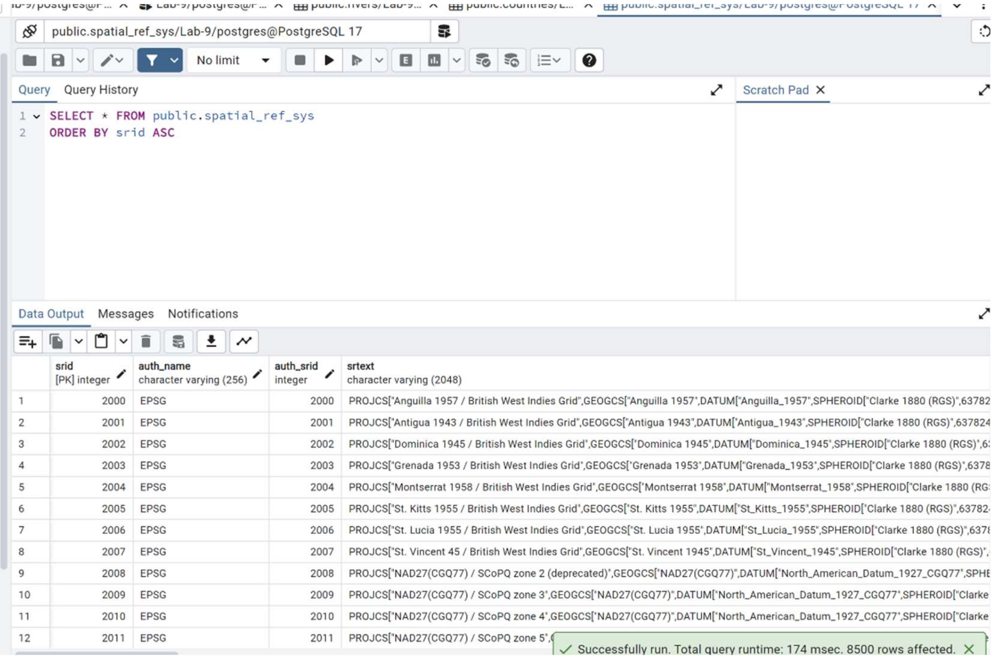
Ranjeet Gupta
SC24M138

Scientific Computing LAB 9 – SPATIAL DBMS

1. Spatial Queries

(a) Explore spatial_ref_sys table

spatial_ref_sys is a PostGIS table that contains information about different spatial reference systems (SRS).
The spatial_ref_sys table is a default system table in PostGIS, the spatial extension for PostgreSQL. When PostGIS is installed and enabled on our database, this table is automatically created as part of the PostGIS setup.



```
SELECT * FROM public.spatial_ref_sys
ORDER BY srid ASC
```

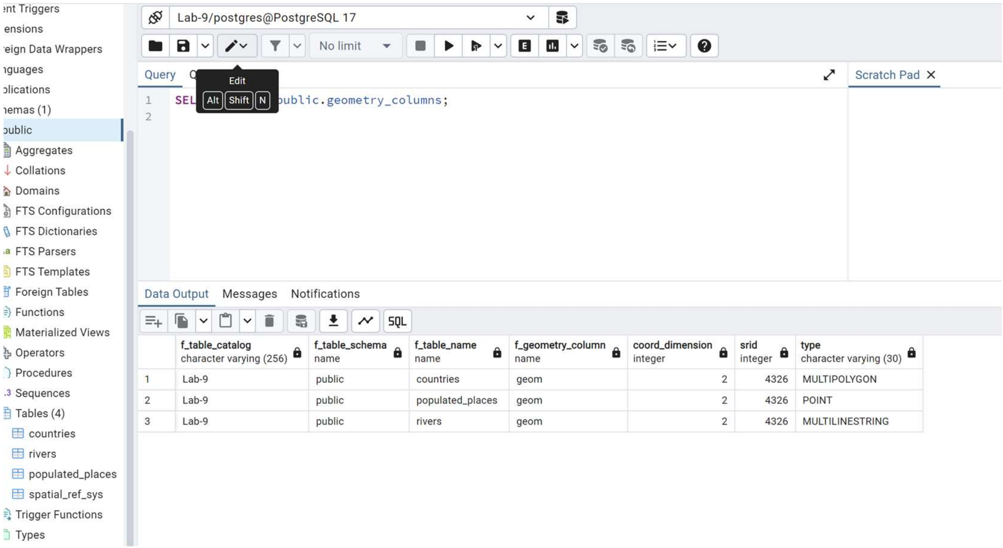
	srid [PK] integer	auth_name character varying (256)	auth_srid integer	srtext character varying (2048)
1	2000	EPSG	2000	PROJCS["Anguilla 1957 / British West Indies Grid",GEOGCS["Anguilla 1957",DATUM["Anguilla_1957",SPHEROID["Clarke 1880 (RGS)",63782
2	2001	EPSG	2001	PROJCS["Antigua 1943 / British West Indies Grid",GEOGCS["Antigua 1943",DATUM["Antigua_1943",SPHEROID["Clarke 1880 (RGS)",637824
3	2002	EPSG	2002	PROJCS["Dominica 1945 / British West Indies Grid",GEOGCS["Dominica 1945",DATUM["Dominica_1945",SPHEROID["Clarke 1880 (RGS)",6
4	2003	EPSG	2003	PROJCS["Grenada 1953 / British West Indies Grid",GEOGCS["Grenada 1953",DATUM["Grenada_1953",SPHEROID["Clarke 1880 (RGS)",6378
5	2004	EPSG	2004	PROJCS["Montserrat 1958 / British West Indies Grid",GEOGCS["Montserrat 1958",DATUM["Montserrat_1958",SPHEROID["Clarke 1880 (RG
6	2005	EPSG	2005	PROJCS["St. Kitts 1955 / British West Indies Grid",GEOGCS["St. Kitts 1955",DATUM["St_Kitts_1955",SPHEROID["Clarke 1880 (RGS)",63782
7	2006	EPSG	2006	PROJCS["St. Lucia 1955 / British West Indies Grid",GEOGCS["St. Lucia 1955",DATUM["St_Lucia_1955",SPHEROID["Clarke 1880 (RGS)",6378
8	2007	EPSG	2007	PROJCS["St. Vincent 45 / British West Indies Grid",GEOGCS["St. Vincent 1945",DATUM["St_Vincent_1945",SPHEROID["Clarke 1880 (RGS)",
9	2008	EPSG	2008	PROJCS["NAD27(CGQ77) / SCoPQ zone 2 (deprecated)",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHE
10	2009	EPSG	2009	PROJCS["NAD27(CGQ77) / SCoPQ zone 3",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHEROID["Clarke
11	2010	EPSG	2010	PROJCS["NAD27(CGQ77) / SCoPQ zone 4",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1927_CGQ77",SPHEROID["Clarke
12	2011	EPSG	2011	PROJCS["NAD27(CGQ77) / SCoPQ zone 5",

