

# SGD LAB EXP – 5

**Name :** Aditi Chhajer; **Reg. No. :** 221081009  
**Branch :** IT ; **Course Instructor :** Prof. Vedashree Awati

## Aim:

Executing PostGIS Geometry type queries.

## Theory:

### Spatial Schemas and Tables Overview

#### 1. Schemas:

- In PostgreSQL, schemas act as containers that group related tables, views, and other objects, making it easier to organize and manage databases, especially when handling extensive datasets.
- In PostGIS (a spatial extension for PostgreSQL), schemas can be structured to categorize spatial data according to criteria such as geographic regions, data types, or functional purpose. For example, creating an `india` schema helps consolidate geographic data specific to India, such as administrative boundaries, roads, rivers, and landmarks. This makes it simpler to manage and query geographic datasets without cluttering other database elements.

#### 2. Spatial Tables:

- Spatial tables contain geometry columns used for storing spatial data types. These tables often correspond to real-world features, such as points (e.g., individual landmarks), lines (e.g., highways), and polygons (e.g., state boundaries).
- For example:
- A `pts` table might include rows for various landmarks, each with their geographic coordinates.
- A `lines` table could hold line features like highways or rivers, defined as connected sequences of points.
- A `polys` table could contain polygons representing the boundaries of states or districts within India.

### Primary Keys and Geometry Columns

#### 1. Primary Keys:

- Each spatial table typically has a primary key (e.g., `gid`) that uniquely identifies every record. In spatial data management, this ensures each geographic feature, such as a building, a road segment, or a park, is distinctly identified.
- Example: In a table `india.roads`, each road might be assigned a unique `gid` to distinguish it from other roads.

## **2. Geometry Columns:**

- Geometry columns in PostGIS store spatial data such as points, lines, and polygons. These columns are defined with specific data types, such as `'POINT'`, `'LINESTRING'`, and `'POLYGON'`.
- For example, in a table storing river data, a geometry column could be of type `'LINESTRING'` to represent the path of a river.
- The `'AddGeometryColumn()'` function is frequently used to create these columns while registering them with the PostGIS system catalog. This allows for the use of spatial indexes and ensures validation of the spatial data.

## **Spatial Reference Systems (SRS)**

### **1. SRID (Spatial Reference Identifier):**

- The SRID defines the coordinate system used for geometry data, enabling precise geographic positioning. For instance, SRID 4326 represents the WGS 84 coordinate system, which is widely used in GPS.
- Example: When storing locations of cities across India, using SRID 4326 ensures that the geographic coordinates are compatible with GPS data.
- The functions `'ST_GeomFromText()'` and `'ST_SetSRID()'` can assign an SRID to spatial data, guaranteeing that all geometries adhere to a consistent coordinate system for accurate spatial operations.

## **Geometry Types and Related Functions**

### **1. Points, Lines, and Polygons:**

- **POINT:** Represents a specific location in space, such as a monument or a building.
- Example: The location of the Taj Mahal could be stored as a `'POINT'` with its latitude and longitude.
- **LINESTRING:** Represents a continuous line formed by connecting a series of points, commonly used to model features like roads or rivers.
- Example: A `'LINESTRING'` can represent National Highway 44, defined by a series of geographic points tracing its path.
- **POLYGON:** Represents an enclosed area defined by a series of points that form a closed loop.
- Example: The boundaries of New Delhi can be stored as a `'POLYGON'` to depict its area.

### **2. Key Functions:**

- `ST_GeomFromText()`: Converts a textual representation of a geometry (e.g., `"POINT(77.231 28.613)"`) into a PostGIS geometry object with a specified SRID.

- Example: `ST_GeomFromText('POINT(77.231 28.613)', 4326)` creates a point at longitude 77.231 and latitude 28.613 using SRID 4326.
- `ST_SetSRID()`: Assigns an SRID to a geometry object that may not already have one.
- `ST_MakePoint()`: Creates a point geometry using specified coordinates.
- Example: `ST_MakePoint(77.231, 28.613)` creates a point geometry for a location with specified longitude and latitude.

## Advanced Geometry Types

### 1. 3D and 4D Geometries:

- PostGIS supports:
- 3D (XYZ) geometries: Adds a third dimension, typically for elevation.
- Example: A 3D point could represent the location of the Qutub Minar with latitude, longitude, and height above sea level.
- 3D with Measure (XYM) geometries: Includes a "measure" value, often used for representing distances or other metrics.
- Example: A route stored with an XYM line might represent distance traveled along each segment.
- 4D (XYZM) geometries: Combines both elevation and measure values for comprehensive spatial representation.
- Example: Flight paths, where each point contains latitude, longitude, altitude, and a time or distance measure.

### 2. Multipart Geometries:

- MULTIPOINT, MULTILINESTRING, and MULTIPOLYGON types represent groups of related geometries.
- Example: A `MULTIPOLYGON` can store the boundaries of all islands in the Andaman and Nicobar group as a single feature, even if they are spatially separated.

## Mixed Geometry Storage

### 1. Mixed Geometry Columns:

- PostGIS allows storing different geometry types within a single column, enabling a table to hold points, lines, and polygons.
- Example: In an `india.mixed` table, a single column could contain landmarks as points, rivers as lines, and parks as polygons. This flexibility simplifies data storage but may complicate certain queries or interactions with external tools.

### 2. Usage Example:

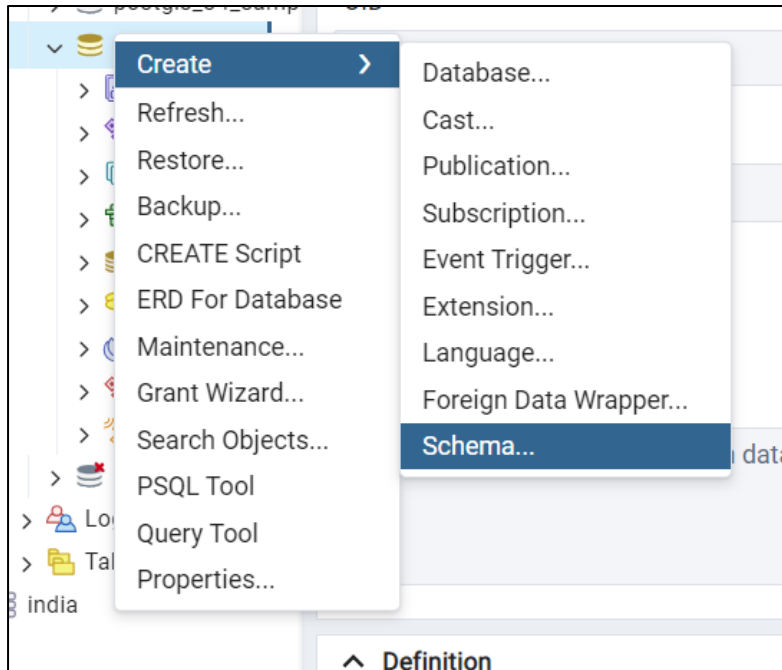
- Consider an `india.mixed` table. A row might contain a `POINT` representing the Gateway of India, another row could store a `LINESTRING` representing the Ganga River's path, and a third row could hold a `POLYGON` outlining a national park. This

approach provides a convenient way to manage different geographic features within a single table.

