



DATABASE MANAGEMENT SYSTEMS

SPRING 2023

CS – 632 PROJECT

Professor: Mr. Buchi Okoli Okoli

Project Members

Kaleeswaran Sivasankaran, James Helloween, Ravi Shankar Dhwarakesh

GOAL

Our goal is to provide the Geographic Information System (GIS) analysis, using database that supports spatial data types which is PostGIS.

DATASOURCE: <https://github.com/opengeos/data>

In the above git hub link, the Geographic Information of US, China, World Cities and other spatial data will be available from that we will be using US data source in this analysis.

Data Source of US: In this data source 3 different tables will be created for states, cities and counties.

Importing the Data: Data will be imported using PostGIS share file import/export manager, which is a tool that allows us to import shapefiles from a specified path.

After adding the files in the UI alter the SRID to 4326 since it represents spatial data using longitude and latitude coordinates on the Earth's surface and import the data.

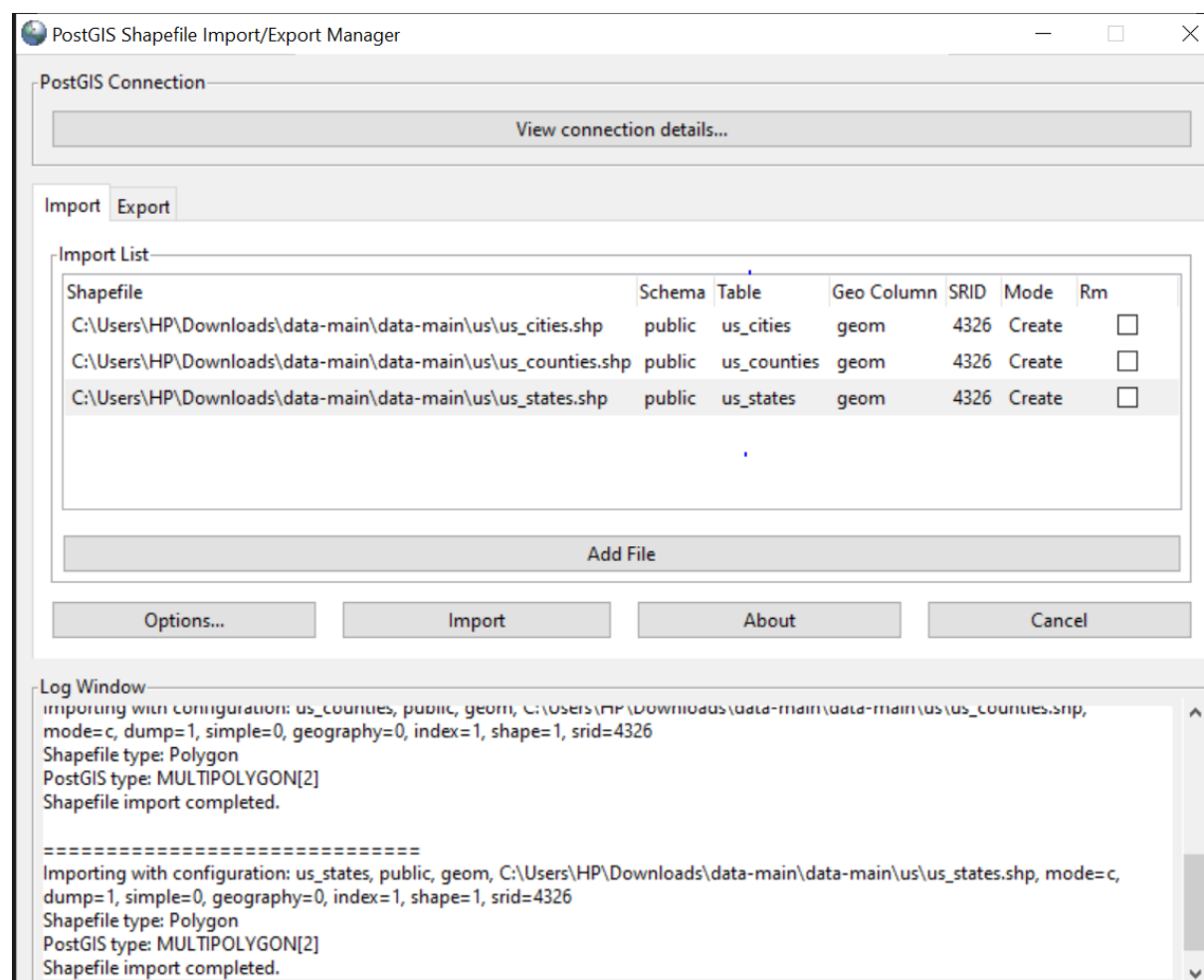


Table Schema of us_cities, us_states and us_counties.

