Experiment No.5

Title: Execution of Spatial database queries

Batch:SY-IT(B3) **Roll No.:16010423076** **Experiment No.:5**

Aim: To execute spatial queries using PostGIS.

Resources needed: PostgreSQL 9.6, PostGIS 2.0

_

Theory

PostGIS is an open source software program that adds support for geographic objects to the PostgreSQL object-relational database. PostGIS follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC). PostGIS turns the PostgreSQL Database Management System into a spatial database by adding support for the three features: spatial types, indexes, and functions. Because it is built on PostgreSQL, PostGIS automatically inherits important "enterprise" features as well as open standards for implementation. PostgreSQL is a powerful, object-relational database management system (ORDBMS). It is also open source software.

Features of PostGIS

Geometry types for points, line strings, polygons, multi-points, multi-line-strings, multi-polygons and geometry collections.

Spatial predicates for determining the interactions of geometries using the 3x3 Egenhofer matrix (provided by the GEOS software library).

Spatial operators for determining geospatial measurements like area, distance, length and perimeter.

Spatial operators for determining geospatial set operations, like union, difference, symmetric difference and buffers (provided by GEOS).

R-tree-over-GiST (Generalised Search Tree) spatial indexes for high speed spatial querying.

Index selectivity support, to provide high performance query plans for mixed spatial/non-spatial queries.

For raster data

Geometry is and abstract type and concrete subtypes can be **atomic** or **collection** types

- Atomic
  - Point : It represents a single location in coordinate space

e.g. POINT(3, 4), POINT (3,5,4,8)

o LineString : It is a 1-dimensional line formed by a contiguous sequence of line segments. Each line segment is defined by two points, with the end point of one segment forming the start point of the next segment
e.g. LINESTRING (1 2, 3 4, 5 6)

o LineRing : It is a LineString which is both closed and simple. The first and last points must be equal, and the line must not self-intersect
e.g. LINEARRING (0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0)

o Polygon : It is a 2-dimensional planar region, delimited by an exterior boundary (the shell) and zero or more interior boundaries (holes). Each boundary is a LinearRing.
e.g. POLYGON ((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))

▪ **Collection**

o MultiPoint : It is a collection of points
e.g. MULTIPOINT ((0 0), (1 2))

o MultiLineString : It is a collection of LineStrings. A MultiLineString is closed if each of its elements is closed
e.g. MULTILINESTRING ((0 0,1 1,1 2), (2 3,3 2,5 4))

o MultiPolygon : It is a collection of non-overlapping, non-adjacent polygons. Polygons in the collection may touch only at a finite number of points.
e.g. MULTIPOLYGON (((1 5, 5 5, 5 1, 1 1, 1 5)), ((6 5, 9 1, 6 1, 6 5)))

o GeometryCollection : It is a is a heterogeneous (mixed) collection of geometries
e.g. GEOMETRYCOLLECTION (POINT(2 3), LINESTRING(2 3, 3 4))

o Also there are PolyHedralSurface, Triangle and TIN

PostGIS provides different functions for determining relationships(topological or distance) between geometries, compute measurements, overlays and geometry construction also besides other provisions.
Few of the functions are

Measurement functions
ST_Area : `float ST_Area(geometry g1);`
Returns the area of a polygonal geometry

ST_Length : `float ST_Length(geometry a_2dlinestring);` R
Returns the 2D Cartesian length of the geometry if it is a LineString, MultiLineString, ST_Curve, ST_MultiCurve

ST_Perimeter : `float ST_Perimeter(geometry g1);`
Returns the 2D perimeter of the geometry/geography if it is a ST_Surface, ST_MultiSurface (Polygon, MultiPolygon)

Named Spatial Relationships

For determining common spatial relationships, OGC SFS defines a set of named spatial relationship predicates. PostGIS provides these as the functions

ST_Contains : `boolean ST_Contains(geometry geomA, geometry geomB);`
ST_Crosses : `boolean ST_Crosses(geometry g1, geometry g2);`
ST_Disjoint : `boolean ST_Disjoint( geometry A , geometry B );`
ST_Equals : `boolean ST_Equals(geometry A, geometry B);`
ST_Intersects : `boolean ST_Intersects( geometry geomA , geometry geomB );`
ST_Overlaps : `boolean ST_Overlaps(geometry A, geometry B);`
ST_Touches : `boolean ST_Touches(geometry A, geometry B);`
ST_Within. : `boolean ST_Within(geometry A, geometry B);`

It also defines the non-standard relationship predicates

ST_Covers : `boolean ST_Covers(geometry geomA, geometry geomB);`
ST_CoveredBy : `boolean ST_CoveredBy(geometry geomA, geometry geomB);`
ST_ContainsProperly : `boolean ST_ContainsProperly(geometry geomA, geometry geomB);`

Spatial predicates are usually used as conditions in SQL WHERE or JOIN clauses.

```
SELECT city.name, state.name, city.geom
FROM city JOIN state ON ST_Intersects(city.geom, state.geom);
```

Visualization of shape file on GOOGLE Earth Engine(GEE):

Google Earth Engine (GEE) is a cloud-based platform developed by Google for planetary-scale environmental data analysis. It provides a powerful infrastructure for analyzing and visualizing geospatial data, making it particularly valuable for tasks such as remote sensing, environmental monitoring, and land cover analysis.
It facilitates the Visualization and Mapping of geospatial data. GEE provides tools for visualizing geospatial data, including the ability to create interactive maps and time-lapse animations. Users can visualize the results of their analyses directly within the Code Editor or export visualizations for external use.