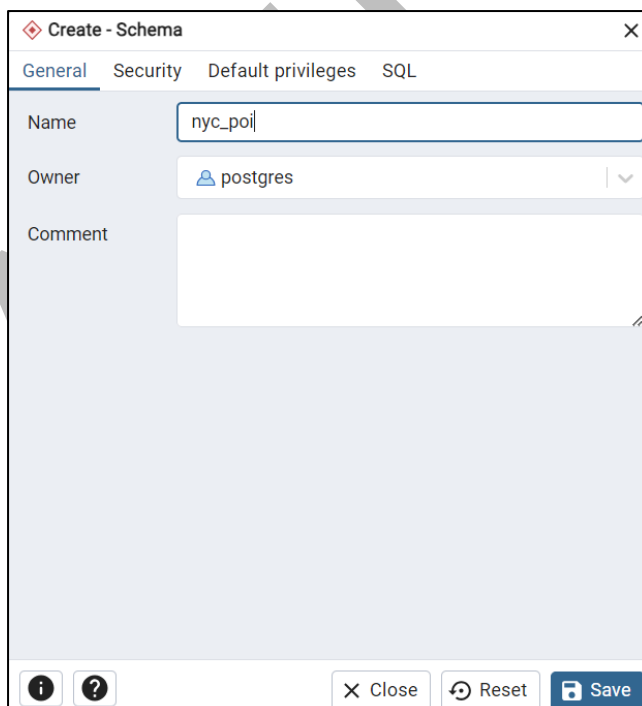
## **Implementation:**

### **1) Create a new empty spatial table:**

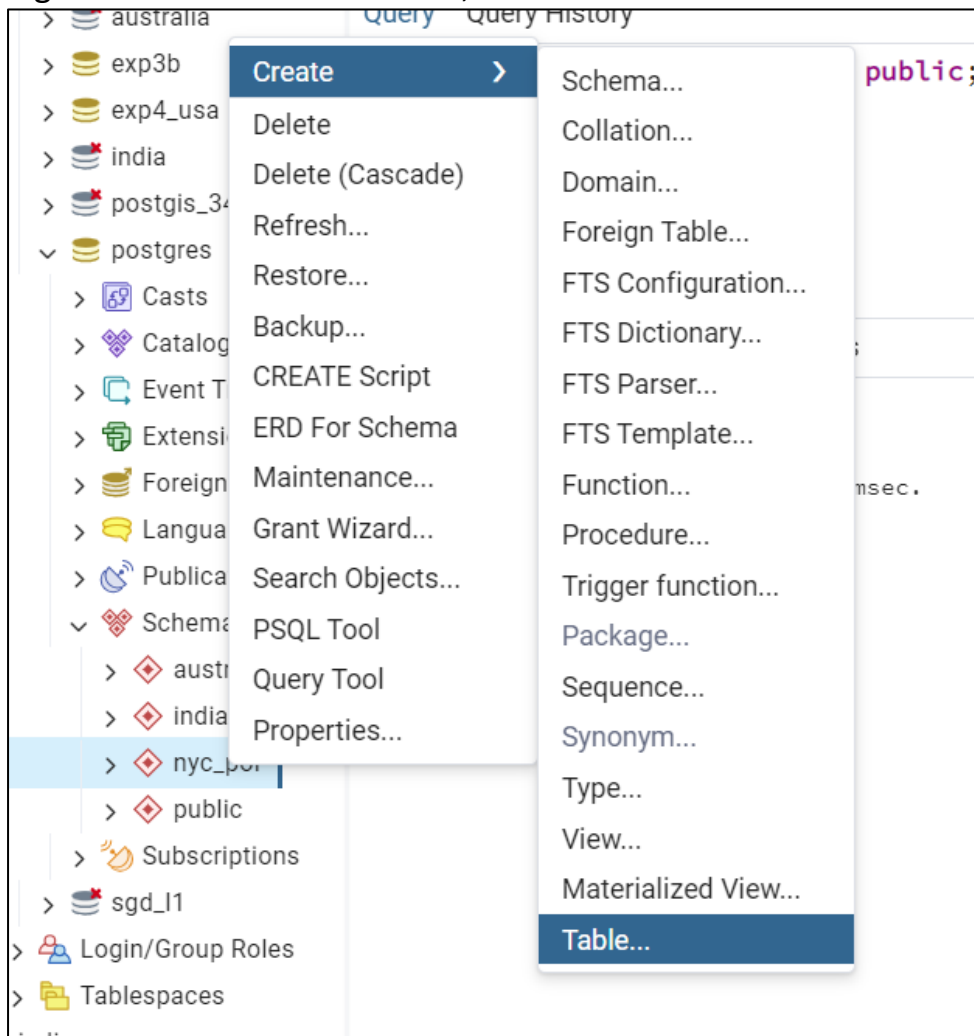
- a) In the **Browser** pane within pgAdmin, right-click on the **Schemas** node beneath the **Lesson3db** database and select **Create > Schema**.



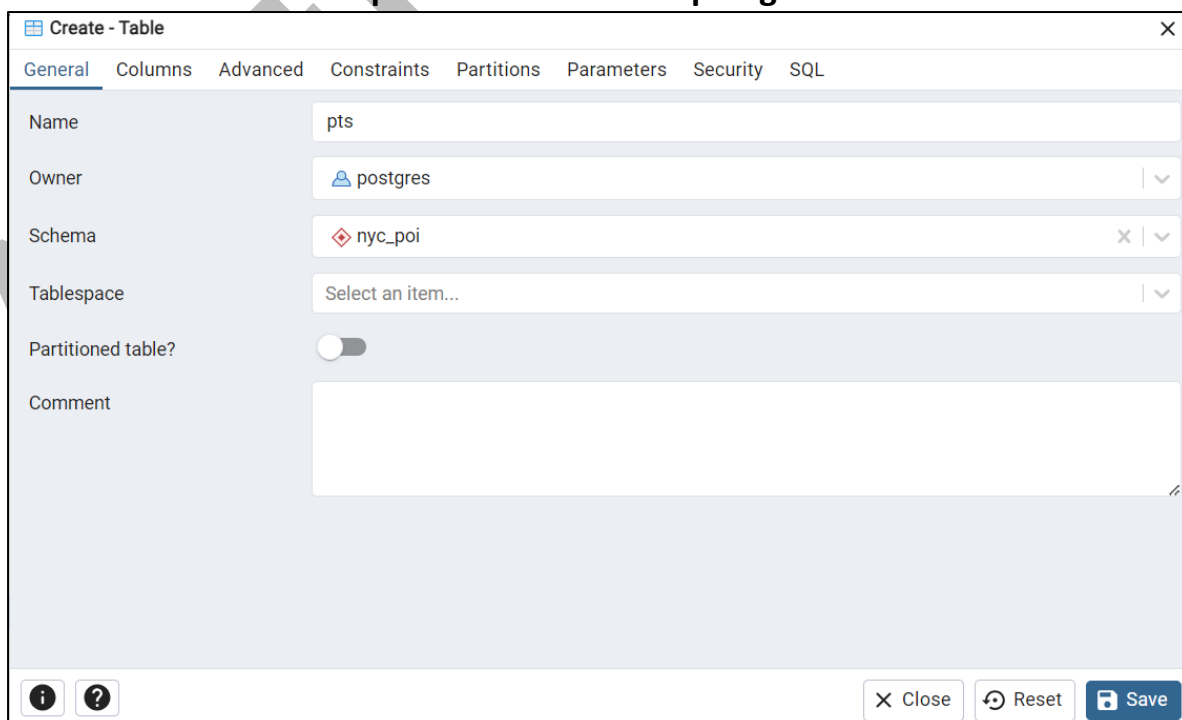
- b) Set the **Name** of the schema to **nyc\_poi** and its **Owner** to **postgres**, then click **Save** to create the schema.