✓ Successfully run. Total query runtime: 174 msec. 8500 rows affected.

Also by, In pgAdmin, go to Schemas > public > Tables > spatial_ref_sys, right-click, and select "View/Edit Data > All rows" to browse the table's contents.

(b) Explore geometry columns (under Views) Geometry of all data type

The geometry_columns view in PostGIS shows metadata about spatial tables with geometry data types. You can find this under Views in pgAdmin (Schemas > public > Views > geometry_columns).
You can also query it directly:



```
SEL public.geometry_columns;
```

	f_table_catalog character varying (256)	f_table_schema name	f_table_name name	f_geometry_column name	coord_dimension integer	srid integer	type character varying (30)
1	Lab-9	public	countries	geom	2	4326	MULTIPOLYGON
2	Lab-9	public	populated_places	geom	2	4326	POINT
3	Lab-9	public	rivers	geom	2	4326	MULTILINESTRING

This shows tables containing spatial data, including each table's geometry type (e.g., POINT, LINESTRING, POLYGON).

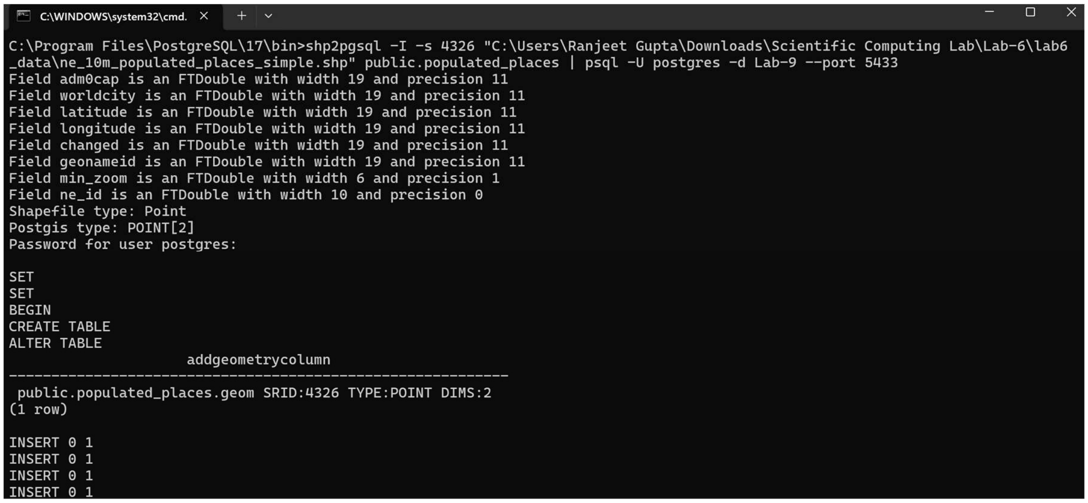
(c) LAB-6 Use the shapefiles given (countries, river, and populated places) for the following query

(d) Import the shape files into Pgadmin

Install shp2pgsql Tool: Ensure the shp2pgsql tool (part of the PostGIS suite) is installed. It's used to convert .shp files into SQL format.

Convert the Shapefiles: Open a command line and use shp2pgsql to convert each shapefile into an SQL file.

