisadvisor.com, LLC.

SQL is not intuitive?

- SQL is designed for a specific, limited purpose — querying data contained in a relational database. As such, it is a set-based, declarative computer language rather than an imperative language such as C or BASIC which, being programming languages, are designed to solve a much broader set of problems.
  - A declarative computer language / high-level programming language is a programming language that is more user-friendly, to some extent platform-independent, and abstract from low-level computer processor operations such as memory accesses.

gisadvisor.com, LLC.

Conclusion: What is SQL

- SQL is the standard language for relational database management systems.
- A recognized standard in computing.
- A specialized defacto language for updating, deleting, and requesting information from databases.
- The SQL ecosystem dwarfs the GIS ecosystem.

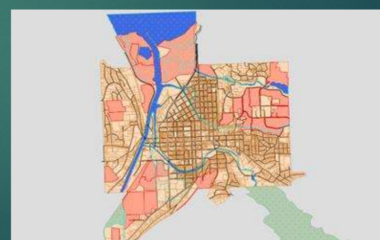
We need specialized consultants?

- Server applications
  - Oracle, SQLServer, MySQL, PostGRES, SYBASE, others
- Personal SQL
  - SQLite, Microsoft Access
- Spatial plug-ins for SQL
  - Oracle spatial, SQLServer spatial, PostGIS, Spatialite, others

gisadvisor.com, LLC.

Lets talk about our data

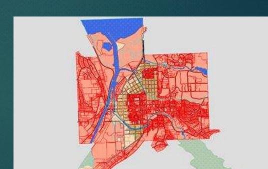
- Regions
- General Themes
- Layers
- Attributes



gisadvisor.com, LLC.

Basic SQL to whet your appetite

```
SELECT * FROM tcparcel
--
SELECT propclass
FROM tcparcel
WHERE acres > 44

--
SELECT tcparcel.*
FROM tcparcel, floodzones
WHERE st_intersects(tcparcel.geom,floodzones.geom)
AND floodzones.zone = 'AE'
```



Another one to talk about

```
SELECT tcparcel.*
FROM tcparcel, floodzones, firestations
WHERE
 st_intersects(tcparcel.geom,floodzones.geom)
AND floodzones.zone = 'AE'
AND tcparcel.asmt > 500000
AND
 st_distance(tcparcel.geom,firestations.geom) <
 6000
```

Part 3

SQL DATA TYPES

gisadvisor.com, LLC.

SQL Data Types

- SQL databases support multiple types of data formats:
  - Numeric (Single, Double, Floating Point)
  - Boolean (True/False)
  - Character (fixed length, varying length
  - Binary
  - Date and Time
  - Geometry (OGC, vendor specific)
  - Monetary

Numeric Operations

Numeric operations allow users to perform math based functions like addition, subtraction, division and multiplication. Additionally, numeric operations allow users to perform statistical and other mathematical functions you would find in a spreadsheet.

Numeric Operations

```
SELECT asmt - land AS STRUCTVALUE, parcelkey
FROM tcparcel

SELECT asmt / land AS STRUCTVALUE, parcelkey
FROM tcparcel
WHERE land > 0

SELECT avg(asmt)
FROM tcparcel

SELECT avg(asmt) AS AVGASMT, stddev(asmt) AS SDASMT,
     sum(asmt) AS SUMASMT, stddev(asmt)/avg(asmt) AS CV
FROM tcparcel
```

Questions

- What is the average acreage for parcels?
- What is the average acreage for single family residential parcels (propclass = 210)?
- What is the total area for the ‘X’ zone (zone) in the floodzone layer?

gisadvisor.com, LLC.

Boolean and Logical Operations

Boolean operations allow users to perform logic based requests that return true and false and comparisons.

The usual logical operators are available:

AND
OR
NOT

SQL uses a three-valued logic system with true, false, and null.

a	*b*	*a* AND *b*	*a* OR *b*
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
=	equal
<> or !=	not equal

```
SELECT *
FROM parcels
WHERE acres > 3

SELECT *
FROM parcels
WHERE acres > 3
AND propclass = 210
```

Character Operations

String operations allow users to perform text based functions like string concatenation, trimming text, case conversion, etc.

Character Operations

```
SELECT *
FROM tcparcel
WHERE Left(location,2) = 'BU'

SELECT * FROM tcparcel
WHERE lower(location) = 'buffalo st e'

SELECT concat(loc, ' ', location)
FROM tcparcel

SELECT asmt, asmt::text
FROM tcparcel
SELECT asmt::text::numeric
FROM tcparcel
```

Date and Time Operations

Date operations allow users to perform date based functions like adding dates, determining the difference in dates, determining day of the week.

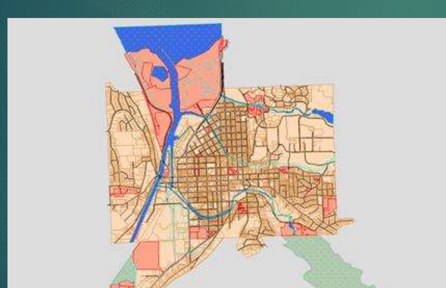
Time operations allow users to perform time based functions like calculating minutes, seconds, and hours.