- c) Expand the object listing associated with the **nyc\_poi** schema.
- d) Right-click on the **Tables** node, and select **Create > Table**.



- e) Set the table's **Name** to **pts** and its **Owner** to **postgres**.



- f) Under the **Columns** tab, click the + button.
- g) For the new column, set the **Name** to **gid** (geometry ID) and the **Data type** to **serial**. This data type is roughly equivalent to the AutoNumber type we saw in Access. Define this column as the table's **Primary key**.
- h) Repeat the previous step to create a column called **name**. Set its **Data type** to **character varying** and its **Length** to **50**. This column should not be the Primary key.

The screenshot shows the 'Create - Table' dialog box with the 'Columns' tab selected. The 'Inherited from table(s)' dropdown is set to 'Select to inherit from...'. The 'Columns' table lists two columns:

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
gid	serial			<input type="checkbox"/>	<input checked="" type="checkbox"/>	
name	character varying	50		<input type="checkbox"/>	<input type="checkbox"/>	

At the bottom right, there are buttons for 'Close', 'Reset', and 'Save'.

- i) Click **Save** to dismiss the dialog and create the table. Before adding the geometry column to the table, let's recall the pgAdmin search path. It's not set to include the nyc\_poi schema, so let's do that first.
- j) Reset the search path by executing the following statement in a **Query** window

The screenshot shows a Query window with the following content:

```
Query Query History
1 SET search_path TO nyc_poi, public;

Data Output Messages Notifications
SET

Query returned successfully in 74 msec.
```

h) We first create an extension of postgis and then add the geometry column.  
(or else it throws an error)

```
3 CREATE EXTENSION postgis;  
4 SELECT AddGeometryColumn('nyc_poi','pts','geom',4269,'POINT',2);
```

Data Output Messages Notifications

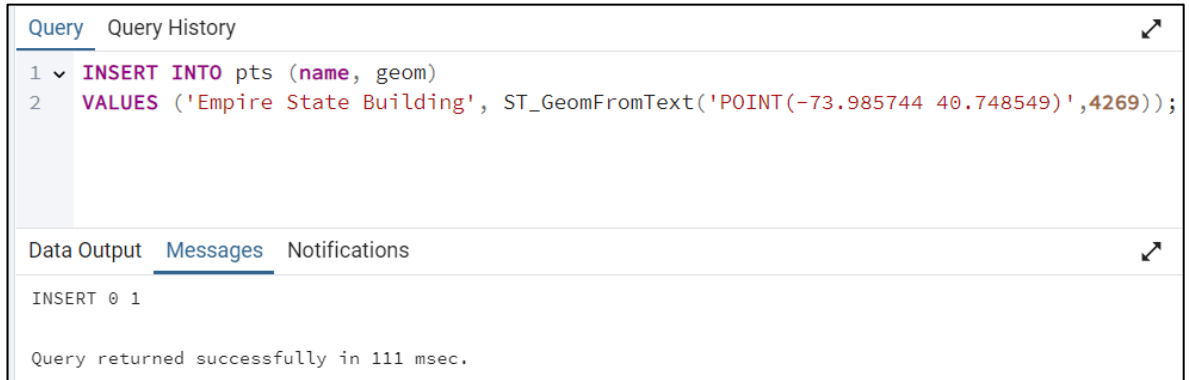
SQL

addgeometrycolumn  
text

1 nyc\_poi.pts.geom SRID:4269 TYPE:POINT DIMS:2

## 2. Add Rows to the Spatial Table:

- a) Insert Rows into the pts Table: Open a new Query Tool window in pgAdmin. Execute the following **INSERT** statement to add the first point:



The screenshot shows the pgAdmin Query Tool interface. The 'Query' tab is active, displaying the following SQL statement:

```
1 INSERT INTO pts (name, geom)
2 VALUES ('Empire State Building', ST_GeomFromText('POINT(-73.985744 40.748549)',4269));
```

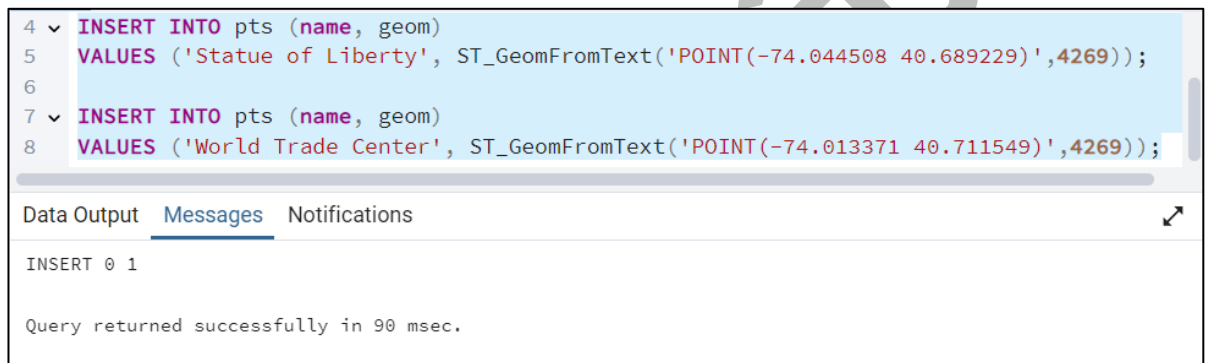
Below the query editor, the 'Messages' tab is active, showing the execution result:

```
INSERT 0 1
```

At the bottom, a status message reads: "Query returned successfully in 111 msec."

*This statement uses the ST\_GeomFromText() function to convert the specified point coordinates (longitude and latitude) into a geometry format, using SRID 4269.*

- b) **Add More Rows:** Execute additional INSERT statements to add more landmarks:



The screenshot shows the pgAdmin Query Tool interface with two INSERT statements:

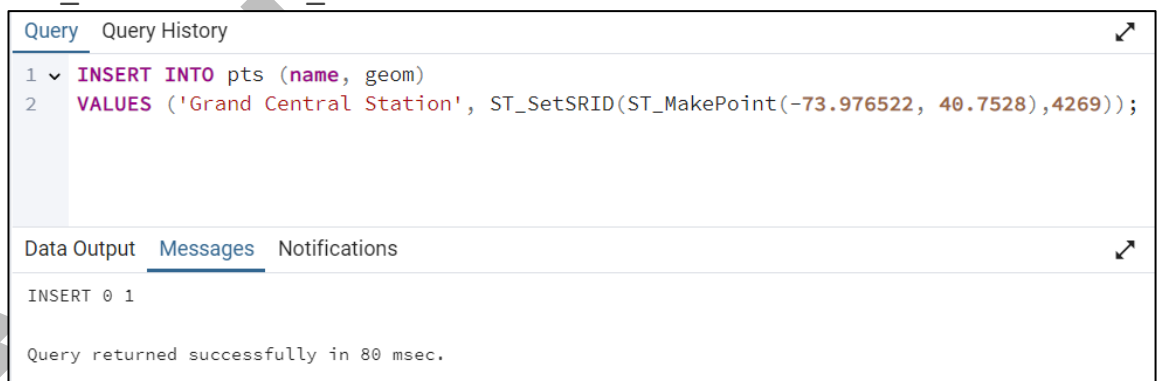
```
4 INSERT INTO pts (name, geom)
5 VALUES ('Stature of Liberty', ST_GeomFromText('POINT(-74.044508 40.689229)',4269));
6
7 INSERT INTO pts (name, geom)
8 VALUES ('World Trade Center', ST_GeomFromText('POINT(-74.013371 40.711549)',4269));
```

The 'Messages' tab shows the execution result:

```
INSERT 0 1
```

The status message reads: "Query returned successfully in 90 msec."