CMD:



```
C:\Program Files\PostgreSQL\17\bin>shp2pgsql -I -s 4326 "C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-6\lab6
_data\ne_10m_populated_places_simple.shp" public.populated_places | psql -U postgres -d Lab-9 --port 5433
Field adm0cap is an FTDouble with width 19 and precision 11
Field worldcity is an FTDouble with width 19 and precision 11
Field latitude is an FTDouble with width 19 and precision 11
Field longitude is an FTDouble with width 19 and precision 11
Field changed is an FTDouble with width 19 and precision 11
Field geonameid is an FTDouble with width 19 and precision 11
Field min_zoom is an FTDouble with width 6 and precision 1
Field ne_id is an FTDouble with width 10 and precision 0
Shapefile type: Point
Postgis type: POINT[2]
Password for user postgres:

SET
SET
BEGIN
CREATE TABLE
ALTER TABLE
                     addgeometrycolumn
-----------------------------------------------------------
 public.populated_places.geom SRID:4326 TYPE:POINT DIMS:2
(1 row)

INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```

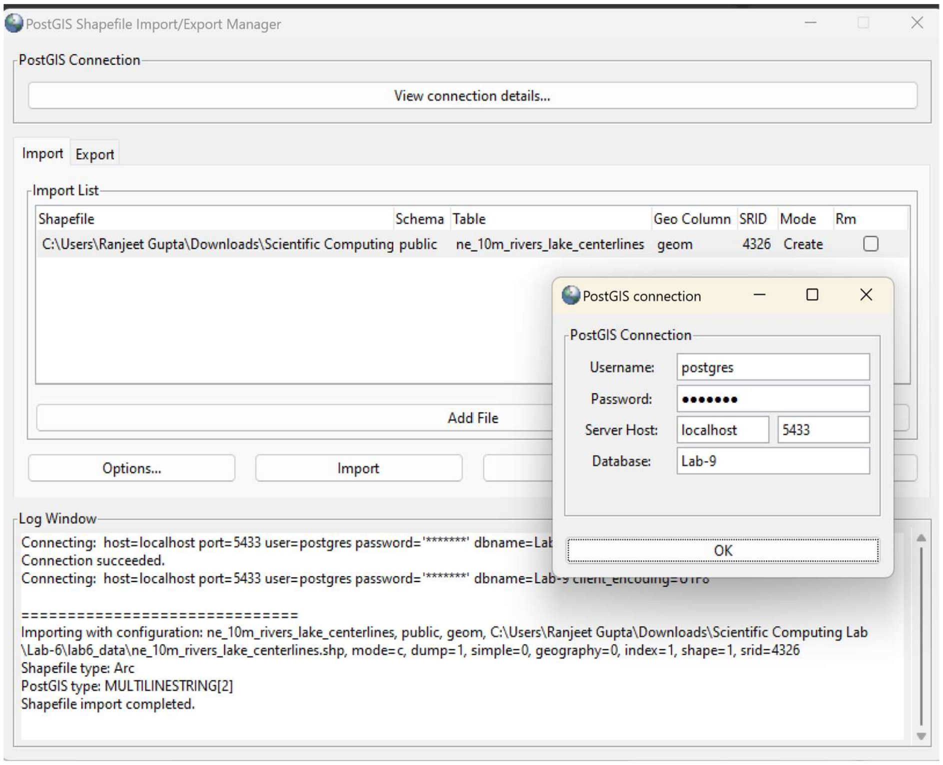
```
C:\Program Files\PostgreSQL\17\bin>shp2pgsql -I -s 4326 "C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-6\lab6_data\ne_10m_admin_0_countries.shp" public.countries | psql -U postgres -d Lab-9 --port 5433
```

-I: Creates a spatial index on the geometry column for faster queries.

-s SRID: Specifies the Spatial Reference ID (SRID) for the shapefile. Replace SRID with the correct SRID code (e.g., 4326 for WGS84).

public.countries: Name of the schema and table where the data will be stored.

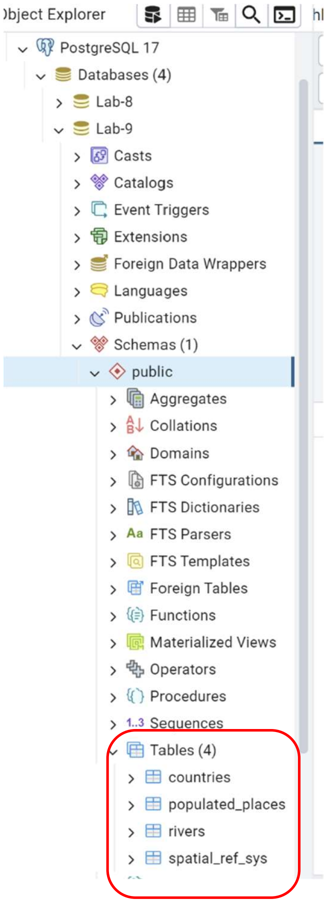
Another method is to import shapefile into Pgadmin through their own postgis GUI PostgreSQL > 17 > bin > postgisgui > shp2pgsql-gui.exe



Shapefile	Schema	Table	Geo Column	SRID	Mode	Rm
C:\Users\Ranjeet Gupta\Downloads\Scientific Computing	public	ne_10m_rivers_lake_centerlines	geom	4326	Create	□

Username: postgres
Password: •••••••
Server Host: localhost 5433
Database: Lab-9

```
Connecting:  host=localhost port=5433 user=postgres password='*******' dbname=Lab
Connection succeeded.
Connecting:  host=localhost port=5433 user=postgres password='*******' dbname=Lab-9 client_encoding=UTF8

==============================
Importing with configuration: ne_10m_rivers_lake_centerlines, public, geom, C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab
\Lab-6\lab6_data\ne_10m_rivers_lake_centerlines.shp, mode=c, dump=1, simple=0, geography=0, index=1, shape=1, srid=4326
Shapefile type: Arc
PostGIS type: MULTILINESTRING[2]
Shapefile import completed.
```



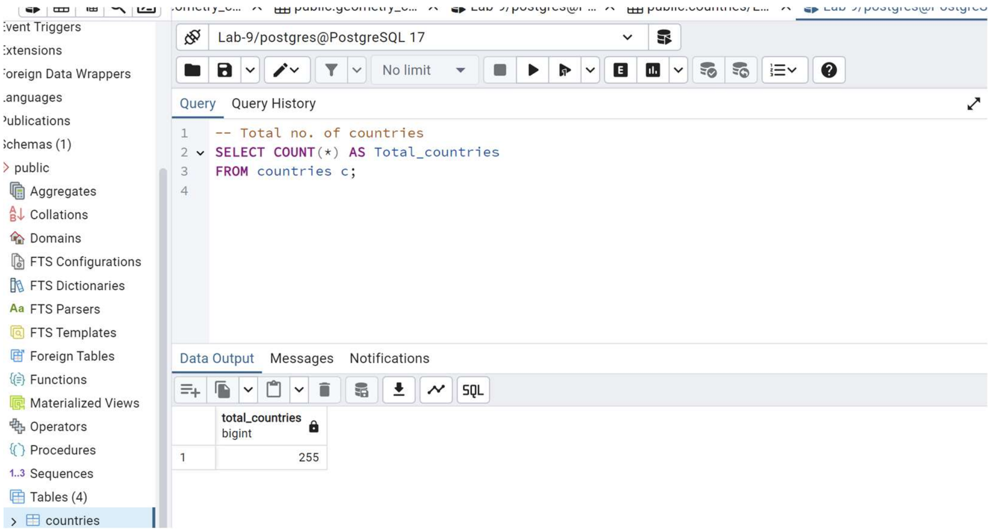
Verify the Import in pgAdmin
Open pgAdmin and check if the tables (countries, rivers, populated_places) were created successfully.
Navigate to Schemas > public > Tables and confirm the tables contain the spatial data from the shapefiles.

2. Explore the attribute

Once the shapefiles are imported, use SELECT * FROM table_name LIMIT 10; to explore the first few rows and understand attributes in each table.

3. Write SQL queries

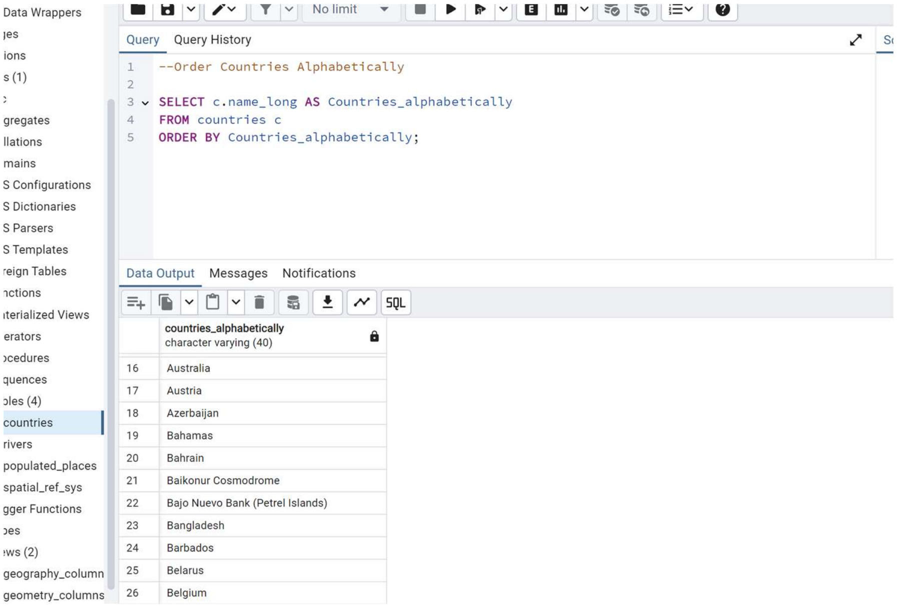
1) Find the total number of countries and order it alphabetically Later, display the names in such a way that countries get grouped alphabetically.