Query

Query History

1

```
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_schema = 'public' AND table_name = 'us_cities';
```

Data Output

Messages

Notifications

column_name

name

data_type

character varying

is_nullable

character varying (3)

1

gid

integer

NO

2

id

character varying

YES

3

pop_2010

numeric

YES

4

elev_in_ft

numeric

YES

5

state

character varying

YES

6

geom

USER-DEFINED

YES

Query

Query History

1

SELECT column_name, data_type, is_nullable

2

FROM information_schema.columns

3

WHERE table_schema = 'public' AND table_name = 'us_counties';

Data Output

Messages

Notifications

column_name

name

data_type

character varying

is_nullable

character varying (3)

1

gid

integer

NO

2

geo_id

character varying

YES

3

state

character varying

YES

4

county

character varying

YES

5

name

character varying

YES

6

lsad

character varying

YES

7

censusarea

numeric

YES

8

geom

USER-DEFINED

YES

Query Query History

```
1 SELECT column_name, data_type, is_nullable
2 FROM information_schema.columns
3 WHERE table_schema = 'public' AND table_name = 'us_states';
```

Data Output Messages Notifications

	column_name name	data_type character varying	is_nullable character varying (3)
1	gid	integer	NO
2	id	character varying	NO
3	name	character varying	YES
4	geom	USER-DEFINED	YES

Add and Verify PostGIS extension

Query to add the postgis extension

```
CREATE EXTENSION postgis;
```

Query to verify the existence

```
SELECT postgis_version();
```

sdb/postgres@PostgreSQL 15

No limit

Query Query History

1 SELECT postgis_version();

Data Output Messages Notifications


	postgis_version text
1	3.3 USE_GEOS=1 USE_PROJ=1 USE_STATS=1

Retrieve Locations of specific features

```
SELECT id, name, ST_AsText(geom)
FROM us_states
WHERE name = 'Texas';
```


ST_AsText is a method in PostGIS, an extension to the PostgreSQL database system, that returns the Well-Known Text (WKT) representation of a geometry or geography instance

Query		Query History	
1	SELECT id, name, ST_AsText(geom)		
2	FROM us_states		
3	WHERE name = 'Texas';		
4			

Data Output		Messages	Notifications
			
	id character varying (80)	name character varying (80)	st_astext text
1	TX	Texas	MULTIPOLYGON(((-101.81294;

```
SELECT id, state, ST_AsText(geom)
FROM us_cities
WHERE state = 'TX';
```

Query		Query History	
1	SELECT id, state, ST_AsText(geom)		
2	FROM us_cities		
3	WHERE state = 'TX';		
4			

Data Output		Messages	Notifications
			
	id character varying (80)	state character varying (80)	st_astext text
1	18193	TX	POINT(-97.77982679999997 31.134621300000106)
2	18218	TX	POINT(-97.73973129999996 30.513532000000055)
3	18219	TX	POINT(-97.77500879999991 30.442701100000193)
4	18248	TX	POINT(-95.17187719999998 29.778282300000114)
5	18250	TX	POINT(-95.38021489999988 29.932445000000087)
6	18251	TX	POINT(-95.17659779999997 29.998830600000019)
7	18253	TX	POINT(-95.11465329999999 29.776059900000064)
8	18260	TX	POINT(-98.26251269999989 29.87521770000012)
9	18265	TX	POINT(-95.65772449999997 29.668566000000055)
10	18266	TX	POINT(-95.44744139999995 29.538846500000009)
11	18267	TX	POINT(-95.67578109999988 29.554125900000088)

```
SELECT c.gid, c.county, c.name, s.name, ST_AsText(c.geom)
```

```
FROM us_counties c, us_states s
```

```
WHERE s.id = 'NY' and s.gid = c.state;
```