- c) Use ST\_SetSRID and ST\_MakePoint:



The screenshot shows the pgAdmin Query Tool interface with the following SQL statement:

```
1 INSERT INTO pts (name, geom)
2 VALUES ('Grand Central Station', ST_SetSRID(ST_MakePoint(-73.976522, 40.7528),4269));
```

The 'Messages' tab shows the execution result:

```
INSERT 0 1
```

The status message reads: "Query returned successfully in 80 msec."



d) Insert Multiple Rows in a Single Statement:

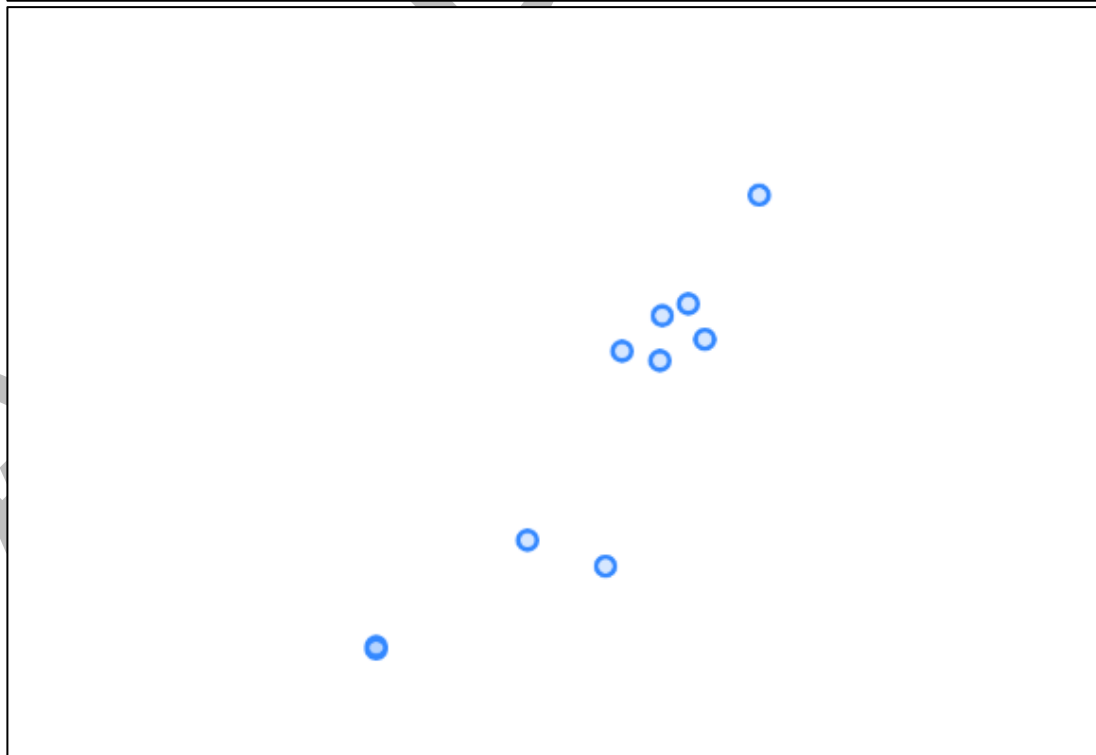
```
Query Query History
1 INSERT INTO pts (name, geom)
2 VALUES ('Radio City Music Hall', ST_GeomFromText('POINT(-73.97988 40.760171)',4269)),
3 ('Madison Square Garden', ST_GeomFromText('POINT(-73.993544 40.750541)',4269));

Data Output Messages Notifications
INSERT 0 2

Query returned successfully in 81 msec.
```

e) Confirm the Insertions: To confirm that the INSERT statements executed successfully, **right click on the pts table in the pgAdmin window and select View/Edit Data > All Rows**. This will allow you to view the data you've inserted into the table.

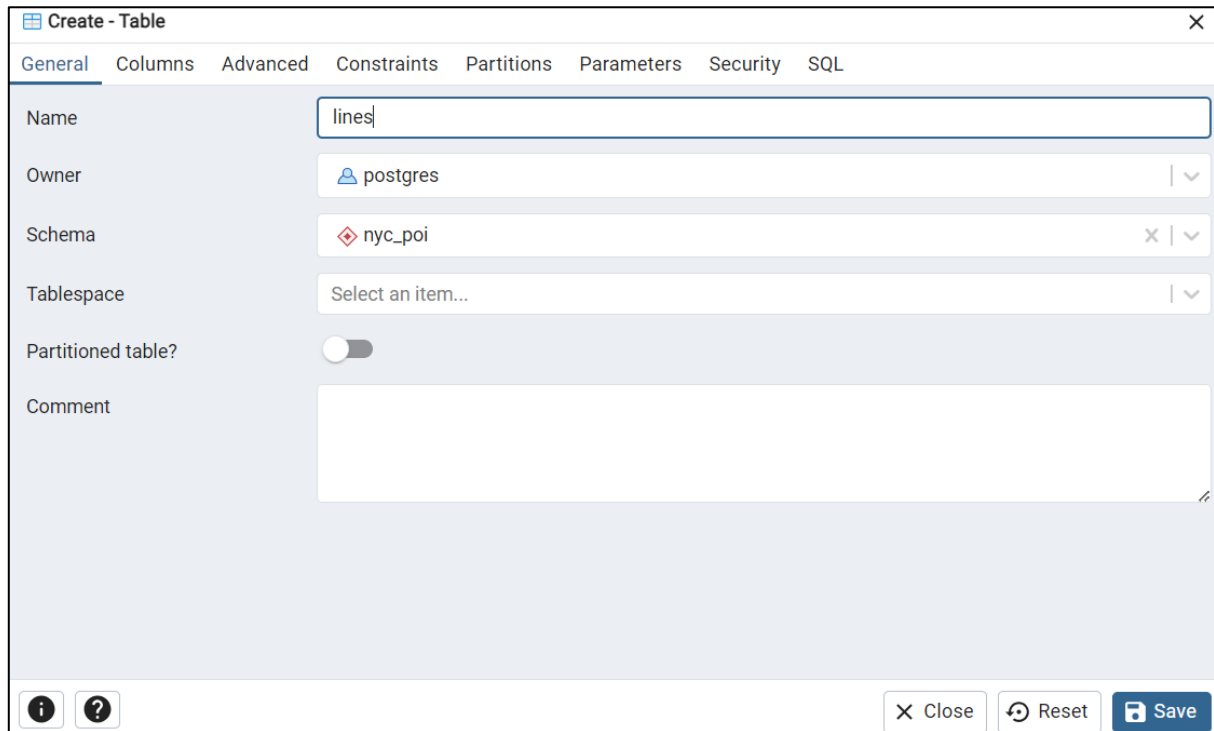
Data Output Messages Notifications			
<div>SQL</div>			
	gid [PK] integer	name character varying (50)	geom geometry
1	1	Empire State Building	0101000020AD1000009B8E006E167F52C00C3A2174D05F4440
2	2	Statue of Liberty	0101000020AD100000431A1538D98252C0363AE7A738584440
3	3	World Trade Center	0101000020AD100000C0ED0912DB8052C03140A209145B4440
4	4	Grand Central Station	0101000020AD100000C57421567F7E52C0E3361AC05B604440
5	5	Radio City Music Hall	0101000020AD10000049809A5AB67E52C00E2F88484D614440
6	6	Madison Square Garden	0101000020AD100000C5C89239967F52C050A73CBA11604440