Procedure:

1. Installation of relational database PostgreSQL 9.6 (download from http://www.enterprisedb.com/products-services-training/pgdownload)
2. Installation of PostGIS using Application stack builder.
3. Download spatial data from `https://www.diva-gis.org/gdata` (OR similar website with FREE usable data) Get it for any country with minimum 3 subjects.
4. Import the data in your PostgreSQL
   access video resource

Spatial Database demo-20220223_104701.mpg from https://drive.google.com/drive/folders/1jB7t4zVtyANA70XfHiwF2qU_JSMlU1_n?usp=drive_link

5. Identify spatial relationship between any two geometric entities (any 3 named relationships)
6. Perform any two measurement functions for geometric data.

7. Execute any one range query
8. create account on GEE using somaiya email id
9. upload the shapefile
10. visualize it on GEE

access the video resources **GE_shapefile-SMP.mp4** and **shptocsv,mp4** from https://drive.google.com/drive/folders/1jB7t4zVtyANA70XfHiwF2qU_JSMlU1_n?usp=drive_link

var shapefile : Table `projects/ee-suchitrapatil23/assets/indian_waterbodies_area// this you need to do for your shapefile in assets`

```
// Load the shapefile as a FeatureCollection.
var shapefile = shapefileUrl;

// Display the shapefile on the map.
Map.centerObject(shapefile, 10); // Center the map on the shapefile
Map.addLayer(shapefile, {}, 'Shapefile');

// Print the FeatureCollection to the console to inspect its properties.
print(shapefile);

// Convert the shapefile to a CSV table.
var csvTable = shapefile;
// Print the resulting CSV table.
print(csvTable);

// Export the CSV table to Google Drive.
Export.table.toDrive({
  collection: csvTable,
  description: 'indian_waterbodies_shapefilecsv',
  fileFormat: 'CSV'
});
```

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;

        d         | edabbr | vaabbr
------------------+--------+--------
                0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

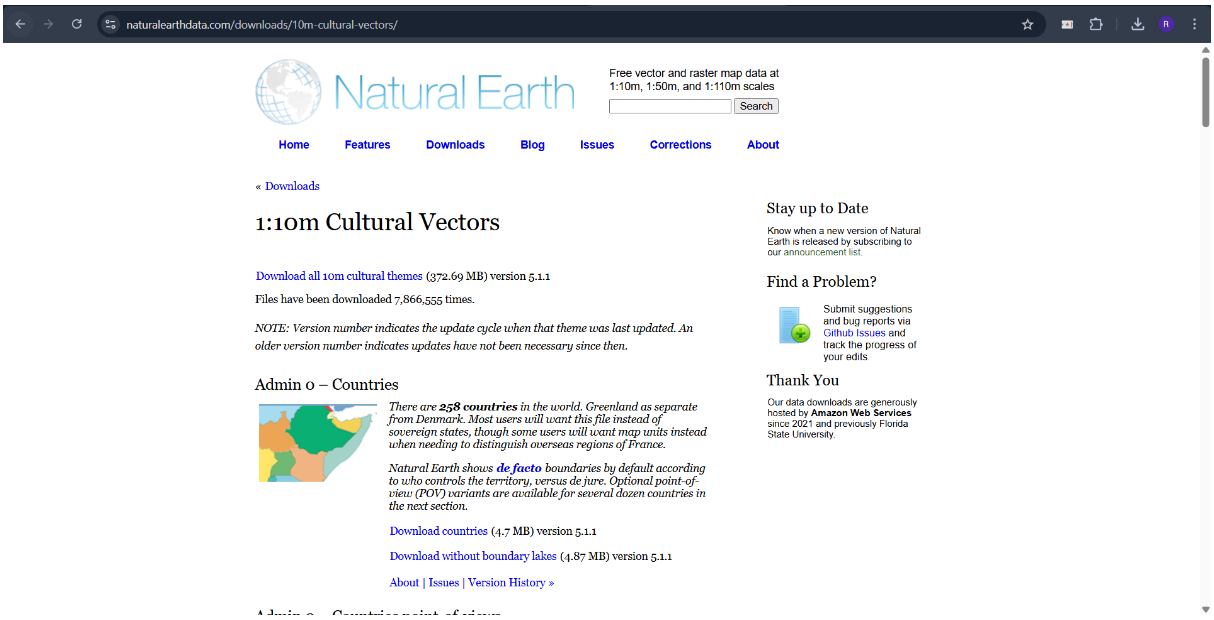
Range query in Postgis
SELECT ST_Reclass(rast, 1,

```
'[0-90]:0,(90-100):1,[100-1000):2',
   '4BUI', 0) AS rast  FROM  sometable
WHERE filename = '123.tif';
```

Results: (Program printout with output)