Query

Query History

Scratch Pad

1

SELECT c.gid, c.county, c.name, s.name, ST_AsText(c.geom)

2

FROM us_counties c, us_states s

3

WHERE s.id = 'NY' and s.gid = c.state;

Data Output

Messages

Notifications

	gid integer	county character varying (3)	name character varying (90)	name character varying (80)	st_astext text
1	1748	001	Churchill	New York	MULTIPOLYGON(((-118.83101500000001
2	1749	003	Clark	New York	MULTIPOLYGON(((-114.61736107700101
3	1750	005	Douglas	New York	MULTIPOLYGON(((-119.65664296384001
4	1751	007	Elko	New York	MULTIPOLYGON(((-115.73322061081201
5	1752	009	Esmeralda	New York	MULTIPOLYGON(((-117.166000489853 36
6	1753	011	Eureka	New York	MULTIPOLYGON(((-116.157506 40.66535
7	1754	013	Humboldt	New York	MULTIPOLYGON(((-117.53019739656601
8	1755	015	Lander	New York	MULTIPOLYGON(((-116.600047 40.10516
9	1756	017	Lincoln	New York	MULTIPOLYGON(((-114.048473 37.809861
10	1757	019	Lyon	New York	MULTIPOLYGON(((-118.905839 38.51420

Query selects the names of US states from a table us_states where the geometry of the state intersects with a polygon defined in the coordinates.

```
SELECT name FROM us_states WHERE ST_Intersects(geom, ST_GeomFromText(
    'POLYGON((-106.65 25.84, -106.65 36.50, -93.51 36.50, -93.51 25.84, -106.65
25.84))', 4326));
```

Query

Query History

1

SELECT name

2

FROM us_states

3

WHERE ST_Intersects(geom, ST_GeomFromText(

4

'POLYGON((-106.65 25.84, -106.65 36.50, -93.51 36.50, -93.51 25.84, -106.65 25.84))', 4326));

Data Output

Messages

Notifications

name

character varying (80)

1

Texas

2

Louisiana

3

Arkansas

4

Oklahoma

5

New Mexico

```
SELECT geo_id, name, ST_AsText(geom)
FROM us_counties
WHERE ST_Intersects(geom, ST_GeomFromText('MULTIPOLYGON(((
-85.79043399999999 31.320266999999998,-85.79032699999999 31.323452,-85.790116
31.330081999999997,-85.79010000000001 31.336275999999998.....)))', 4326));
```

Query gives rows in the us_states table where the geometry of the state intersects with a geometry defined by the given Well-Known Binary (WKB) representation.

```
SELECT id, name, ST_AsText(geom)
FROM us_states
WHERE ST_Intersects(geom,
ST_GeomFromText(ST_AsText('0106000020E610000001000000010300000001...'), 4326));
```

```
Query Query History
```

```
1 SELECT id, name, ST_AsText(geom)
2 FROM us_states
3 WHERE ST_Intersects(geom,
4 ST_GeomFromText(ST_AsText('0106000020E61000000100000001030000000100000020000000D7BE805EB8425BC00D33349E088042
```

Data Output Messages Notifications

	id character varying (80)	name character varying (80)	st_astext text
1	AZ	Arizona	MULTIPOLYGON(((−109.042503 37.000263,−109.04798 31.331629,−111.074448 31.331629,−112.246513 31.704061
2	NM	New Mexico	MULTIPOLYGON(((−107.421329 37.000263,−106.868158 36.994786,−104.337812 36.994786,−103.001438 37.000263
3	UT	Utah	MULTIPOLYGON(((−112.164359 41.995232,−111.047063 42.000709,−111.047063 40.998429,−109.04798 40.998429
4	CO	Colorado	MULTIPOLYGON(((−107.919731 41.003906,−105.728954 40.998429,−104.053011 41.003906,−102.053927 41.003906
5	NV	Nevada	MULTIPOLYGON(((−117.027882 42.000709,−114.04295 41.995232,−114.048427 37.000263,−114.048427 36.195155
6	CA	California	MULTIPOLYGON(((−123.233256 42.006186,−122.378853 42.011663,−121.037003 41.995232,−120.001861 41.995232