Date and Time Operations

```
SELECT date_part('day',sale_date)
FROM tcparcel

SELECT *
FROM tcparcel WHERE sale_date >
'6/03/2001'

SELECT sale_date - '6/03/2001'
FROM tcparcel

SELECT extract(month from sale_date),
parcelkey
FROM tcparcel
```

Spatial Operations

Spatial operations allow users to perform spatial based functions like buffer, containment, intersection, and distance. In this case, spatial data is just another data type, like integers or text strings.



Spatial Operations

```
SELECT tcparcel.*
FROM tcparcel, schooldistricts
WHERE
ST_Intersects(tcparcel.geom,schooldistricts.geom)
AND schooldistricts.district = 'Lansing'

SELECT ST_Intersection(tcparcel.geom,floodzones.geom)
      AS geometry
FROM tcparcel, floodzones
WHERE floodzones.zone = 'AE'
AND st_intersects(tcparcel.geom,floodzones.geom)
```

Part 4

TRADITIONAL SQL

gisadvisor.com, LLC.

SELECT

```
SELECT tcparcel.*
FROM tcparcel
WHERE pc = '210'

SELECT *
FROM tcbuildings, tcmuni
WHERE municipali = tcmuni.fullname

SELECT tcparcel.geom, tcparcel.swis, ludesc.propclass
FROM tcparcel, ludesc
WHERE tcparcel.pc  = ludesc.pc::text

SELECT tcparcel.geom, tcparcel.swis, ludesc.propclass
FROM tcparcel, ludesc
WHERE Left(tcparcel.pc,1) =  Left(ludesc.pc::text,1)
AND Right(ludesc.pc::text,2) = '00'
```

ALL, ANY, LIMIT, SOME, OFFSET Quantifiers

Determine if the value of an expression is equal to any or all values in a specified list.

```
SELECT * FROM states
WHERE ob_2009 > ALL
  (SELECT ob_2000 FROM States)

SELECT *
FROM states
ORDER by ob_2009 desc
Limit 5

SELECT *
FROM states
ORDER by ob_2009 desc
Offset 5
Limit 5
```

ORDER BY

- Sorts records using specified criteria in ascending or descending order

```
SELECT * FROM tcparcel
ORDER BY asmt ASC

SELECT * FROM tcparcel
ORDER BY asmt DESC
```

gisadvisor.com, LLC.

BETWEEN Operator

- Determines whether the value of an expression falls within a specified range of values.

```
SELECT * FROM tcparcel
WHERE asmt
BETWEEN 100000 AND 200000
```

gisadvisor.com, LLC.

IN Operator

- Determines whether the value of an expression is equal to any of several values in a specified list.

```
SELECT * FROM tcparcel
  WHERE  propclass IN
    (SELECT propclass FROM ludesc

     WHERE pc > 500 )
```

gisadvisor.com, LLC.

SQL Aggregate Functions

- An aggregate function is a function that calculates the total value of a group of values, and includes mathematical operations like:
  - avg,
  - count,
  - min,
  - max,
  - Stdev
  - Stdevp
  - sum, var, varp

gisadvisor.com, LLC.

SQL Aggregate Functions

```
SELECT stddev(ob_2009) AS stdev,
avg(ob_2009) AS avg
FROM states
-----------------
SELECT '2009' AS yr, stddev(ob_2009)/
avg(ob_2009) AS CV
FROM states

Union All

SELECT '2000' AS yr, stddev(ob_2000)/
avg(ob_2000) AS CV
FROM states
```

gisadvisor.com, LLC.

GROUP BY

- GROUP BY statements perform aggregate computations on data, and groups the computations by one or more columns
- Aggregate computations include summation, average, count, min, max, etc.
- A single SQL statement with GROUP BY can accomplish a task that might require a couple of pages of programming code.

gisadvisor.com, LLC.

GROUP BY

```
SELECT sum(acres) AS sumacres, propclass
FROM tcparcel
GROUP BY propclass

SELECT sum(acres) AS sumacres, Left(propclass::text,1)
FROM tcparcel
GROUP BY left(propclass::text,1)

SELECT count(*) AS NumProps, propclass
FROM tcparcel
GROUP BY propclass

SELECT  sum(asmt) sumassmt, pc::numeric, zone
FROM tcparcel, floodzones
WHERE st_intersects(tcparcel.geom, floodzones.geom)
AND pc::numeric BETWEEN 200 and 299
AND pc <> ' '
GROUP BY pc, zone
ORDER BY pc, zone
```

GROUP BY FUN

```
SELECT tcparcel.swis, ludesc.propclass
FROM tcparcel, ludesc
WHERE Left(tcparcel.pc,1) =
Left(ludesc.pc::text,1)
AND Right(ludesc.pc::text,2) = '00'
-----
SELECT st_union(tcparcel.geom) AS geom,
propclass
FROM tcparcel
GROUP BY propclass
```

gisadvisor.com, LLC.

Dissolve?

```
SELECT ST_Union(a.geom) as geom,
a.propclass
FROM tcparcel AS a, tcparcel AS b
WHERE ST_touches(a.geom,b.geom)
AND a.propclass = b.propclass
AND a.parcelkey <> b.parcelkey
GROUP BY a.propclass
```

gisadvisor.com, LLC.

Pull it all together

```
SELECT avg(asmt) AS avgasmt, sum(asmt) as sumasmt,
propclass
FROM tcparcel, floodzones
WHERE asmt BETWEEN 100000 AND 200000
AND ST_Intersects(tcparcel.geom, floodzones.geom)
AND tcparcel.propclass in ('Residential',
'Commercial', 'Vacant')
GROUP BY propclass
ORDER by propclass ASC
```

gisadvisor.com, LLC.