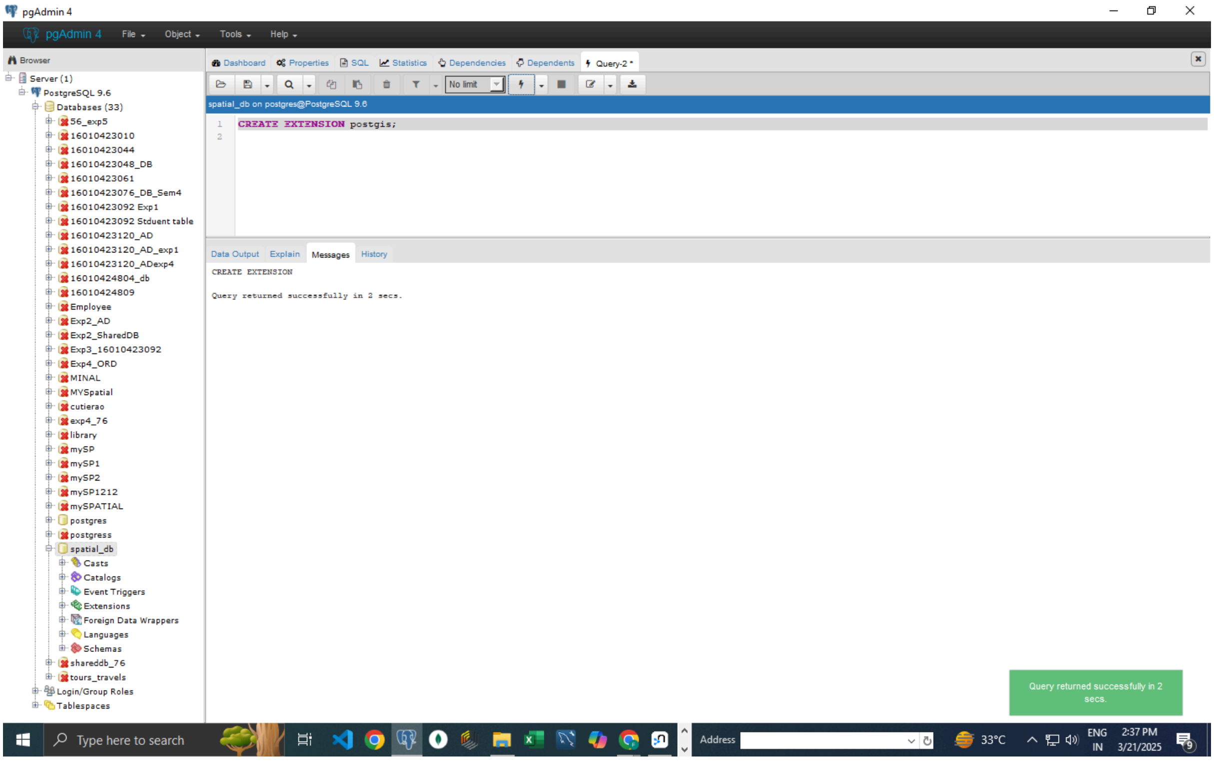
Step 1: Downloading Spatial Data

- To begin, spatial data is required for analysis.
- The dataset can be obtained from [Natural Earth]:
- Select the desired country and download the spatial data.
- Extract the downloaded ZIP file to access the .shp (shapefile) and its related files.

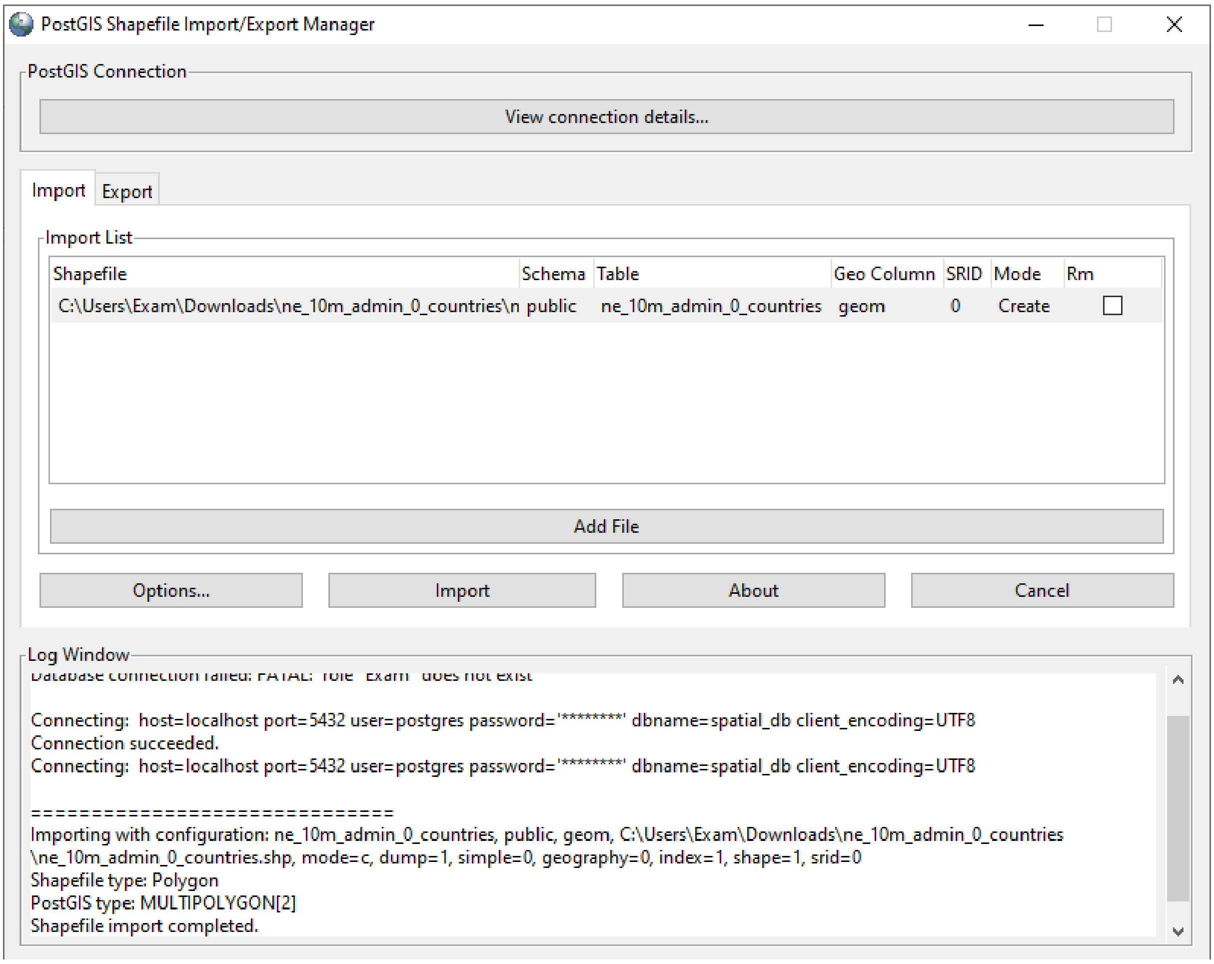


Step 2: Importing Spatial Data into PostgreSQL

- Open pgAdmin and create a new database:
    - Right-click on Databases → Create → Database.
    - Name the database as `spatial_db`.
- Enable the PostGIS extension by executing the following SQL query in pgAdmin:
  `CREATE EXTENSION postgis;`