Geom output of the same

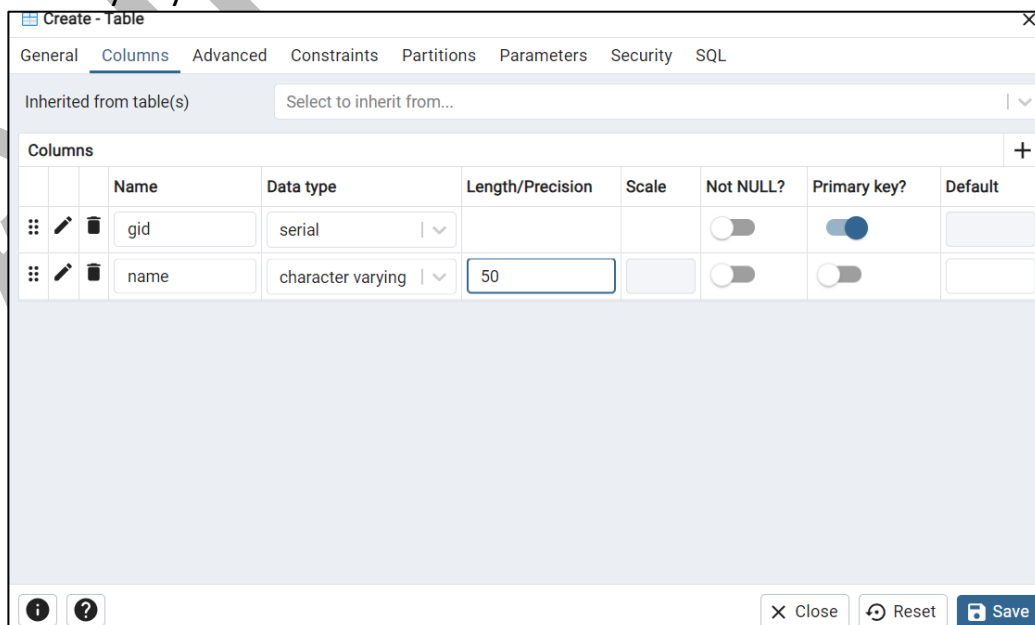
### 3. Create and Populate a Table of Linestrings:

- a) Create a New Table for **Linestrings**: In the **nyc\_poi** schema, create a new table to hold line features. Follow these steps in pgAdmin:
- Right-click on the Tables node within the nyc\_poi schema and select Create > Table.
  - Name the table **lines** and set the Owner to **postgres**



The 'Create - Table' dialog box in pgAdmin, General tab. The Name field is 'lines'. The Owner is 'postgres'. The Schema is 'nyc\_poi'. The Tablespace is 'Select an item...'. The Partitioned table? toggle is off. The Comment field is empty. The bottom bar has 'Close', 'Reset', and 'Save' buttons.

- b) Under the **Columns** tab, click the + button.
- c) For the new column, set the **Name** to **gid** (geometry ID) and the **Data type** to **serial**. This data type is roughly equivalent to the AutoNumber type we saw in Access. Define this column as the table's **Primary key**.
- d) Repeat the previous step to create a column called **name**. Set its **Data type** to **character varying** and its **Length** to **50**. This column should not be the Primary key.



The 'Create - Table' dialog box in pgAdmin, Columns tab. The 'Inherited from table(s)' field is 'Select to inherit from...'. The Columns table has two rows: 'gid' with data type 'serial' and 'name' with data type 'character varying' and length '50'. The 'gid' column is marked as the Primary key. The bottom bar has 'Close', 'Reset', and 'Save' buttons.

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	gid	serial			<input type="checkbox"/>	<input checked="" type="checkbox"/>	
	name	character varying	50		<input type="checkbox"/>	<input type="checkbox"/>	

- e) Add the Geometry Column Using **AddGeometryColumn()**: Before adding the geometry column, ensure that the search path is set correctly.

```
3 SELECT AddGeometryColumn('nyc_poi','lines','geom',4269,'LINESTRING',2);
4
```

Data Output Messages Notifications

addgeometrycolumn  
text

1	nyc_poi.lines.geom SRID:4269 TYPE:LINESTRING DIMS:2
---	---

- f) **Insert Rows** into the lines Table: Add lines representing various notable road or bridge features in nyc\_poi using **INSERT** statements:

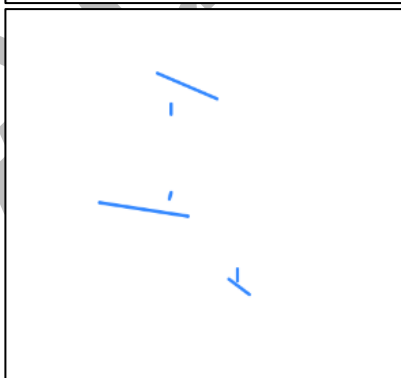
Query Query History

```
1 INSERT INTO lines (name, geom)
2 VALUES ('Holland Tunnel',ST_GeomFromText('LINESTRING(
3 -74.036486 40.730121,
4 -74.03125 40.72882,
5 -74.011123 40.725958)',4269)),
6 ('Lincoln Tunnel',ST_GeomFromText('LINESTRING(
7 -74.019921 40.767119,
8 -74.002841 40.759773)',4269)),
9 ('Brooklyn Bridge',ST_GeomFromText('LINESTRING(
10 -73.99945 40.708231,
11 -73.9937 40.703676)',4269)));
```

Data Output Messages Notifications

INSERT 0 3

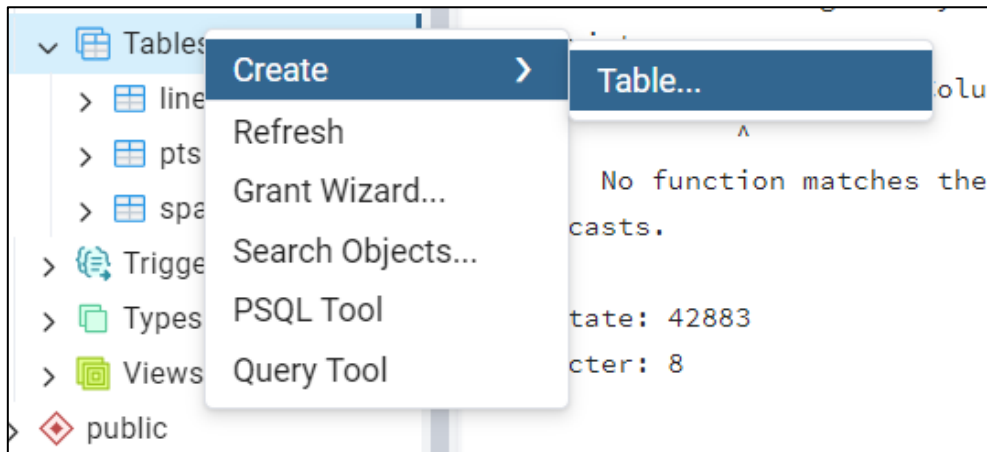
Query returned successfully in 73 msec.