Conditional Statements

```
SELECT propclass, asmt,
      CASE WHEN propclass = 'Residential' THEN asmt * 1.04
           WHEN propclass = 'Commercial' THEN asmt * 1.6
           WHEN propclass = 'Vacant' THEN asmt * .9
           WHEN propclass = 'Agriculture' THEN asmt * .9
           ELSE asmt * 1
     END AS newassmt
FROM tcparcel
```

gisadvisor.com, LLC.

Conditional Statements - spatial

```
SELECT propclass, asmt,
     CASE WHEN ST_Intersects(geom, g) AND zone = 'AE' THEN 'uh oh'
          WHEN ST_Intersects(geom, g) AND zone = 'X500' THEN 'eh'
          WHEN ST_Intersects(geom, g) AND zone ='X' THEN 'ok'
          ELSE 'aok'
     END AS risk
FROM tcparcel, (SELECT geom AS g, zone FROM floodzones) AS t1
```

gisadvisor.com, LLC.

Conditional Statements – updating

```
UPDATE test
SET asmt = CASE
               WHEN propclass = 'Residential' THEN asmt * 1.05
               WHEN propclass = 'Commercial'  THEN asmt * 1.1
               WHEN propclass = 'Vacant'      THEN asmt * .9
               ELSE asmt * 1
          END
```

gisadvisor.com, LLC.

Conditional Statements – updating II

```
UPDATE test
SET swis = CASE
               WHEN zone = 'AE' THEN 'uh oh'
               WHEN zone = 'X' THEN 'ok'
               WHEN zone = 'X500' THEN 'eh'
               ELSE 'na'
          END
FROM floodzones
WHERE ST_Intersects(test.geom, floodzones.geom);
```

gisadvisor.com, LLC.

Conditional Statements – updating III

```
SELECT tcparcel.swis, zone,
           CASE
               WHEN zone = 'AE' THEN 'uh oh'
               WHEN zone = 'X' THEN 'ok'
               WHEN zone = 'X500' THEN 'eh'
               ELSE 'na'
          END AS risk
FROM floodzones, tcparcel
WHERE ST_Intersects(tcparcel.geom, floodzones.geom);
```

gisadvisor.com, LLC.

Part 5 - The Spatial Stuff

SIMPLE EXAMPLES TO ILLUSTRATE SQL CONSTRUCTS WITHIN A SPATIAL PARADIGM

REMEMBER, GEOMETRY IS JUST ANOTHER DATA TYPE

HTTPS://POSTGIS.NET/DOCS/MANUAL-2.5/

HTTPS://POSTGIS.NET/DOCS/MANUAL-2.5/REFERENCE.HTML#OPERATORS

gisadvisor.com, LLC.

Let’s talk about indexes

- What are indexes?
- What are the results of using an index?

gisadvisor.com, LLC.

Coordinate Systems

- Find a projection

```
SELECT ST_SRID(geom) FROM states
```

- Define a projection

```
SELECT UpdateGeometrySRID('states','geom',2796)
```

- Change a projection

```
SELECT ST_Transform(geom,3450) FROM states;
SELECT ST_Transform(geom,2796) FROM states;
```

gisadvisor.com, LLC.

Adjacent

```
SELECT * FROM tcparcel
WHERE ST_touches(tcparcel.geom,(SELECT geom
  FROM tcparcel
      WHERE parcelkey = '50070006200000010150000000'))
(now get the sum of the land values)

SELECT SUM(asmt) FROM tcparcel
WHERE ST_touches(tcparcel.geom,
  (SELECT geom FROM tcparcel
   WHERE parcelkey = '50070006200000010150000000'))
  )
                          )
```

gisadvisor.com, LLC.

Buffer

```
SELECT *, ST_Buffer(geom,300) As Buffered
FROM tcparcel AS D
WHERE parcelkey =
  '50070006200000010150000000'

Now do it for ASMT > 300000
```

gisadvisor.com, LLC.

Contains / Touches

```
SELECT tcparcel.*
FROM tcparcel,floodzones
WHERE ST_Contains(floodzones.geom,tcparcel.geom)
AND floodzones.zone = 'AE'
--------------
SELECT tcparcel.*
FROM tcparcel,floodzones
WHERE ST_Touches(floodzones.geom,tcparcel.geom)
AND floodzones.zone = 'AE'
```

Now, try Intersects (what is different?)

gisadvisor.com, LLC.

Contains / Touches

```
SELECT sum(tcparcel.asmt)
FROM tcparcel,floodzones
WHERE
 ST_Contains(floodzones.geom,tcparcel.geom)
AND floodzones.zone = 'AE'
```

Now, do it with a group by statement for the propclass

gisadvisor.com, LLC.

Distance

```
SELECT
  min(ST_Distance(tcparcel.geom,floodzones.geom))::inte
  ger AS   dist, tcparcel.parcelkey
FROM tcparcel, floodzones
WHERE ST_Distance(tcparcel.geom,floodzones.geom) < 300
GROUP BY parcelkey
```

---- http://postgis.refractions.net/docs/ST_DWithin.html

```
SELECT
  min(ST_Distance(tcparcel.geom,floodzones.geom))::inte
  ger AS   dist, tcparcel.parcelkey
FROM tcparcel, floodzones
WHERE ST_DWithin(tcparcel.geom,floodzones.geom,300)
GROUP BY parcelkey
```

gisadvisor.com, LLC.