```
-- Total no. of countries
SELECT COUNT(*) AS Total_countries
FROM countries c;
```

	total_countries bigint
1	255

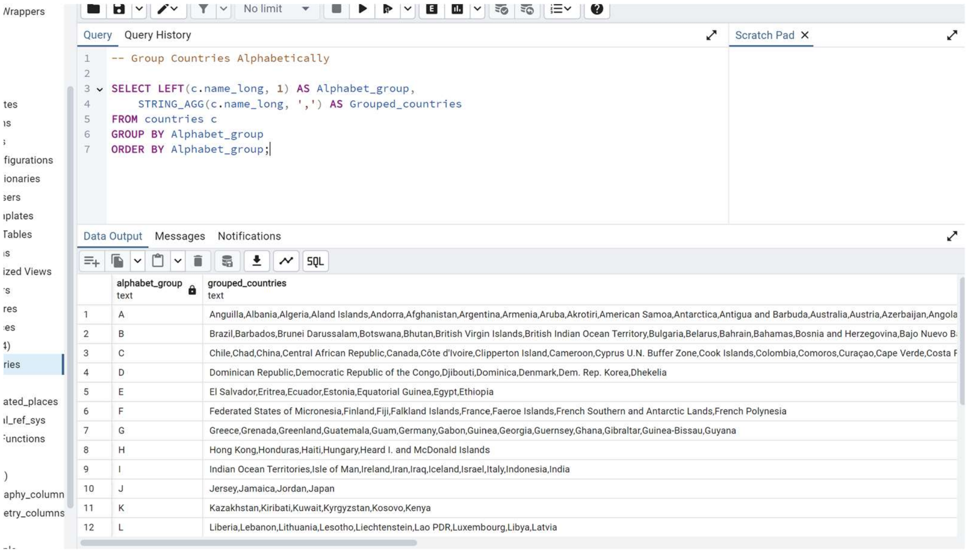
This query returns the total number of rows in the countries table, which corresponds to the number of countries.



```
--Order Countries Alphabetically

SELECT c.name_long AS Countries_alphabetically
FROM countries c
ORDER BY Countries_alphabetically;
```

	countries_alphabetically character varying (40)
16	Australia
17	Austria
18	Azerbaijan
19	Bahamas
20	Bahrain
21	Baikonur Cosmodrome
22	Bajo Nuevo Bank (Petrel Islands)
23	Bangladesh
24	Barbados
25	Belarus
26	Belgium

This query lists all the country names in alphabetical order.

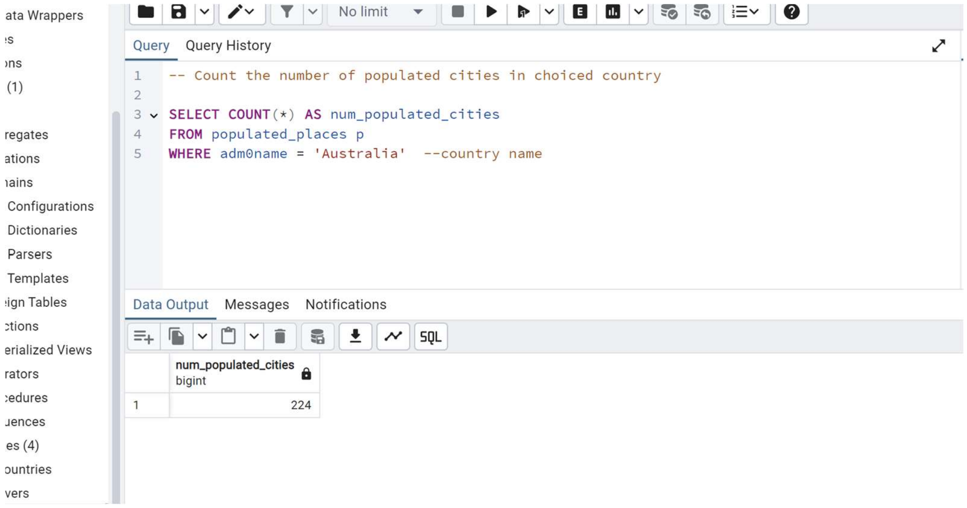