GEOM output of the lines table.

## 4. Create and Populate a Table of Polygons:

- a) Create a New Table for **Polygon**: In the **nyc\_poi** schema, create a new table to hold polygon features. Follow these steps in pgAdmin:
- Right-click on the Tables node within the **nyc\_poi** schema and select Create > Table.
  - Name the table **polys** and set the Owner to **postgres**

A screenshot of the 'Create - Table' dialog box in pgAdmin. The 'General' tab is selected. The fields are: 'Name' (polys), 'Owner' (postgres), 'Schema' (nyc\_poi), 'Tablespace' (Select an item...), 'Partitioned table?' (toggle off), and 'Comment' (empty text area). At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

- b) Under the **Columns** tab, click the + button.
- c) For the new column, set the **Name** to **gid** (geometry ID) and the **Data type** to **serial**. This data type is roughly equivalent to the AutoNumber type we saw in Access. Define this column as the table's **Primary key**.
- d) Repeat the previous step to create a column called **name**. Set its **Data type** to **character varying** and its **Length** to **50**. This column should not be the Primary key.

Create - Table

General Columns Advanced Constraints Partitions Parameters Security SQL

Inherited from table(s) Select to inherit from...

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
	gid	serial			<input type="checkbox"/>	<input checked="" type="checkbox"/>		
	name	character varying	50		<input type="checkbox"/>	<input type="checkbox"/>		

Close Reset Save

- g) Add the Geometry Column Using **AddGeometryColumn()**: Before adding the geometry column, ensure that the search path is set correctly.

```

1 SET search_path to nyc_poi, public;
2 SELECT AddGeometryColumn('nyc_poi', 'polys', 'geom', 4269, 'POLYGON', 2);

```

Data Output Messages Notifications

addgeometrycolumn text

1 nyc\_poi.polys.geom SRID:4269 TYPE:POLYGON DIMS:2

- h) **Insert Rows** into the lines Table: Add lines representing various notable road or bridge features in nyc\_poi using **INSERT** statements:

Query Query History

```

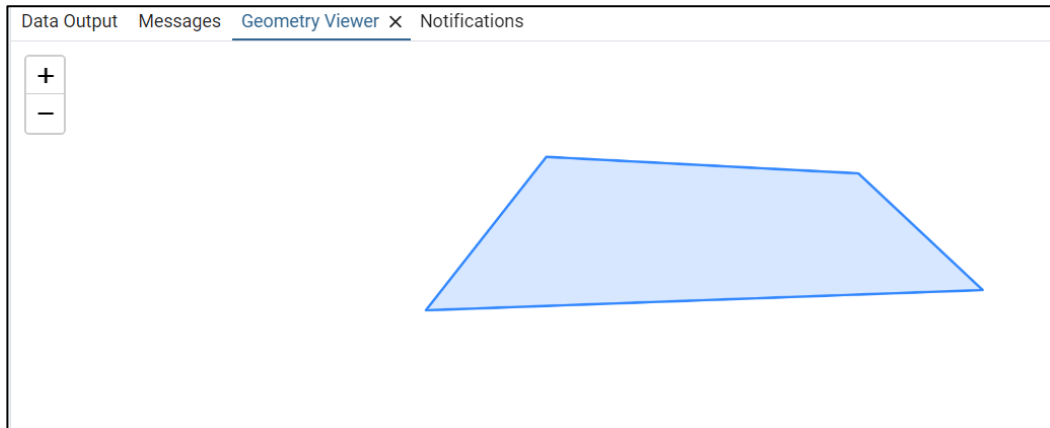
1 INSERT INTO polys (name, geom)
2 VALUES ('Central Park', ST_GeomFromText('POLYGON((
3 -73.973057 40.764356,
4 -73.981898 40.768094,
5 -73.958209 40.800621,
6 -73.949282 40.796853,
7 -73.973057 40.764356))', 4269));

```

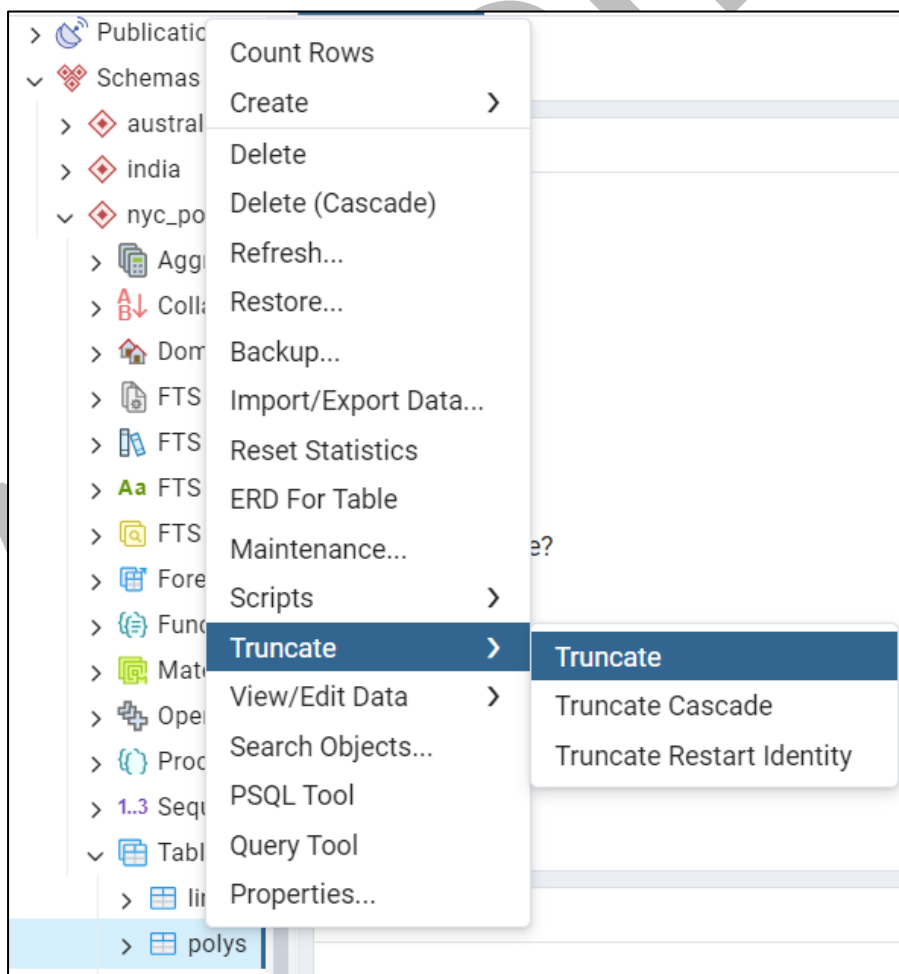
Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 81 msec.



- The first X/Y (lon/lat) pair should be the same as the last (to close the polygon).
- Note that the coordinate list is enclosed in an additional set of parentheses. This set of parentheses is required because polygons are actually composed of potentially multiple rings. Every polygon has a ring that defines its exterior. Some polygons also have additional rings that define holes in the interior. When constructing a polygon with holes, the exterior ring is supplied first, followed by the interior rings. Each ring is enclosed in a set of parentheses, and the rings are separated by commas. To see an example, let's add Central Park again, this time cutting out the large reservoir near its center.
- First, let's remove the original Central Park row. In pgAdmin, right-click on the **polys** table and select **Truncate > Truncate**. Note that this deletes all rows from the table.