Distance

```
SELECT
  ST_Distance(tcparcel.geom,floodzones.geom)::integer
  as dist, tcparcel.parcelkey
FROM tcparcel, floodzones
WHERE tcparcel.parcelkey =
  '50070000900000040130000000'
AND floodzones.zone = 'AE'        -- now get the
  nearest one!!
```

gisadvisor.com, LLC.

Many ways to skin an SQL cat - distance

Find the sum of the ASMT for the residential properties within 300 feet of the AE flood zone – note the difference in time!

gisadvisor.com, LLC.

Many ways to skin an SQL cat - distance

```
SELECT sum(asmt) FROM
 (SELECT asmt, ST_Distance(tcparcel.geom,tcparcel.geom) AS dist,
  tcparcel.parcelkey
  FROM tcparcel, floodzones
  WHERE floodzones.zone = 'AE' AND propclass = 'Residential'
  ) AS T1
WHERE dist < 300

SELECT sum(asmt)
FROM tcparcel, floodzones
WHERE floodzones.zone = 'AE' AND propclass = 'Residential'
AND ST_Distance(tcparcel.geom,floodzones.geom) < 300
```

gisadvisor.com, LLC.

Many ways to skin an SQL cat - distance

```
SELECT sum(asmt)
FROM tcparcel, floodzones
WHERE floodzones.zone = 'AE' AND propclass =
  'Residential'
AND ST_DWithin(tcparcel.geom,floodzones.geom,
  300)
```

gisadvisor.com, LLC.

ST_Intersects

```
SELECT d.geom, d.parcelkey
FROM tcparcel AS d, floodzones
WHERE ST_Intersects(d.geom,floodzones.geom)
AND floodzones.zone = 'AE'
```

--- Now, get the sum of the geometry area

```
SELECT
sum(st_area(ST_Intersection(d.geom,floodzones.geom
))) as sumarea
FROM tcparcel AS d, floodzones
WHERE ST_Intersects(d.geom,floodzones.geom)
AND floodzones.zone = 'AE' ) AS T1
```

gisadvisor.com, LLC.

ST_Union (ST_Collect)

```
SELECT St_Union(geometry) AS geom
FROM parcels
```

ST_Union is an aggregate function, just like SUM. So, create a MAP Book layer, grouping the geometries by the left 3 most letters in the print key.

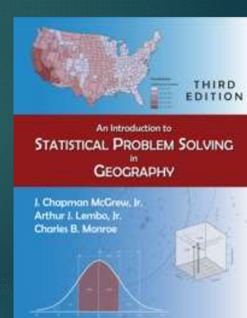
Next, create a land use map

```
SELECT St_Union(geom) AS geom, propclass
FROM tcparcel
WHERE muni = 'Lansing'
GROUP BY propclass
```

gisadvisor.com, LLC.

Geography and SQL

BASED ON THE TEXTBOOK
AN INTRODUCTION TO STATISTICAL PROBLEM SOLVING IN GEOGRAPHY. WAVELAND PRESS.
C. MCGREW, LEMBO, A, MONROE, C.



gisadvisor.com, LLC.

Basic problems in geography

- McGrew, Lembo, and Monroe present an introduction to spatial analysis and GIS within the context of statistics in geography
- Many of these classic spatial analysis tasks are difficult to accomplish with GIS
- However, with spatial SQL, they become self contained autonomous queries

gisadvisor.com, LLC.

Basic problems in geography

```
CREATE TABLE points(gid serial PRIMARY KEY, name varchar, freq integer, geom
geometry(POINT, 2248) );
INSERT INTO points (name, freq, geom)
VALUES ('A', 5, ST_SetSRID(ST_MakePoint(2.8,1.5),2248));

INSERT INTO points (name, freq, geom)
VALUES ('B', 20, ST_SetSRID(ST_MakePoint(1.6,3.8),2248));
INSERT INTO points (name, freq, geom)
VALUES ('C', 8, ST_SetSRID(ST_MakePoint(3.5,3.3),2248));
INSERT INTO points (name, freq, geom)
VALUES ('D', 4, ST_SetSRID(ST_MakePoint(4.4,2.0),2248));
INSERT INTO points (name, freq, geom)
VALUES ('E', 6, ST_SetSRID(ST_MakePoint(4.3,1.1),2248));
INSERT INTO points (name, freq, geom)
VALUES ('F', 5, ST_SetSRID(ST_MakePoint(5.2,2.4),2248));

INSERT INTO points (name, freq, geom)
VALUES ('G', 3, ST_SetSRID(ST_MakePoint(4.9,3.5),2248));
```

gisadvisor.com, LLC.