Query Calculates the distance between the two points.

```
SELECT ST_Distance(  
    ST_GeomFromText('MULTIPOLYGON((( -85.79043399999999 31.3202669999999998,-  
85.790326999999999 31.323452,-85.790116 31.330081999999997,-85.790100000000001  
31.3362759999999998,.....)))', 4326),  
    ST_GeomFromText(ST_AsText('0106000020E610000001000000010300....'), 4326)  
)  
FROM us_counties;
```

The screenshot shows a SQL query editor with a 'Query' tab selected. The query is as follows:

```
1 SELECT ST_Distance(  
2     ST_GeomFromText('MULTIPOLYGON((( -85.79043399999999 31.3202669999999998,-85.790326999999999 31.323452,-85.79  
3     ST_GeomFromText(ST_AsText('0106000020E6100000010000000103000000010000004000000003E962D34A1A57C046EA3D95D3  
4  
5 )  
6 FROM us_counties;
```

Below the query editor, the 'Data Output' tab is selected, showing a single result row:

	st_distance double precision
1	11.358713160787058

Query calculates the maximum distance between the points.

```
SELECT ST_MaxDistance(  
    ST_GeomFromText('POINT(-97.739731299999996 30.5135320000000055)', 4326),  
    ST_GeomFromText('LINESTRING (2 2,2 2)', 4326)  
)  
FROM us_states;
```

The screenshot shows a SQL query editor with a 'Query' tab selected. The query is as follows:

```
1 SELECT ST_MaxDistance(  
2     ST_GeomFromText('POINT(-97.739731299999996 30.5135320000000055)', 4326),  
3     ST_GeomFromText('LINESTRING (2 2,2 2)', 4326)  
4 )  
5 FROM us_states;
```

Below the query editor, the 'Data Output' tab is selected, showing a single result row:

	st_maxdistance double precision
1	103.73541105577796

Query gives the total area of interest for each county in square kilometers.

```
SELECT name, SUM(ST_Area(geom::geography)) / 1000000 AS area_sq_km
FROM public.us_counties
GROUP BY name;
```

Query		Query History
1 SELECT name, SUM(ST_Area(geom::geography)) / 1000000 AS area_sq_km		
2 FROM public.us_counties		
3 GROUP BY name;		
Data Output		Messages Notifications
	name character varying (90)	area_sq_km double precision
22	Alcona	1800.3224020917792
23	Alcorn	1039.803876404917
24	Aleutians East	18772.68773492995
25	Aleutians West	12102.884586780807
26	Alexander	1337.3602996864051
27	Alexandria	40.366989894338936
28	Alfalfa	2283.1644256238587
29	Alger	2450.2145055021583
30	Allamakee	1706.0983033053176
31	Allegan	2182.662969981314
32	Allegany	3795.0627250155494
33	Alleghany	1775.1709375588246

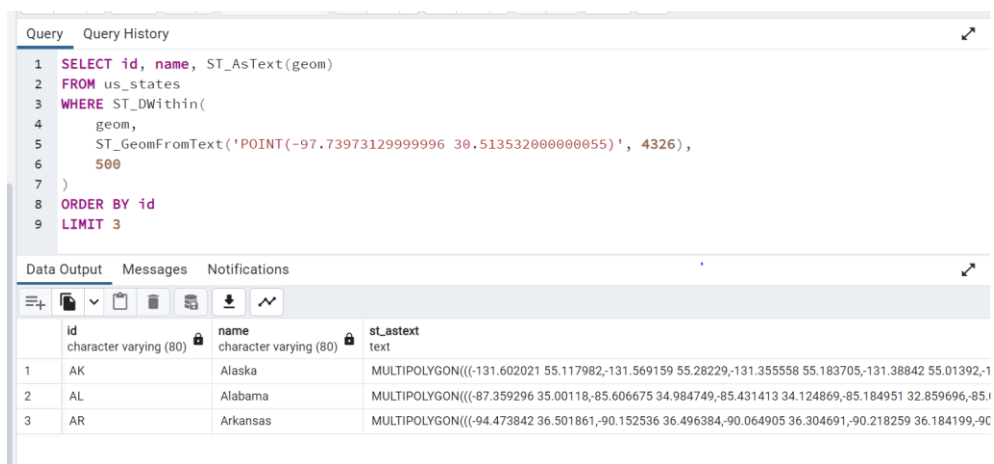
Query returns counties which is less than or equal to 1000 units.

```
SELECT name, ST_AsText(geom)
FROM us_counties
WHERE ST_DWithin(
    geom,
    ST_GeomFromText(ST_AsText('0106000020E610.....'), 4326),
    1000
)
LIMIT 10;
```