- Add Central Park (minus the reservoir) back into the table using this statement.

```
SET search_path TO nyc_poi, public;
```

Query Query History

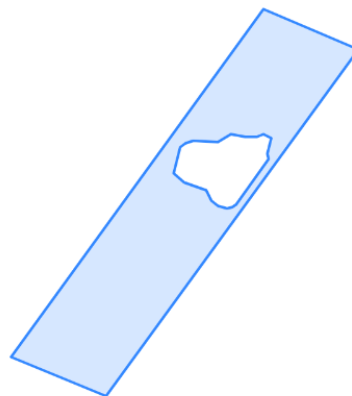
```
1  ✓ INSERT INTO polys(name, geom)
2  VALUES ('Central Park',ST_GeomFromText('POLYGON((
3  -73.973057 40.764356,-73.981898 40.768094,
4  -73.958209 40.800621,-73.949282 40.796853,
5  -73.973057 40.764356),(-73.966681 40.785221,
6  -73.966058 40.787674,-73.965586 40.788064,
7  -73.9649 40.788291,-73.963913 40.788194,
8  -73.963333 40.788291,-73.962539 40.788259,
9  -73.962153 40.788389,-73.96181 40.788714,
10 -73.961359 40.788909,-73.960887 40.788925,
11 -73.959986 40.788649,-73.959492 40.788649,
12 -73.958913 40.78873,-73.958269 40.788974,
13 -73.957797 40.788844,-73.957497 40.788568,
14 -73.957497 40.788259,-73.957776 40.787739,
15 -73.95784 40.787057,-73.957819 40.786569,
16 -73.960801 40.782394,-73.961145 40.78215,
17 -73.961638 40.782036,-73.962518 40.782199,
18 -73.963076 40.78267,-73.963677 40.783661,
19 -73.965694 40.784457,-73.966681 40.785221)
20 ),4269));
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 100 msec.

Data Output Messages Geometry Viewer x Notifications



## 5) 3D and 4D Geometries:

Creating tables for 3d, 4d geometries :

Query	Query History
1	--ADITI CHHAJED
2	create table geo_m (
3	id SERIAL PRIMARY KEY,
4	geom GEOMETRY(POINTM,4326)
5	)

Data Output	Messages	Notifications
CREATE TABLE		
Query returned successfully in 130 msec.		

Query	Query History
1	--ADITI CHHAJED
2	create table geo_z (
3	id SERIAL PRIMARY KEY,
4	geom GEOMETRY(POINTZ,4326)
5	)

Data Output	Messages	Notifications
CREATE TABLE		
Query returned successfully in 114 msec.		

Query	Query History
1	--ADITI CHHAJED
2	create table geo_4d (
3	id SERIAL PRIMARY KEY,
4	geom GEOMETRY(POINTZM,4326)
5	)

Data Output	Messages	Notifications
CREATE TABLE		
Query returned successfully in 111 msec.		



Query

Query History

1

--ADITI CHHAJED

2

▼

INSERT INTO geo\_m(geom)

3

VALUES (ST\_GeomFromText('POINTM(-73.985744 40.748549 9999)',4326));

4

5

▼

INSERT INTO geo\_z(geom)

6

VALUES (ST\_GeomFromText('POINT(-73.9 40.7 190)',4326));

7

8

▼

INSERT INTO geo\_4d(geom)

9

VALUES (ST\_GeomFromText('POINT(-73.9 40.7 9999)',4326));

Data Output

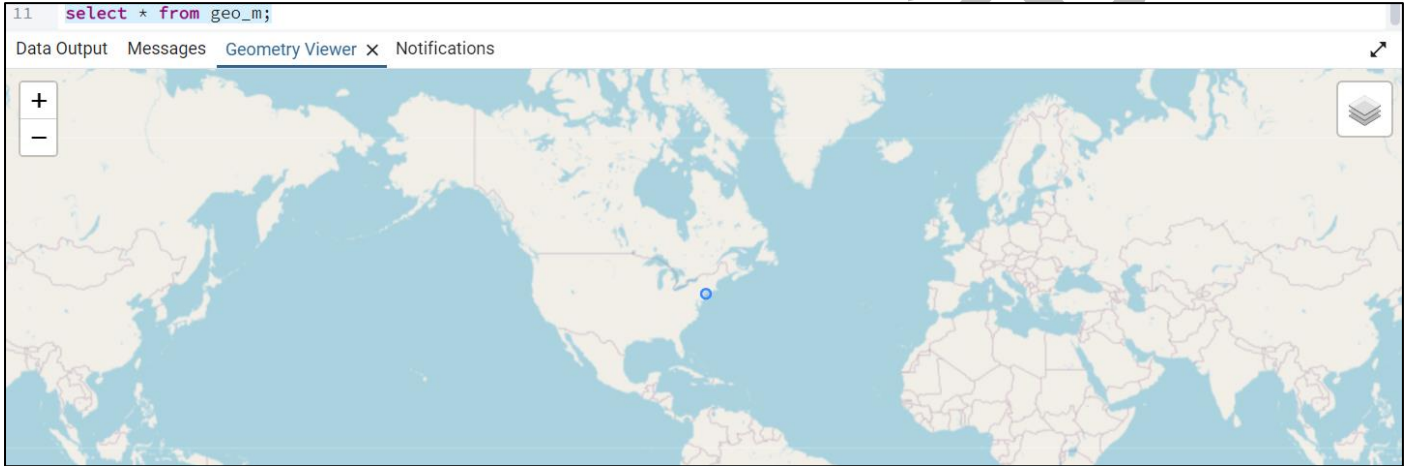
Messages

Notifications

INSERT 0 1

Query returned successfully in 119 msec.

Geom outputs -



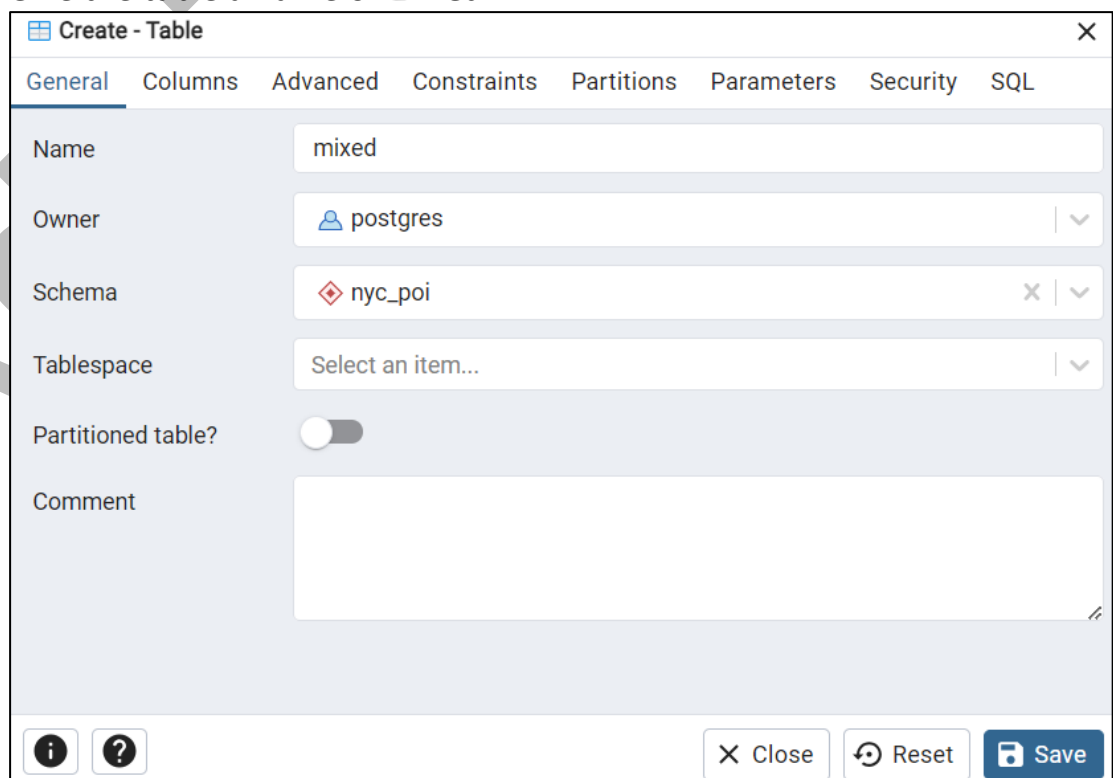
## 6) Multipart Geometries:

- i. PostGIS provides support for features with Multiple parts through the **MULTIPOINT**, **MULTILINESTRING**, and **MULTIPOLYGON** data types
- ii. Let us see an example of **Multipart Polygon** (MULTIPOLYGON): A MULTIPOLYGON can represent multiple disconnected areas, such as groups of islands or regions.
- iii. The footprints of the World Trade Center Towers 1 and 2 (now fountains in the 9/11 Memorial) can be represented as a single multipart polygon as follows:

```
INSERT INTO polys (name, geom)
VALUES (
  'World Trade Center ',
  ST_GeomFromText(
    'MULTIPOLYGON(
      (
        (-73.973057 40.764356, -73.981898 40.768094,
        -73.958209 40.800621, -73.949282 40.796853, -73.973057 40.764356)
      ),
      (
        (-73.981423 40.737509, -73.987898 40.742567,
        -73.965157 40.759423, -73.960678 40.752134, -73.981423 40.737509)
      )
    )',
    4269
  )
);
```

## 7) Mixing Geometries:

- a) Create the **mixed** Table in the **nyc\_poi** Schema:
  - Give the table a name of **mixed**.



**Create - Table**

**General** Columns Advanced Constraints Partitions Parameters Security SQL

Name: mixed

Owner: postgres

Schema: nyc\_poi

Tablespace: Select an item...

Partitioned table?: ☐

Comment:

**Close** **Reset** **Save**

- The table should have the same column definitions, with the exception that the geometry type should be set to GEOMETRY

Create - Table

General

Columns

Advanced

Constraints

Partitions

Parameters

Security

SQL

Inherited from

Select to inherit from...

table(s)

Columns

+

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary
<div></div> <div></div> <div></div>	gid	serial			<input type="checkbox"/>	<input checked="" type="checkbox"/>
<div></div> <div></div> <div></div>	name	character vary			<input type="checkbox"/>	<input type="checkbox"/>
<div></div> <div></div> <div></div>		Select an item...			<input type="checkbox"/>	<input type="checkbox"/>

?

?

Close

Reset

Save

Query

Query History

```

1 SET search_path to nyc_poi, public;
2 SELECT AddGeometryColumn('nyc_poi','mixed','geom',4269,'GEOMETRY',2);

```

Data Output

Messages

Notifications

SQL

	addgeometrycolumn	
	text	
1	nyc_poi.mixed.geom SRID:4269 TYPE:GEOMETRY DIMS:2	

b) Add the features to this new table:

Query Query History

```

1  ✓ INSERT INTO mixed (name, geom)
2  VALUES ('Empire State Building', ST_GeomFromText('POINT(-73.985744 40.748549)',4269)),
3  ('Statue of Liberty', ST_GeomFromText('POINT(-74.044508 40.689229)',4269)),
4  ('World Trade Center', ST_GeomFromText('POINT(-74.013371 40.711549)',4269)),
5  ('Radio City Music Hall', ST_GeomFromText('POINT(-73.97988 40.760171)',4269)),
6  ('Madison Square Garden', ST_GeomFromText('POINT(-73.993544 40.750541)',4269)),
7  ('Holland Tunnel',ST_GeomFromText('LINESTRING(
8  -74.036486 40.730121,
9  -74.03125 40.72882,
10 -74.011123 40.725958)',4269)),
11 ('Lincoln Tunnel',ST_GeomFromText('LINESTRING(
12 -74.019921 40.767119,
13 -74.002841 40.759773)',4269)),
14 ('Brooklyn Bridge',ST_GeomFromText('LINESTRING(
15 -73.99945 40.708231,
16 -73.9937 40.703676)',4269)),
17 ('Central Park',ST_GeomFromText('POLYGON((
18 -73.973057 40.764356,
19 -73.981898 40.768094,
20 -73.958209 40.800621,
21 -73.949282 40.796853,
22 -73.973057 40.764356)',4269));

```

Data Output Messages Notifications

INSERT 0 9

Query returned successfully in 119 msec.

- c) In the pgAdmin window right-click on the **mixed** table and select **View/Edit Data > All Rows** to confirm that the INSERT statement executed properly.

Data Output Messages Notifications			
SQL			
	gid [PK] integer	name character varying	geom geometry
1	1	Empire State Building	0101000020AD1000009B8E006E167F52C00C3A2174D05F4440
2	2	Statue of Liberty	0101000020AD100000431A1538D98252C0363AE7A738584440
3	3	World Trade Center	0101000020AD100000C0ED0912DB8052C03140A209145B4440
4	4	Radio City Music Hall	0101000020AD10000049809A5AB67E52C00E2F88484D614440
5	5	Madison Square Garden	0101000020AD100000C5C89239967F52C050A73CBA11604440
6	6	Holland Tunnel	0102000020AD10000003000000C23060C9558252C0B88FDC9A745D4440000000008252C0DA5548F9495D4440F04E3E3DB68052C081221631EC5C4440
7	7	Lincoln Tunnel	0102000020AD100000020000003BE0BA62468152C0F39194F430624440A185048C2E8052C021B1DD3D40614440
8	8	Brooklyn Bridge	0102000020AD100000020000002EFF21FDF67F52C0B6813B50A75A44404182E2C7987F52C0747D1F0E125A4440
9	9	Central Park	0103000020AD10000001000000050000003509DE90467E52C02D40DB6AD66144408080B56AD77E52C07D2079E750624440E76ED74B537D52C0D38EB9BF7A6644409D2B4A09C17C5

## **Conclusion:**

*In this experiment, we **successfully created and managed spatial data in PostgreSQL using PostGIS, exploring schemas, spatial tables, and diverse geometry types like POINT, LINESTRING, POLYGON, 3D – (XYZ,XYM), 4D (XYZM), multipart and mixing geometries.***

*By organizing data into different schemas and tables with geometry columns, we gained a practical understanding of managing geographic features and executing **spatial operations, such as inserting and querying geographic data in structured formats.***

*Additionally, this experiment demonstrated how different geometry types can be stored and queried within a single table, providing flexibility and efficiency in handling complex spatial database management.*