Central Feature

```
SELECT SUM(ST_Distance(a.geom,b.geom)) AS
dist, a.name
FROM points AS a, points AS b
WHERE a.name <> b.name
GROUP BY a.name
ORDER BY dist
LIMIT 1
```

TABLE 4.2

Worktable for Calculating Central Point

Distance matrix between points in figure 4.1

	A	B	C	D	E	F	G	Total Distance
A	0	2.59	1.93	1.68	1.55	2.56	2.90	13.21
B	2.59	0	1.96	3.33	3.82	3.86	3.31	18.87
C	1.93	1.96	0	1.58	2.34	1.92	1.41	11.14
D	1.68	3.33	1.58	0	0.91	0.89	1.58	9.97
E	1.55	3.82	2.34	0.91	0	1.58	2.47	12.67
F	2.56	3.86	1.92	0.89	1.58	0	1.14	11.95
G	2.90	3.31	1.41	1.58	2.47	1.14	0	12.81

Point "D" is the lowest total distance from all other points: 9.97

Euclidean distance from point A to point B:

$$d_{AB} = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2}$$
$$= \sqrt{(2.8 - 1.6)^2 + (1.5 - 3.8)^2}$$
$$= 2.59$$

Locational coordinates: A (2.8, 1.5), B (1.6, 3.8)

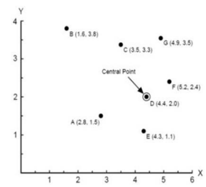


FIGURE 4.1
Graph of Locational Coordinates and Central Point

gisadvisor.com, LLC.

Mean Center

```
-- MEAN CENTER

SELECT  Avg(x) AS X,
        Avg(y) AS Y
FROM points

 -- USING GEOMETRY
SELECT  Avg(ST_X(geom)) AS X,
        Avg(ST_Y(geom)) AS Y
FROM points

-- WEIGHTED MEAN CENTER

SELECT Sum(ST_X(geom)*freq)/(SELECT sum(freq) FROM points),
       Sum(ST_Y(geom)*freq)/(SELECT sum(freq) FROM points)
FROM points
```

TABLE 4.3

Worktable for Calculating Weighted Mean Center

	Locational coordinates *		Weight	Weighted coordinates	
Point	X_i	Y_i	f_i	$f_i X_i$	$f_i Y_i$
A	2.8	1.5	5	14.0	7.5
B	1.6	3.8	20	32.0	76.0
C	3.5	3.3	8	28.0	26.4
D	4.4	2.0	4	17.6	8.0
E	4.3	1.1	6	25.8	6.6
F	5.2	2.4	5	26.0	12.0
G	4.9	3.5	3	14.7	10.5

$n = 7$ $\quad \sum f_i = 51$ $\quad \sum f_i X_i = 158.1$ $\quad \sum f_i Y_i = 147.0$

$$\bar{X}_{wc} = \frac{\sum f_i X_i}{\sum f_i} = \frac{158.1}{51} = 3.10$$

$$\bar{Y}_{wc} = \frac{\sum f_i Y_i}{\sum f_i} = \frac{147.0}{51} = 2.88$$

Weighted mean center coordinates: (3.10, 2.88) *

* See figure 4.4 for graph of point locations, frequencies, and weighted mean center.

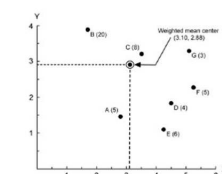


FIGURE 4.4
Graph of Point Locations, Weights (in Parentheses) and Weighted Mean Center

gisadvisor.com, LLC.

Nearest Neighbor

```
SELECT a.geom, st_distance(a.geom, b.geom) dist, a.placename aname, b.placename bname
FROM firestations a, firestations b
WHERE a.placename <> b.placename
ORDER BY aname, dist ASC-----

-----

SELECT aname, min(dist) dist FROM
        (SELECT a.geom, st_distance(a.geom, b.geom) dist, a.placename aname, b.placename
bname
         FROM firestations a, firestations b
         WHERE a.placename <> b.placename
         ORDER BY aname, dist ASC
        ) AS T1
GROUP BY aname

------
SELECT avg(dist) FROM
(SELECT aname, min(dist) dist FROM
        (SELECT a.geom, st_distance(a.geom, b.geom) dist, a.placename aname, b.placename
bname
         FROM firestations a, firestations b
         WHERE a.placename <> b.placename
         ORDER BY aname, dist ASC
        ) AS T1
GROUP BY aname) AS T2
```

The Variogram

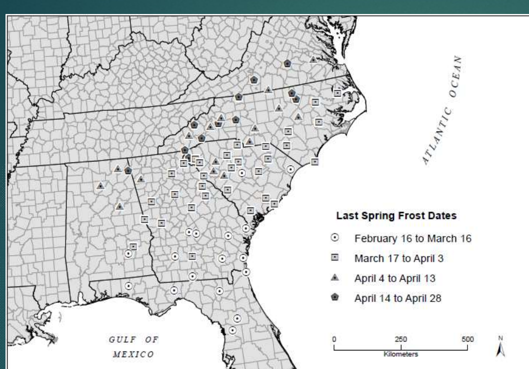


FIGURE 1.6
Average Date of Last Spring Frost, Selected Weather Stations in Southeast United States, 1950 to 2010

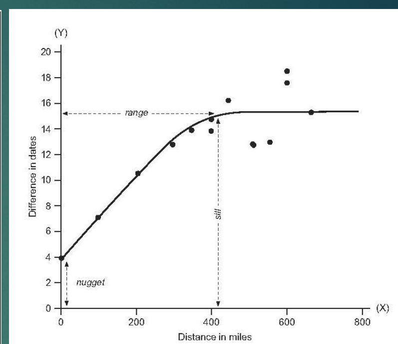


FIGURE 13.3
Variogram of Last Spring Frost (LSF) Date

gisadvisor.com, LLC.