```sql
-- Group Countries Alphabetically

SELECT LEFT(c.name_long, 1) AS Alphabet_group,
    STRING_AGG(c.name_long, ',') AS Grouped_countries
FROM countries c
GROUP BY Alphabet_group
ORDER BY Alphabet_group;
```

	alphabet_group text	grouped_countries text
1	A	Anguilla,Albania,Algeria,Aland Islands,Andorra,Afghanistan,Argentina,Armenia,Aruba,Akrotiri,American Samoa,Antarctica,Antigua and Barbuda,Australia,Austria,Azerbaijan,Angola
2	B	Brazil,Barbados,Brunei Darussalam,Botswana,Bhutan,British Virgin Islands,British Indian Ocean Territory,Bulgaria,Belarus,Bahrain,Bahamas,Bosnia and Herzegovina,Bajo Nuevo B
3	C	Chile,Chad,China,Central African Republic,Canada,Côte d'Ivoire,Clipperton Island,Cameroon,Cyprus U.N. Buffer Zone,Cook Islands,Colombia,Comoros,Curaçao,Cape Verde,Costa F
4	D	Dominican Republic,Democratic Republic of the Congo,Djibouti,Dominica,Denmark,Dem. Rep. Korea,Dhekelia
5	E	El Salvador,Eritrea,Ecuador,Estonia,Equatorial Guinea,Egypt,Ethiopia
6	F	Federated States of Micronesia,Finland,Fiji,Falkland Islands,France,Faeroe Islands,French Southern and Antarctic Lands,French Polynesia
7	G	Greece,Grenada,Greenland,Guatemala,Guam,Germany,Gabon,Guinea,Georgia,Guernsey,Ghana,Gibraltar,Guinea-Bissau,Guyana
8	H	Hong Kong,Honduras,Haiti,Hungary,Heard I. and McDonald Islands
9	I	Indian Ocean Territories,Isle of Man,Ireland,Iran,Iraq,Iceland,Israel,Italy,Indonesia,India
10	J	Jersey,Jamaica,Jordan,Japan
11	K	Kazakhstan,Kiribati,Kuwait,Kyrgyzstan,Kosovo,Kenya
12	L	Liberia,Lebanon,Lithuania,Lesotho,Liechtenstein,Lao PDR,Luxembourg,Libya,Latvia

LEFT(country_name, 1) : extracts the first letter of the country name.
STRING_AGG(country_name, ', ') combines all country names starting with the same letter into a single string, separated by commas.
GROUP BY LEFT(country_name, 1) groups countries based on their starting letter.
ORDER BY Alphabet_group ensures the grouped result is displayed alphabetically.

2) Find the number of populated cities within your choice of country(excluding India) listed in the given data



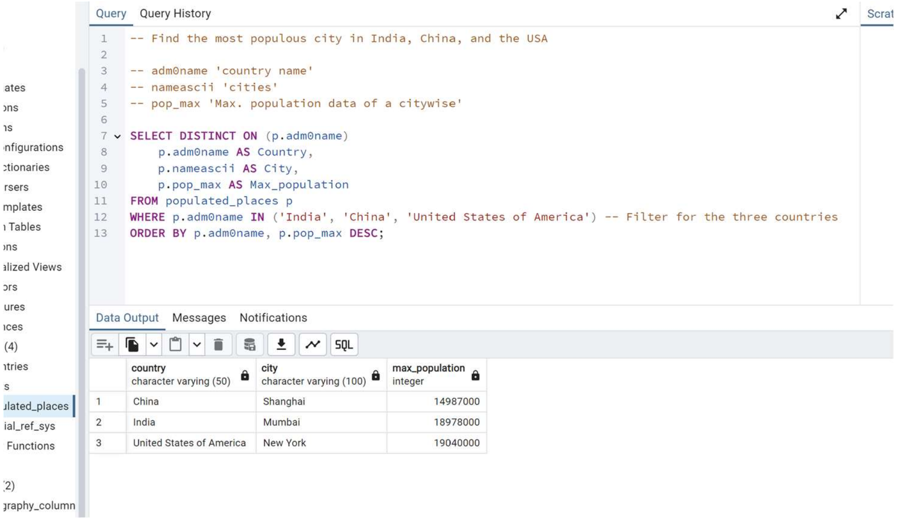
```sql
-- Count the number of populated cities in choiced country

SELECT COUNT(*) AS num_populated_cities
FROM populated_places p
WHERE adm0name = 'Australia'  --country name
```

	num_populated_cities bigint
1	224

□ populated_places Table: This is assumed to be the table containing information about cities and their respective countries.
□ Filter with WHERE Clause: The condition country_name = 'Australia' conclude cities in Australia.

□ Count Cities: COUNT(*) calculates the number of populated cities for each country.

3) Which is the most populous city in India, China and USA

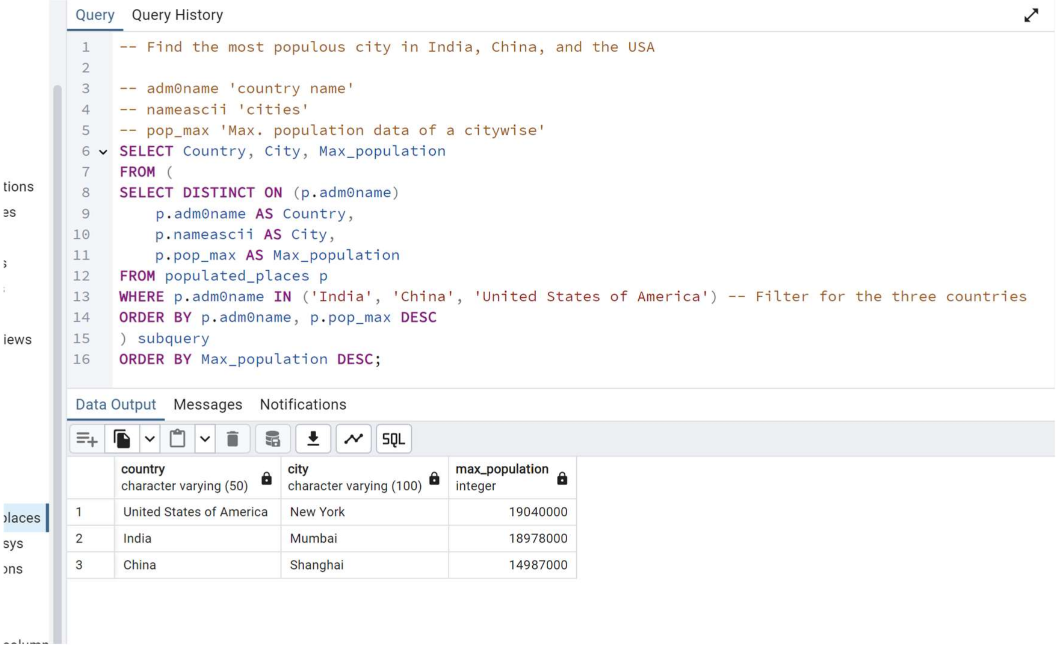