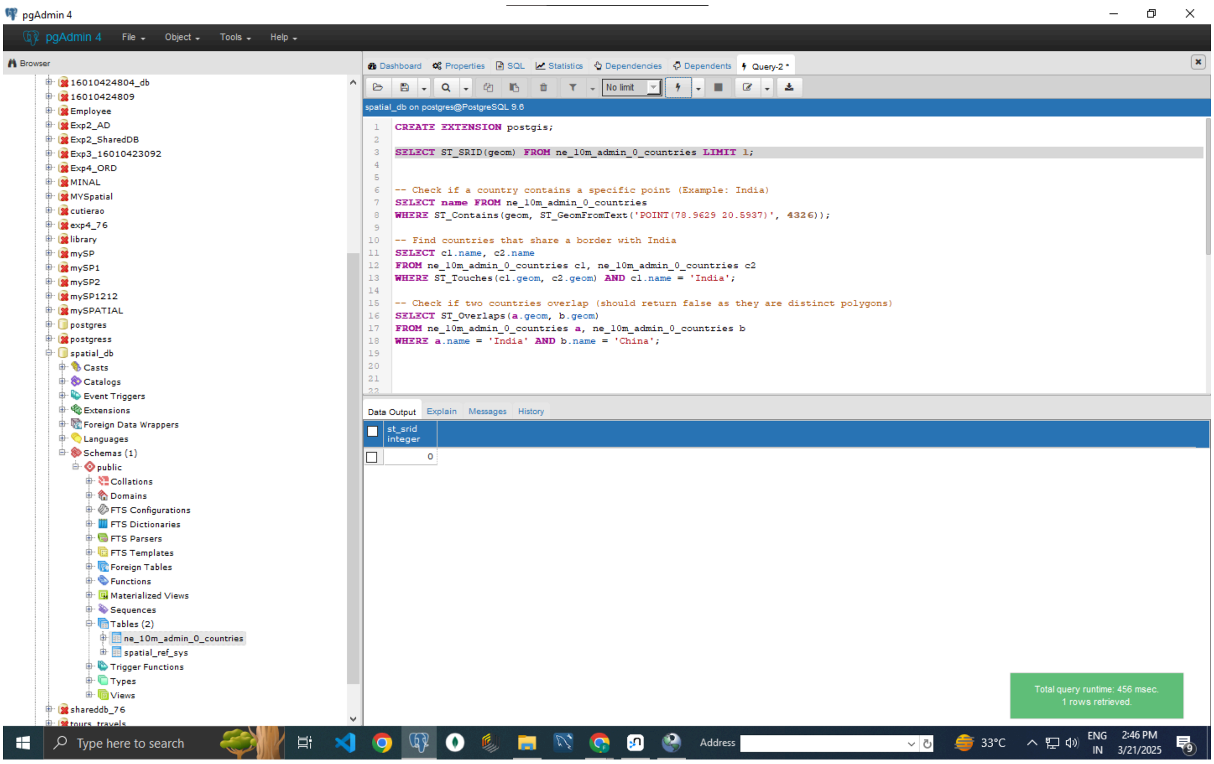
- Use the PostGIS Shapefile and DBF Loader to import .shp files:
  - Navigate to Start Menu → PostGIS Bundle → Shapefile and DBF Loader.
  - Click Add and select the extracted .shp files.
  - Choose the `spatial_db` database and click Import.



Step 3: Running Spatial Queries

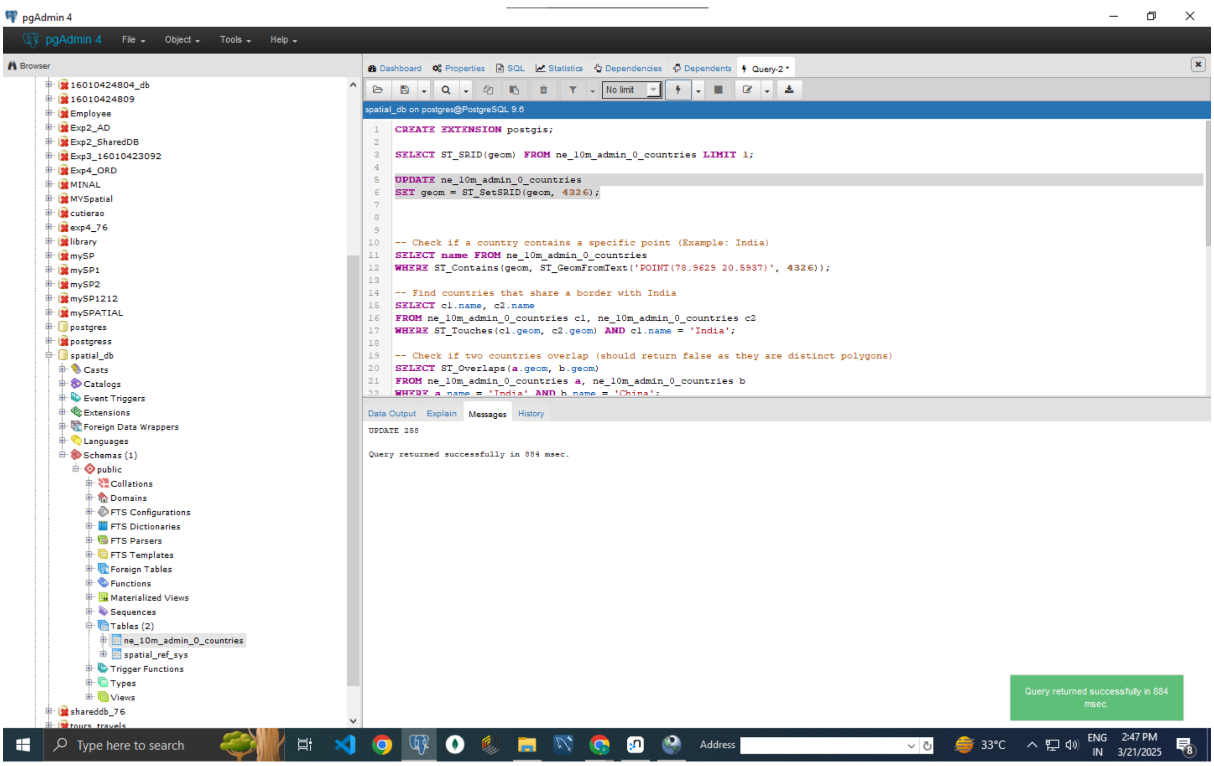
Confirm the Spatial Reference System Identifier (SRID):

```
SELECT ST_SRID(geom) FROM ne_10m_admin_0_countries LIMIT 1;
```