Calculating the Variogram

```
SELECT dist*100 as h,semivariance
FROM
   (SELECT dist,avg(abs(diff))/2 AS semivariance
    FROM
   (SELECT Floor(ST_Distance(p.geom,P3.geom,true)*0.000621371/100) AS dist,
    (p.avgdlsf - p3.avgdlsf) AS diff
   FROM lsf AS p, lsf AS p3) AS T1
GROUP BY dist
ORDER BY dist
) AS T2
```

```
---- EXCEL FORMULA ----
=NUMBERVALUE(LEFT(A1,SEARCH(";",A1)-1))
=NUMBERVALUE(RIGHT(A1,LEN(A1)-SEARCH(";",A1)))
```

gisadvisor.com, LLC.

The Variogram (lat/lon)

```
SELECT dist*100 as h,semivariance
FROM
   (SELECT dist,avg(abs(diff))/2 AS semivariance
    FROM
   (SELECT Floor(ST_Distance_Sphere(p.geom,P3.geom)*0.000621371/100) AS dist,
    (p.avgdlsf - p3.avgdlsf) AS diff
   FROM lsf AS p, lsf AS p3) AS T1
GROUP BY dist
ORDER BY dist
) AS T2
```

gisadvisor.com, LLC.

Distance, Adjacency, and Interaction

- One chapter discusses the concept of distance, adjacency, and interaction.
- One way to integrate the concepts together are to represent them in the form of matrices which are later used in other computations.
  - These matrices represent *distance, adjacency,* and *interaction*.

gisadvisor.com, LLC.

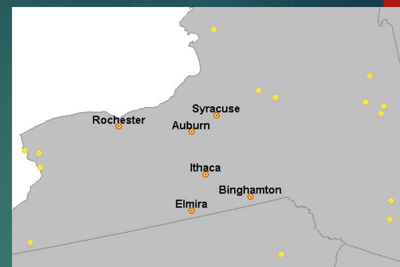
Creating the dataset

```
CREATE TABLE sixcities(gid serial PRIMARY KEY, name varchar, geom geometry(POINT,
4326) );
INSERT INTO sixcities (name, geom)
VALUES ('Syracuse',ST_SetSRID(ST_MakePoint(-76.1474,43.0481),4326));

INSERT INTO sixcities (name, geom)
VALUES ('Rochester',ST_SetSRID(ST_MakePoint(-77.6088,43.1566),4326));

INSERT INTO sixcities (name, geom)
VALUES ('Ithaca',ST_SetSRID(ST_MakePoint(-76.5019,42.4440),4326));

INSERT INTO sixcities (name, geom)
VALUES ('Auburn',ST_SetSRID(ST_MakePoint(-76.5661,42.9317),4326));

INSERT INTO sixcities (name, geom)
VALUES ('Binghamton',ST_SetSRID(ST_MakePoint(-75.9180,42.0987),4326));

INSERT INTO sixcities (name, geom)
VALUES ('Elmira',ST_SetSRID(ST_MakePoint(-76.8077,42.0898),4326));

SELECT * FROM sixcities
```

gisadvisor.com, LLC.

Distance

- **As an example of distance let's illustrate a 6 x 6 symmetric distance matrix.**
  - Similarly, we illustrate the creation of a distance matrix using six cities in upstate New York (Rochester, Syracuse, Auburn, Ithaca, and Binghamton).



- **The computation of distances between cities is easily accomplished with the following SQL statement:**

```
SELECT sixcitiesA.name, sixcitiesB.name,
ST_Distance_Sphere(sixcitiesA.geom,sixcitiesB.geom)*0.000621371
AS citydist
FROM sixcities AS sixcitiesA, sixcities AS sixcitiesB
```

gisadvisor.com, LLC.

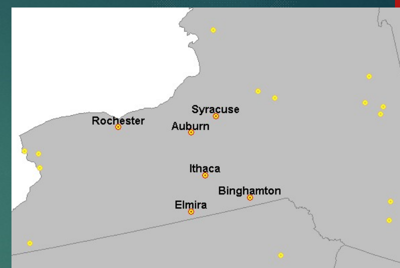
Distance

However, this SQL declaration returns a table, and not a matrix. To create a distance matrix, a pivot table is used to show distances between each city:

```
COPY (SELECT 'cities' || ', ' || string_agg(name, ',')

FROM (
       SELECT name::text
       FROM sixcities
       ORDER BY name
) AS T

         UNION ALL
SELECT cid || ',' || string_agg(did, ',')
FROM (
SELECT a.name as cid, st_distance_sphere(a.geom,b.geom)::int::text as did
FROM sixcities AS a, sixcities AS b
ORDER by a.name, b.name
) AS T
GROUP BY cid) TO 'c:\temp\dist.csv'
```

gisadvisor.com, LLC.

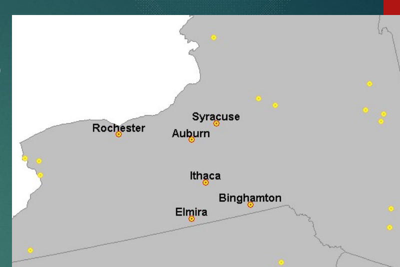
Adjacency



- An adjacency matrix is similar to the distance matrix except the matrix elements are now ones or zeros. To illustrate let's quantify an adjacent relationship as one where the distance between two objects is less than 50 meters.
- Similarly, we can set a distance of 50 miles as an indication of adjacency and modify the SQL declarations from above, to create the adjacency matrix as follows:

gisadvisor.com, LLC.