```sql
-- Find the most populous city in India, China, and the USA

-- adm0name 'country name'
-- nameascii 'cities'
-- pop_max 'Max. population data of a citywise'

SELECT DISTINCT ON (p.adm0name)
    p.adm0name AS Country,
    p.nameascii AS City,
    p.pop_max AS Max_population
FROM populated_places p
WHERE p.adm0name IN ('India', 'China', 'United States of America') -- Filter for the three countries
ORDER BY p.adm0name, p.pop_max DESC;
```

	country character varying (50)	city character varying (100)	max_population integer
1	China	Shanghai	14987000
2	India	Mumbai	18978000
3	United States of America	New York	19040000

- populated_places Table: This table is assumed to contain columns such as country_name, city_name, and population.
- DISTINCT ON (country_name): Ensures only one city is selected per country.
- ORDER BY country_name, population DESC: Orders by country first and then by population (descending) to pick the most populous city.

Now we ensure the countries are sorted by their population in descending order



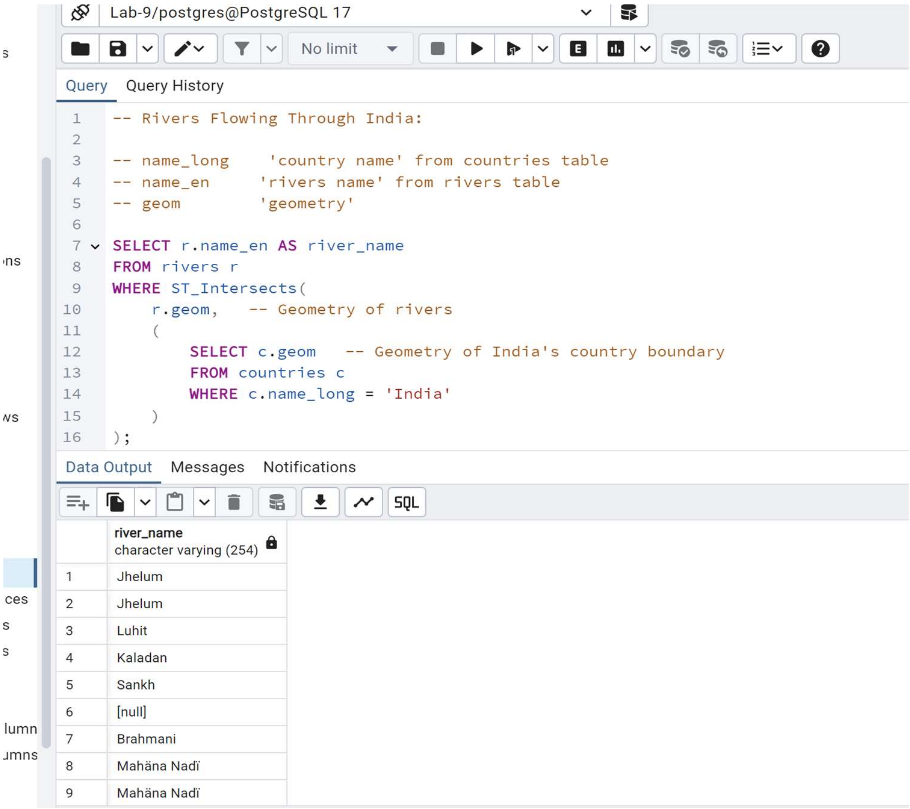
```sql
-- Find the most populous city in India, China, and the USA

-- adm0name 'country name'
-- nameascii 'cities'
-- pop_max 'Max. population data of a citywise'
SELECT Country, City, Max_population
FROM (
SELECT DISTINCT ON (p.adm0name)
    p.adm0name AS Country,
    p.nameascii AS City,
    p.pop_max AS Max_population
FROM populated_places p
WHERE p.adm0name IN ('India', 'China', 'United States of America') -- Filter for the three countries
ORDER BY p.adm0name, p.pop_max DESC
) subquery
ORDER BY Max_population DESC;
```

	country character varying (50)	city character varying (100)	max_population integer
1	United States of America	New York	19040000
2	India	Mumbai	18978000
3	China	Shanghai	14987000

□ Subquery: The inner query (subquery) uses DISTINCT ON to get the most populous city for each country.

- Outer Query: The outer query sorts the results by max_population in descending order (ORDER BY max_population DESC).
- Final Sorting: Ensures the country with the most populous city appears first.

4) Find the rivers which flow through India



Lab-9/postgres@PostgreSQL 17

No limit

Query | Query History

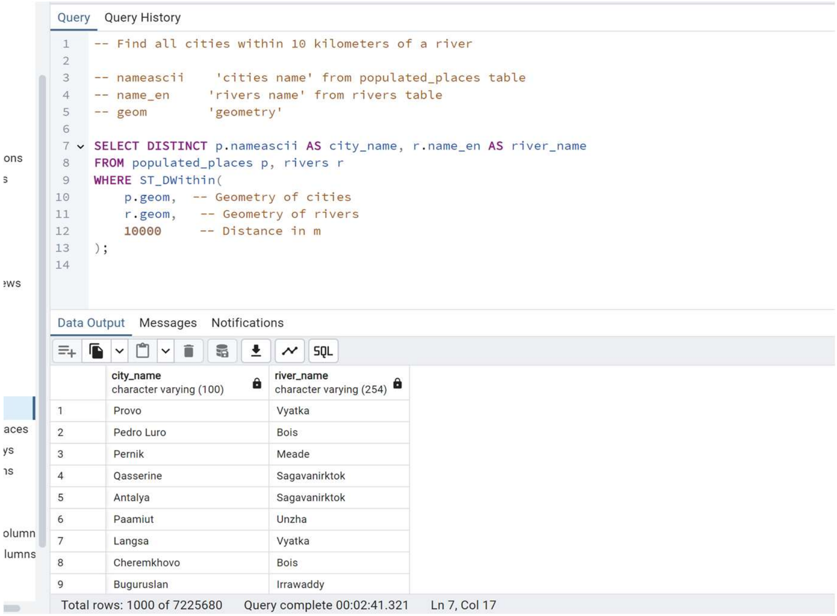
```sql
-- Rivers Flowing Through India:

-- name_long    'country name' from countries table
-- name_en     'rivers name' from rivers table
-- geom        'geometry'

SELECT r.name_en AS river_name
FROM rivers r
WHERE ST_Intersects(
    r.geom,   -- Geometry of rivers
    (
        SELECT c.geom   -- Geometry of India's country boundary
        FROM countries c
        WHERE c.name_long = 'India'
    )
);
```

Data Output | Messages | Notifications

	river_name character varying (254)
1	Jhelum
2	Jhelum
3	Luhit
4	Kaladan
5	Sankh
6	[null]
7	Brahmani
8	Mahäna Nadï
9	Mahäna Nadï

1) Tables and Columns:
rivers Table: Contains information about rivers, including their geometries (geom) and names (name_en).
countries Table: Contains country boundaries and their geometries (geom), along with the country name (name_long).
2) ST_Intersects Function:
Checks if two geometries intersect. In this case, it checks if the river geometry intersects with India's boundary geometry.
3) Subquery:
Retrieves the geometry of India from the countries table (c.geom), filtering by the name_long column where the country name is 'India'.
4) Result:
Returns the names of rivers (r.name_en) whose geometries intersect with India's boundary.

5) Find all cities that are within 10 kms from a river.



```sql
-- Find all cities within 10 kilometers of a river

-- nameascii    'cities name' from populated_places table
-- name_en     'rivers name' from rivers table
-- geom        'geometry'

SELECT DISTINCT p.nameascii AS city_name, r.name_en AS river_name
FROM populated_places p, rivers r
WHERE ST_DWithin(
    p.geom,  -- Geometry of cities
    r.geom,   -- Geometry of rivers
    10000     -- Distance in m
);
```

	city_name character varying (100)	river_name character varying (254)
1	Provo	Vyatka
2	Pedro Luro	Bois
3	Pernik	Meade
4	Qasserine	Sagavanirktok
5	Antalya	Sagavanirktok
6	Paamiut	Unzha
7	Langsa	Vyatka
8	Cheremkhovo	Bois
9	Buguruslan	Irrawaddy

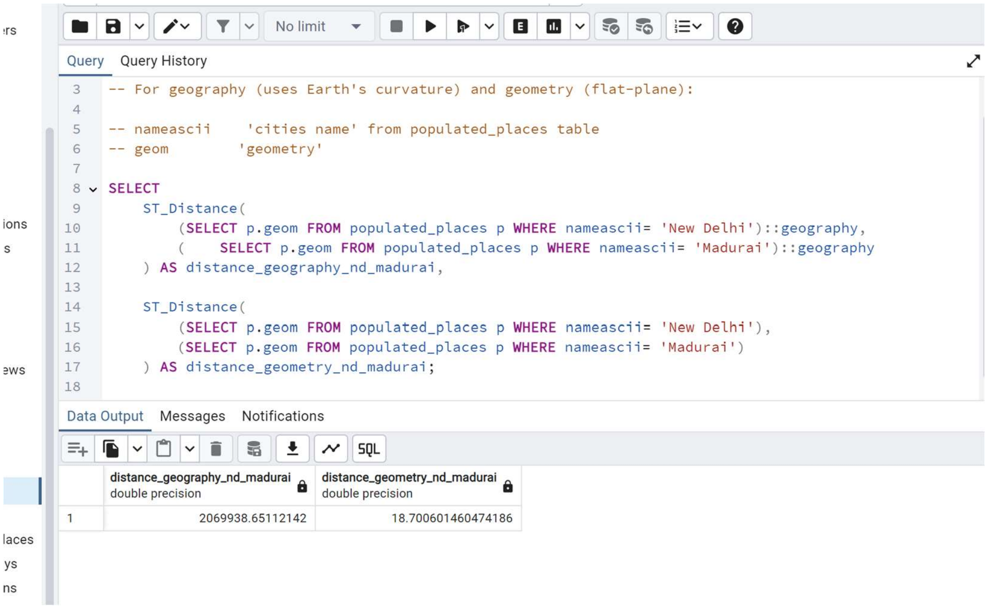
Total rows: 1000 of 7225680   Query complete 00:02:41.321   Ln 7, Col 17

- ST_DWithin Function:
  Checks if the geometry of a city (c.geom) is within 10 kilometers (10,000 meters) of a river's geometry (r.geom).

- DISTINCT:
  Ensures that cities are not listed multiple times if they are within 10 km of multiple rivers.

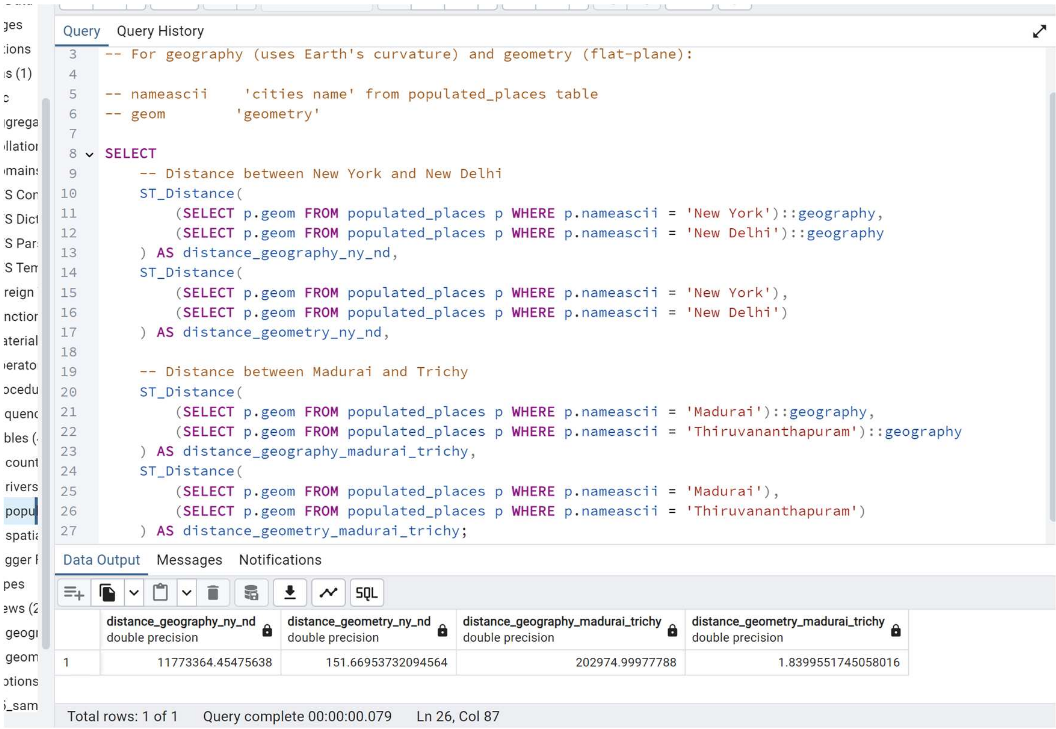
6) Find the distance between a) NewDelhi and Madurai b) NewYork and NewDelhi c) Madurai and Trichy (report in terms of geography and Geometry

a) NewDelhi and Madurai