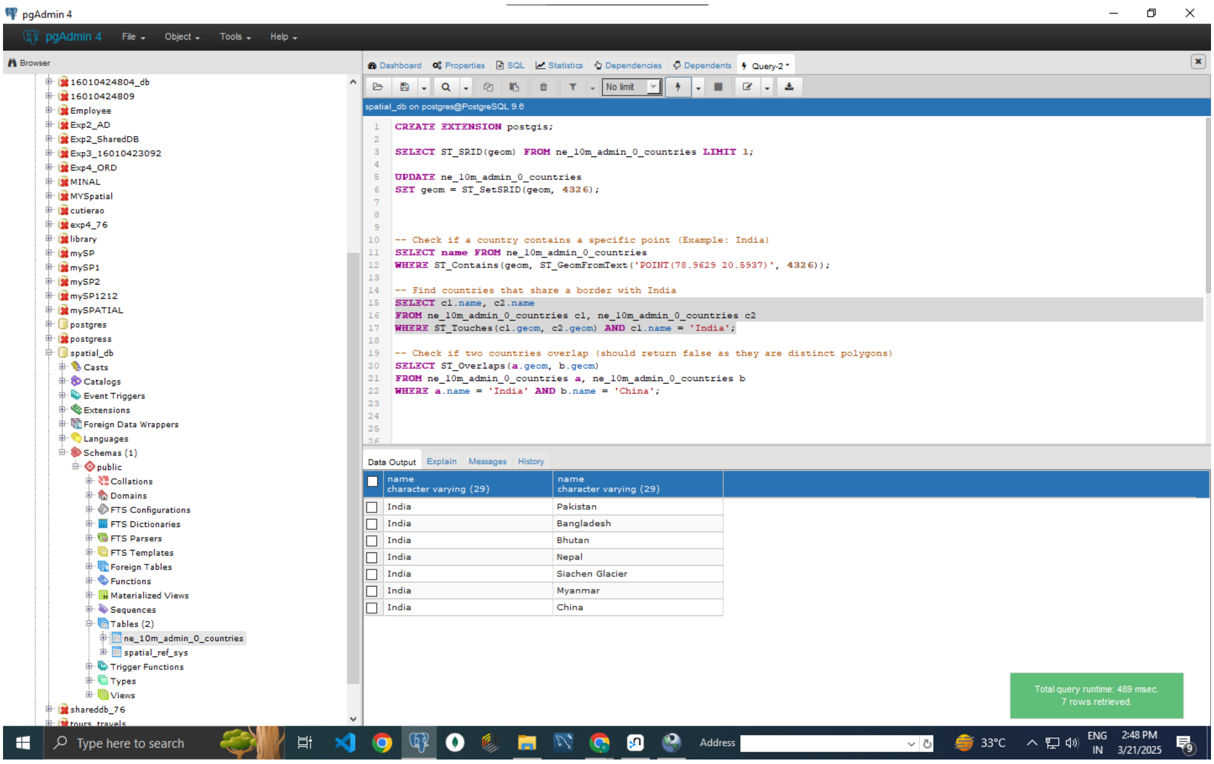
Standardize the SRID to EPSG:4326:

```
UPDATE ne_10m_admin_0_countries
SET geom = ST_SetSRID(geom, 4326);
```