Adjacency




```
COPY
(SELECT 'cities' || ', ' || string_agg(name, ',')

FROM (
        SELECT name::text
        FROM sixcities
        order by name
) AS T

UNION ALL

SELECT cid || ',' || string_agg(did::text, ',')
FROM (
SELECT a.name as cid,
                                    CASE
                                   WHEN
st_distance_sphere(a.geom,b.geom)*0.000621371 > 50 THEN '1'
                                   ELSE '0'
                                END AS did
FROM sixcities AS a, sixcities AS b
ORDER by a.name, b.name
) AS T
GROUP BY cid)

TO 'c:\temp\adj.csv'
```

gisadvisor.com, LLC.

Interaction

One can easily adapt the SQL code to create an *interaction* or *weights* matrix, W.
To illustrate this, we'll use the concept of an *inverse distance weight* (1/d), which we easily reproduce here in SQL using a single character change from the previous SQL query (code change reflected by enlargement):

```
COPY (SELECT 'cities' || ', ' || string_agg(name,
',')

FROM (
       SELECT name::text
       FROM sixcities
       ORDER BY name
) AS T

         UNION ALL
SELECT cid || ',' || string_agg(did, ',')
FROM (
SELECT a.name as  cid,
(1/(st_distance(a.geom,b.geom)*0.000621+1))::text as
did
FROM sixcities AS a, sixcities AS b
ORDER by a.name, b.name
) AS T
GROUP BY cid) TO 'c:\temp\dist.csv'
```

gisadvisor.com, LLC.

Replicating complex formulas

MORAN'S *I* CALCULATION

gisadvisor.com, LLC.

Morans *I*

- Remember the formula for Morans *I*

$$I = \frac{n\sum_i \sum_j w_{i,j}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i \sum_j w_{i,j}(x_i - \bar{x})^2}$$

- Now, we have to break things down

gisadvisor.com, LLC.

What is $w_{i,j}$

```
SELECT 'states' || ', ' || string_agg(name, ',')
FROM (
       SELECT name
       FROM states
       order by name
) AS T

UNION ALL

SELECT cid || ',' || string_agg(did::text, ',')
FROM (
SELECT a.name as cid,
                      CASE
                       WHEN st_touches(a.geom,b.geom) THEN '1'
                       ELSE '0'
                     END AS did
FROM states AS a,states AS b
ORDER by a.name, b.name
) AS T
GROUP BY cid;
```

gisadvisor.com, LLC.

What is $w_{i,j}$

```
COPY
(SELECT *
FROM
(SELECT 'states' || ', ' || string_agg(name, ',')
FROM (
        SELECT name
        FROM states
        order by name
) AS T

union all

SELECT cid || ',' || string_agg(did::text, ',')
FROM (
SELECT a.name as cid,
                          CASE
                         WHEN st_touches(a.geom,b.geom) THEN '1'
                         ELSE '0'
                       END AS did
FROM states AS a,states AS b
ORDER by a.name, b.name
) AS T
GROUP BY cid) as w)
TO 'c:\temp\w.csv'
```

gisadvisor.com, LLC.

Morans *I*

$$I = \frac{n \sum_i \sum_j w_{i,j}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i \sum_j w_{i,j}(x_i - \bar{x})^2}$$

- where do we get *n*

```
SELECT count(*)
FROM states
```

$$I = \frac{(select\ count(*)\ from\ ms) \sum_i \sum_j w_{i,j}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i \sum_j w_{i,j}(x_i - \bar{x})^2}$$

- *where do we get*

```
SELECT avg(obesity)
FROM states
```

$$I = \frac{(select\ count(*)\ from\ ms) \sum_i \sum_j w_{i,j}(x_i - (select\ avg(obesity)\ from\ ms))(x_j - (select\ avg(obesity)\ from\ ms))}{\sum_i \sum_j w_{i,j}(x_i - \bar{x})^2}$$

- where do we get

$$\sum_i \sum_j w_{i,j}(x_i - \bar{x})(x_j - \bar{x})$$

```
SELECT sum((a.ob_2009 - (SELECT avg(ob_2009)
                                   FROM states))
           * (b.ob_2009 - (SELECT avg(ob_2009)
                                   FROM states)))*.5
        FROM  states a, states b
        WHERE st_touches(a.geom,b.geom)
```

gisadvisor.com, LLC.

Morans *I*

- Lets grab the numerator

$$(select\ count(*)\ from\ \ ms)\sum_i \sum_j w_{i,j}(x_i - \bar{x})(x_j - \bar{x})$$

```
SELECT count(*) *
       (SELECT sum((a.ob_2009 - (SELECT avg(ob_2009) FROM states))
             * (b.ob_2009 - (SELECT avg(ob_2009) from states)))*.5
       FROM  states a, states b
       WHERE st_touches(a.geom,b.geom))
       FROM states
```

gisadvisor.com, LLC.