```sql
-- For geography (uses Earth's curvature) and geometry (flat-plane):

-- nameascii    'cities name' from populated_places table
-- geom        'geometry'

SELECT
    ST_Distance(
        (SELECT p.geom FROM populated_places p WHERE nameascii= 'New Delhi')::geography,
        (    SELECT p.geom FROM populated_places p WHERE nameascii= 'Madurai')::geography
    ) AS distance_geography_nd_madurai,

    ST_Distance(
        (SELECT p.geom FROM populated_places p WHERE nameascii= 'New Delhi'),
        (SELECT p.geom FROM populated_places p WHERE nameascii= 'Madurai')
    ) AS distance_geometry_nd_madurai;
```

	distance_geography_nd_madurai double precision	distance_geometry_nd_madurai double precision
1	2069938.65112142	18.700601460474186



```sql
-- For geography (uses Earth's curvature) and geometry (flat-plane):

-- nameascii    'cities name' from populated_places table
-- geom        'geometry'

SELECT
    -- Distance between New York and New Delhi
    ST_Distance(
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'New York')::geography,
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'New Delhi')::geography
    ) AS distance_geography_ny_nd,
    ST_Distance(
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'New York'),
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'New Delhi')
    ) AS distance_geometry_ny_nd,

    -- Distance between Madurai and Trichy
    ST_Distance(
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'Madurai')::geography,
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'Thiruvananthapuram')::geography
    ) AS distance_geography_madurai_trichy,
    ST_Distance(
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'Madurai'),
        (SELECT p.geom FROM populated_places p WHERE p.nameascii = 'Thiruvananthapuram')
    ) AS distance_geometry_madurai_trichy;
```

	distance_geography_ny_nd double precision	distance_geometry_ny_nd double precision	distance_geography_madurai_trichy double precision	distance_geometry_madurai_trichy double precision
1	11773364.45475638	151.66953732094564	202974.99977788	1.8399551745058016

Total rows: 1 of 1 Query complete 00:00:00.079 Ln 26, Col 87

1) ST_Distance with ::geography:
Calculates the great-circle distance on the Earth's surface, accounting for the curvature of the Earth. It is suitable for long distances.

2) ST_Distance without ::geography:
Computes the planar distance between geometries based on the coordinate system (geometry data type). This is less accurate for long distances.

3) City Selection:
The SELECT geom FROM populated_places WHERE nameascii = 'City Name' retrieves the geometry for the specified city.

4) Pairs of Cities:
The query calculates distances for three pairs of given cities

4. Write your inference

Inferences:

The spatial_ref_sys and geometry_columns tables help understand the spatial reference systems and geometries of spatial data in the database. They provide essential metadata for spatial queries.

Shapefile Import: Using shp2pgsql or the GUI, shapefiles were successfully imported into PostGIS, creating spatial tables (countries, rivers, populated_places) with geometry data for analysis.

SQL Query Results:

a. **Countries**: Queries effectively retrieve and group data based on attributes like names and population.
b. **Populated Cities**: Filters and counts cities accurately by conditions like country and population.
c. **Rivers**: Spatial intersections identify relationships between rivers and countries & cities.
d. **City Proximity**: ST_DWithin is invaluable for proximity queries, demonstrating spatial relationships (cities near rivers).
e. **Distances**: ST_Distance with ::geography provides accurate real-world distances, while geometry supports planar calculations.

Overall Observations:

PostGIS enhances PostgreSQL's ability to process spatial data efficiently. It allows for advanced spatial queries, proximity analyses, and real-world spatial computations, making it a powerful tool for geospatial applications.

Challenges:

Ensuring correct spatial reference systems and accurate data (e.g., city names and geometries) is crucial for meaningful results.