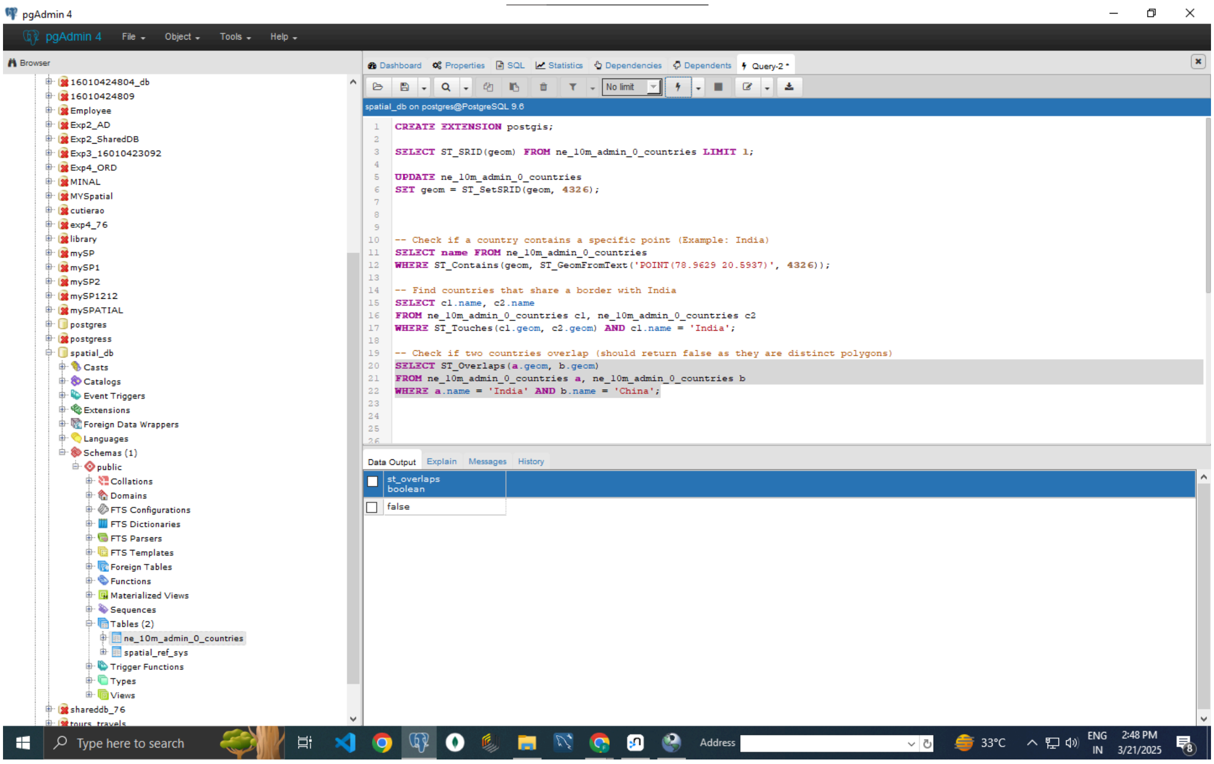
Find neighboring countries of India:

```
SELECT c1.name, c2.name
FROM ne_10m_admin_0_countries c1, ne_10m_admin_0_countries c2
WHERE ST_Touches(c1.geom, c2.geom) AND c1.name = 'India';
```



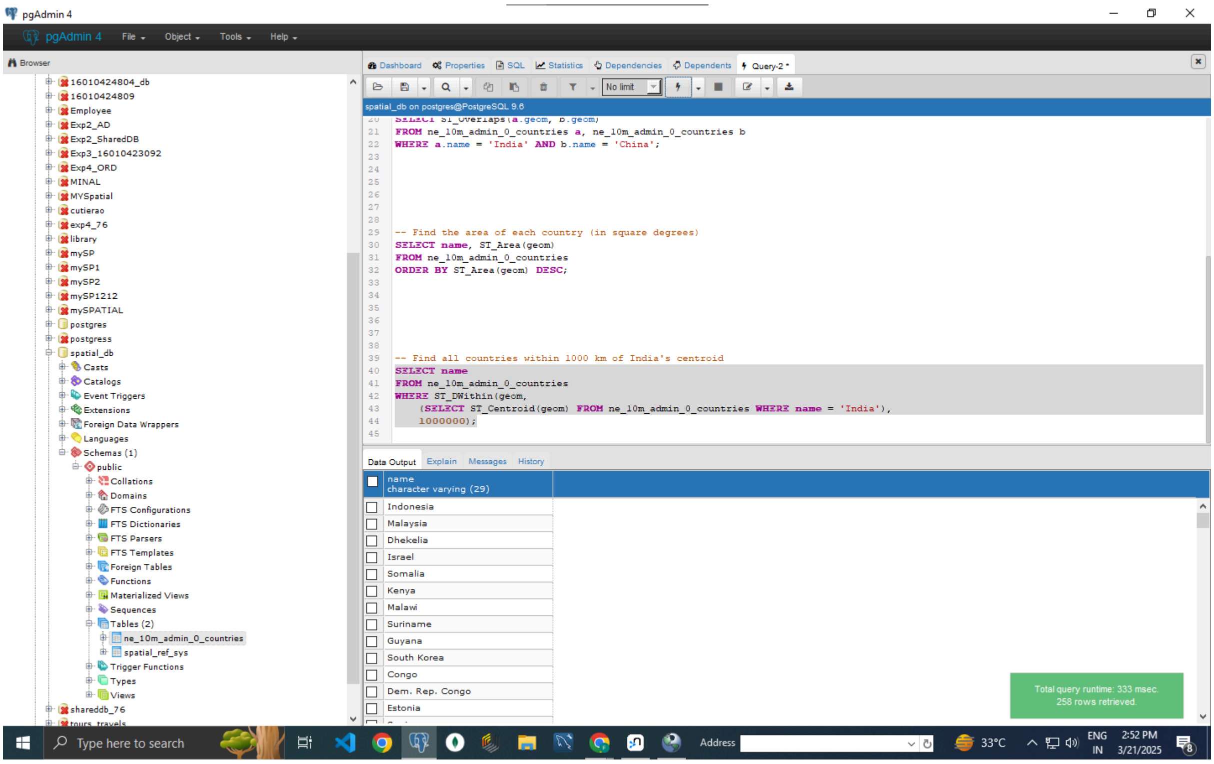
Check if two countries overlap (should return false as they are distinct polygons):

```
SELECT ST_Overlaps(a.geom, b.geom)
FROM ne_10m_admin_0_countries a, ne_10m_admin_0_countries b
WHERE a.name = 'India' AND b.name = 'China';
```