Query		Query History
1 SELECT name, ST_AsText(geom)		
2 FROM us_counties		
3 WHERE ST_DWithin(
4 geom,		
5 ST_GeomFromText(ST_AsText('0106000020E610.....'), 4326),		
6 1000		
7)		
8 LIMIT 10;		
Data Output		Messages Notifications
	name character varying (90)	st_astext text
5	Crenshaw	MULTIPOLYGON(((−86.146992 31.680455,−86.147114 31.663,−86.151156 31.662481,−86.151046 31.661721,−86.155051 31.661575,−86.157
6	Dale	MULTIPOLYGON(((−85.79043399999999 31.320266999999998,−85.790326999999999 31.323452,−85.790116 31.330081999999997,−85.790
7	DeKalb	MULTIPOLYGON(((−85.5759270787553 34.8237342946789,−85.561416 34.750079,−85.5580670538503 34.734425844267,−85.552482 34.7
8	Escambia	MULTIPOLYGON(((−87.16308387060401 30.9990407372053,−87.1640837991927 30.999010384644297,−87.17480027248219 30.9986850
9	Fayette	MULTIPOLYGON(((−87.635931 33.878744,−87.63604 33.871988999999995,−87.598446 33.870049,−87.564379 33.868604,−87.56108499999
10	Geneva	MULTIPOLYGON(((−85.772671867107 30.9946200694171,−85.7796568228993 30.9945534353192,−85.8311224511907 30.994062470763

Query returns counties which is less than or equal to 500 units.

```
SELECT id, name, ST_AsText(geom)
FROM us_states
WHERE ST_DWithin(
    geom,
    ST_GeomFromText('POINT(-97.73973129999996 30.513532000000055)', 4326),
    500
) ORDER BY id
LIMIT 3;
```



The screenshot shows a database query interface with a 'Query' tab selected. The query is as follows:

```
1 SELECT id, name, ST_AsText(geom)
2 FROM us_states
3 WHERE ST_DWithin(
4     geom,
5     ST_GeomFromText('POINT(-97.73973129999996 30.513532000000055)', 4326),
6     500
7 )
8 ORDER BY id
9 LIMIT 3
```

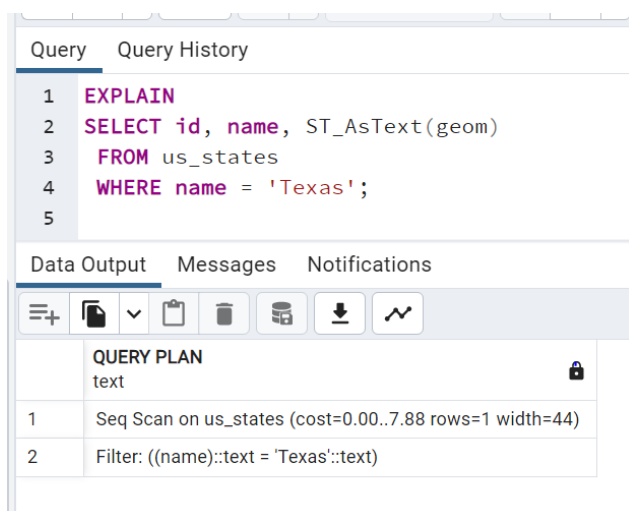
Below the query, the 'Data Output' tab is selected, displaying the results in a table format:

id	name	st_astext
1	AK	Alaska
2	AL	Alabama
3	AR	Arkansas

In PostgreSQL, **EXPLAIN** and **EXPLAIN ANALYZE** commands are used to analyze the performance of a query. These commands show the execution plan that the PostgreSQL query planner generates for a given query. The execution plan shows how the query will be executed and helps to identify the potential performance issues.