Morans *I*

- Lets grab the denominator

$$\sum_i \sum_j w_{i,j}(x_i - \bar{x})^2$$

```
SELECT sum((states.ob_2009 - (select avg(ob_2009) from states))^2)
                    *   (select count(*)/2 from states a, states b
                         where st_touches(a.geom,b.geom))
                         from states
```

- Now we can put it all together

```
SELECT ( (SELECT count(*) *        (SELECT sum((a.ob_2009 - (SELECT avg(ob_2009)
FROM states))              *
(b.ob_2009 - (SELECT avg(ob_2009) from states)))*.5         FROM  states a,
states b
WHERE st_touches(a.geom,b.geom)) AS numer
FROM states) /
(SELECT sum((states.ob_2009 - (select avg(ob_2009) from states))^2)
                                 *   (select count(*)/2 from states a, states
b
                                 where st_touches(a.geom,b.geom))
                                 from states
```

gisadvisor.com, LLC.

Functions

- Functions are confusing, let's take this slow....

```
CREATE OR REPLACE FUNCTION GetAccts(text) RETURNS SETOF text AS
$$
    SELECT parcelkey FROM tcparcel WHERE propclass = $1
$$ LANGUAGE SQL;
SELECT GetAccts('Residential');
```

gisadvisor.com, LLC.

Functions

- Create a function with a spatial construct.

```
CREATE OR REPLACE FUNCTION getflood (x text)
RETURNS TABLE(parcelkey text) AS $$
    SELECT parcelkey FROM tcparcel,floodzones
     WHERE
st_intersects(tcparcel.geom,floodzones.geom) AND
floodzones.zone = $1;
$$ LANGUAGE SQL;

SELECT getflood('X500')
```

gisadvisor.com, LLC.

Spatial Function

```
CREATE FUNCTION getfloodgeom (x text)
RETURNS TABLE(mygeom geometry) AS $$
    SELECT
ST_Intersection(parcels.geometry,flood.geometry) AS
geometry
    FROM parcels,flood WHERE
ST_Intersects(parcels.geometry,flood.geometry) AND
flood.zone = $1;
$$ LANGUAGE SQL;

SELECT getfloodgeom('X')
```

gisadvisor.com, LLC.

Real World Examples

- Parcels on hydric soils
- Closing a fire station

gisadvisor.com, LLC.

Parcels on hydric soils

```
SELECT *
FROM tcsoils
WHERE mukey IN (SELECT mukey FROM
  (SELECT sum(comppct_r) AS comppct, mukey
   FROM component
   WHERE hydricrating = 'Yes'
   GROUP BY mukey) AS t1
WHERE comppct > 50 )

-- now add in the parcels….
```

gisadvisor.com, LLC.

Closing a fire station

```
SELECT count(tcbuildings.geom) AS totbldg,
firestationisochrone.placename
FROM firestationisochrone, tcbuildings
WHERE ST_Contains(ST_Transform(firestationisochrone.geom,32618),
tcbuildings.geom)
GROUP BY placename
ORDER BY totbldg DESC


 SELECT count(*), closedstation
 FROM tcbuildings,
(SELECT st_union(g) as g, placename as closedstation
FROM firestationisochrone, (SELECT ST_Transform(geom,32618) g,
placename AS pn
 FROM firestationisochrone) AS t
 WHERE placename <> pn
 GROUP BY placename) AS t
WHERE NOT ST_Intersects(tcbuildings.geom,t.g)
GROUP BY closedstation
```

gisadvisor.com, LLC.

Closing a fire station

```
SELECT st_union(g) as g1, placename as closedstation
INTO test FROM firestationisochrone, (SELECT geom g, placename AS pn
 FROM firestationisochrone) AS t
 WHERE placename <> pn
 GROUP BY placename;

SELECT count(*), closedstation
FROM tcparcel, test
WHERE NOT ST_Intersects(ST_Transform(tcparcel.geom,4326),test.g1)
GROUP BY closedstation
```

gisadvisor.com, LLC.

Conclusion

- Spatial SQL makes it easy to perform powerful spatial and attribute queries
- A single declarative SQL query can perform as much geo-processing as a couple of pages of scripting
- Users should consider learning spatial SQL to become more productive
- Vendors should consider integrating spatial SQL into their desktop software products

gisadvisor.com, LLC.

Real life scenario

CRISFIELD FLOODS

gisadvisor.com, LLC.

Scenario: The Town of Crisfield just got flooded this week after the rainstorms. You have been brought in to support the response to the disaster. The Mayor is getting hammered by the press, families, and the State Emergency Management Office:

1. Total number of properties (parcels) that are under 2 feet of water (flood layer, gridcode > 6)
2. Total value of the land (parcels layer, column "NFMTTLVL)" that is under 2 feet of water (flood layer, column gridcode > 6):
3. Total value of land, grouped by the landuse ("LU") that is under 2 feet of water (flood layer, column gridcode > 6)

gisadvisor.com, LLC.

4. What government buildings (gov_bldg) are under water?

5. Greg Sterling just called. He is out of town and wants to know if his house is under water. If it is, what is the maximum amount of water depth (gridcode). His account number is '2007124449'

6. The Highway Superintendent needs to know the names of the streets ("FULL_NAME") and their cross streets ("FROMCROSS", "TOCROSS") that are under 2 foot of water (gridcode = 5) so he can close off the street.

gisadvisor.com, LLC.

Shameless plug

- www.gisadvisor.com/tugis
  - Spatial SQL: A Language for Geographers
  - Python for Geospatial
- www.artlembo.com
  - *How do I do that…*

gisadvisor.com, LLC.