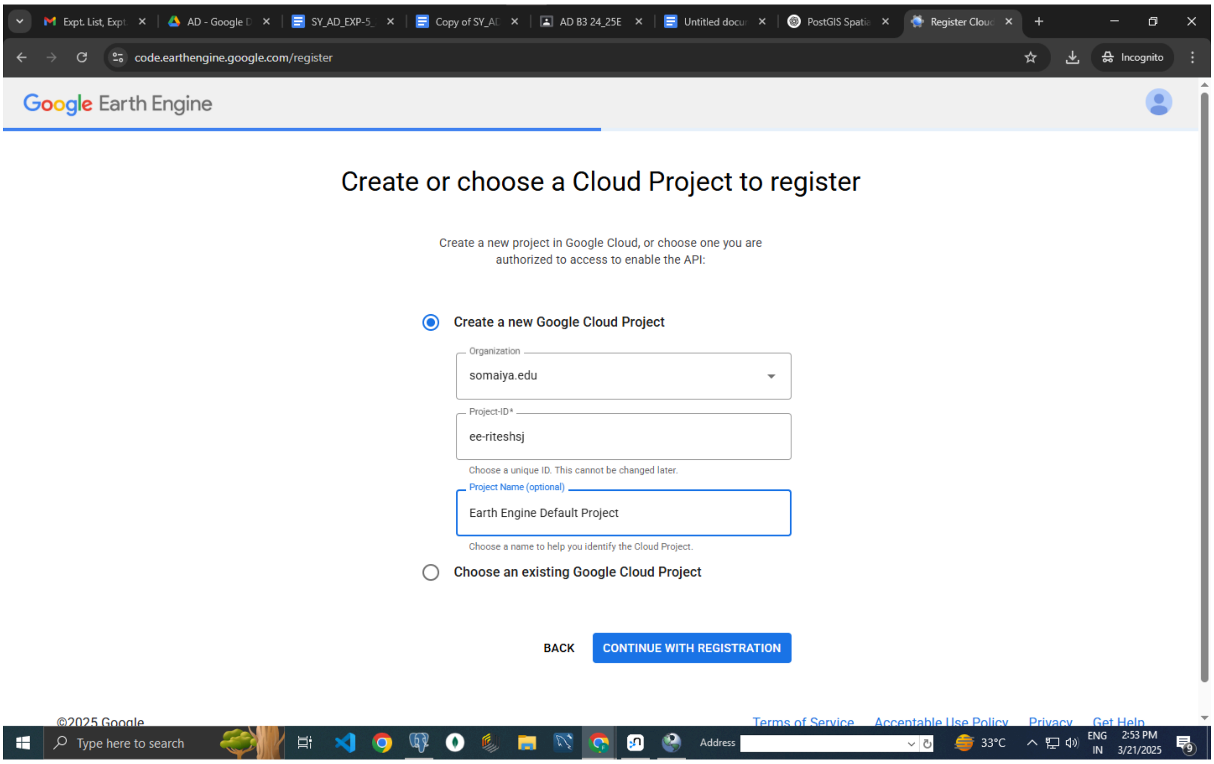
Find all countries within 1000 km of India's centroid:

```
SELECT name
FROM ne_10m_admin_0_countries
WHERE ST_DWithin(geom,
    (SELECT ST_Centroid(geom) FROM ne_10m_admin_0_countries
WHERE name = 'India'),
    1000000);
```