EXPLAIN

```
SELECT id, name, ST_AsText(geom)
FROM us_states WHERE name = 'Texas';
```



The screenshot shows a database query interface with a 'Query' tab selected. The query is as follows:

```
1 EXPLAIN
2 SELECT id, name, ST_AsText(geom)
3 FROM us_states
4 WHERE name = 'Texas';
5
```

Below the query, the 'Data Output' tab is selected, displaying the execution plan in a table format:

	QUERY PLAN
1	Seq Scan on us_states (cost=0.00..7.88 rows=1 width=44)
2	Filter: ((name)::text = 'Texas')::text

EXPLAIN ANALYZE

```
SELECT c.gid, c.county, c.name, s.name, ST_AsText(c.geom)
FROM us_counties c, us_states s
WHERE s.id = 'NY' and s.gid = c.state;
```

Query Query History

```
1 EXPLAIN ANALYZE
2 SELECT c.gid, c.county, c.name, s.name, ST_AsText(c.geom)
3 FROM us_counties c, us_states s
4 WHERE s.id = 'NY' and s.gid = c.state;
5
```

Data Output Messages Notifications

QUERY PLAN
text

1	Hash Join (cost=6.64..1554.48 rows=16 width=57) (actual time=1.709..3.759 rows=17 loops=1)
2	Hash Cond: (c.state = s.gid)
3	-> Seq Scan on us_counties c (cost=0.00..1519.21 rows=3221 width=4228) (actual time=0.019..2.077 rows=3221 loops=1)
4	-> Hash (cost=6.63..6.63 rows=1 width=13) (actual time=0.055..0.056 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on us_states s (cost=0.00..6.63 rows=1 width=13) (actual time=0.030..0.039 rows=1 loops=1)
7	Filter: ((id)::text = 'NY'::text)
8	Rows Removed by Filter: 49
9	Planning Time: 0.364 ms
10	Execution Time: 3.807 ms

```
CREATE INDEX IF NOT EXISTS us_counties_name_idx
ON us_counties (name);
```

Query Query History

```
1 CREATE INDEX IF NOT EXISTS us_counties_name_idx
2 ON us_counties (name);
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 183 msec.

```
CREATE INDEX IF NOT EXISTS us_counties_geom_idx
ON us_counties
USING GIST(geom);
```

Query

Query History

1

2

3

4

5

CREATE INDEX IF NOT EXISTS us_counties_geom_idx

ON us_counties

USING GIST(geom);

Data Output

Messages

Notifications

CREATE INDEX

Query returned successfully in 102 msec.

```
SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'us_counties' AND schemaname = 'public';
```

Query

Query History

1

2

3

SELECT indexname, indexdef

FROM pg_indexes

WHERE tablename = 'us_counties' AND schemaname = 'public'

Data Output

Messages

Notifications

indexname

name

indexdef

text

1

us_counties_pkey

CREATE UNIQUE INDEX us_counties_pkey ON public.us_counties USING btree (g...

2

us_counties_name_idx

CREATE INDEX us_counties_name_idx ON public.us_counties USING btree (name)

3

us_counties_geom_idx

CREATE INDEX us_counties_geom_idx ON public.us_counties USING gist (geom)

Query	Query History
1	<code>SELECT * FROM us_counties where name like 'A%'</code>

Data Output	Messages	Notifications
	Successfully run. Total query runtime: 118 msec. 121 rows affected.	

N-Optimization by creating index for name.

```
CREATE INDEX IF NOT EXISTS us_counties_name_idx  
ON us_counties (name);
```

Query	Query History
1	<code>SELECT * FROM us_counties where name like 'A%';</code>
2	
3	<code>CREATE INDEX IF NOT EXISTS us_counties_name_idx</code>
4	<code>ON us_counties (name);</code>

Data Output	Messages	Notifications
	CREATE INDEX	
	Query returned successfully in 248 msec.	

After indexing the name column, the query execution time reduced almost 30 milli seconds.

Query	Query History
1	<code>SELECT * FROM us_counties where name like 'A%';</code>
2	
3	

Data Output	Messages	Notifications
Successfully run. Total query runtime: 81 msec. 121 rows affected.		