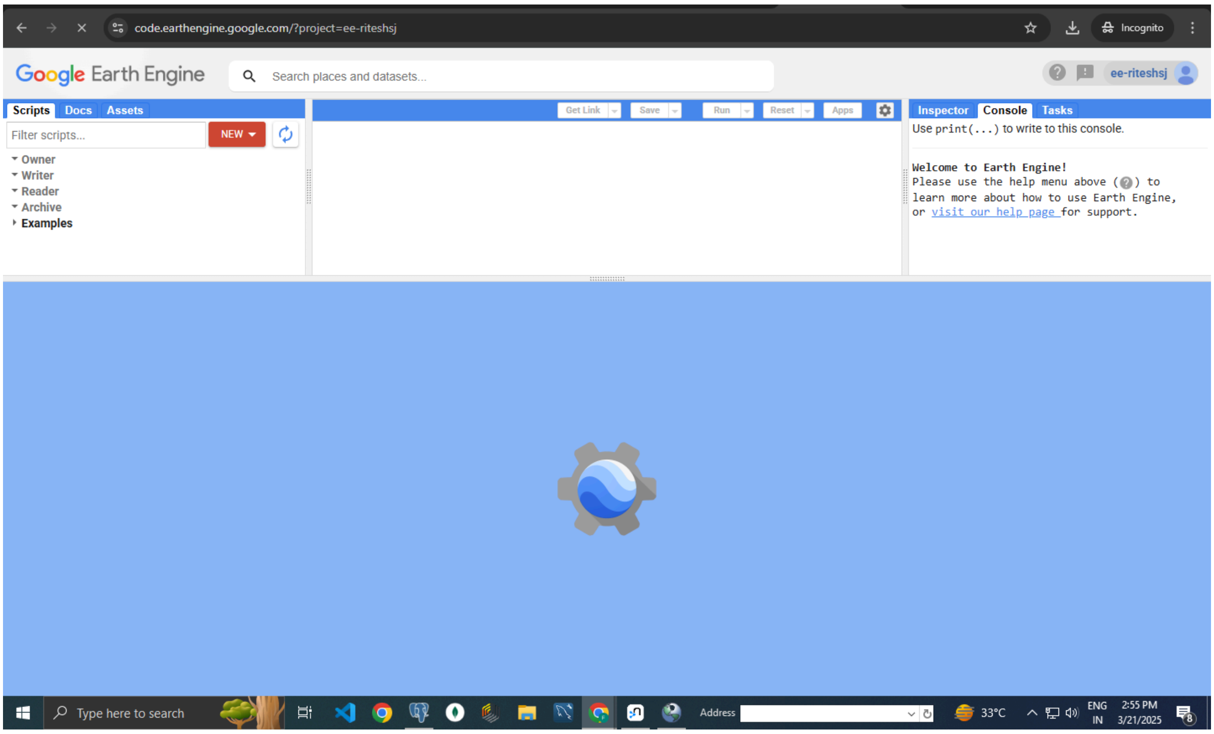


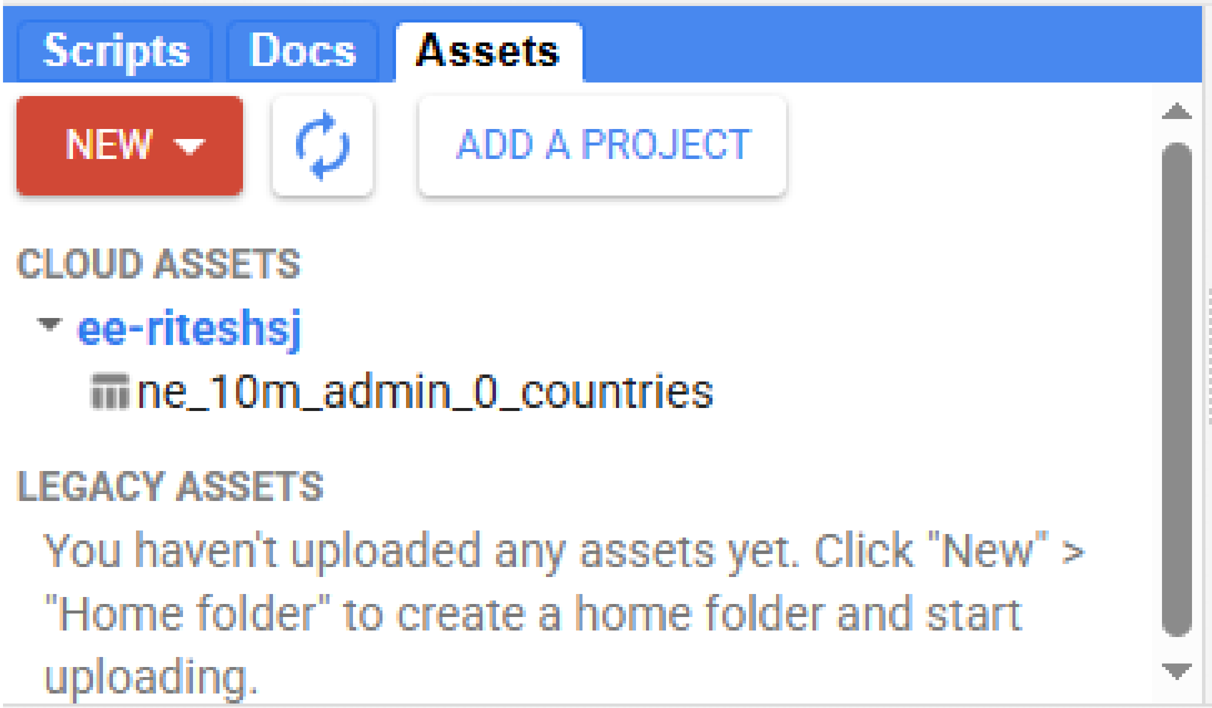
Step 4: Uploading a Shapefile to Google Earth Engine (GEE)

- Register for a Google Earth Engine account using a Somaiya email ID.



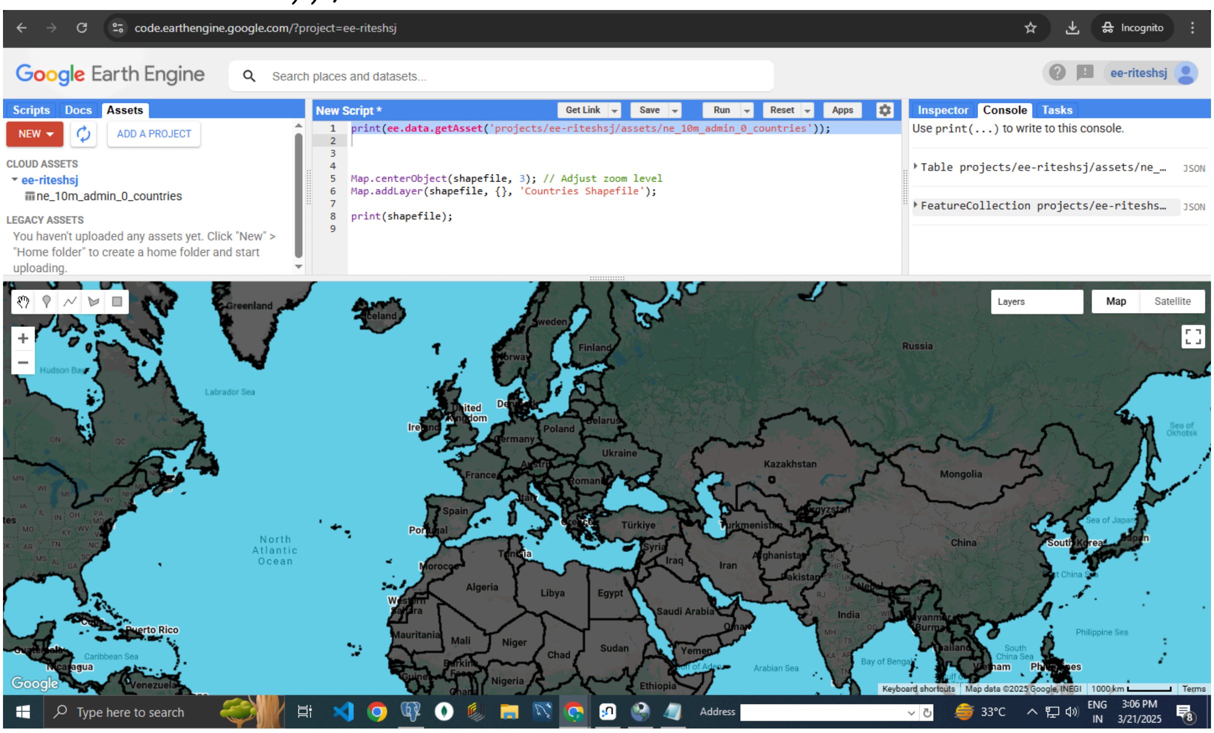
- Access the Google Earth Engine Code Editor.
- Upload the shapefile to the Assets section.



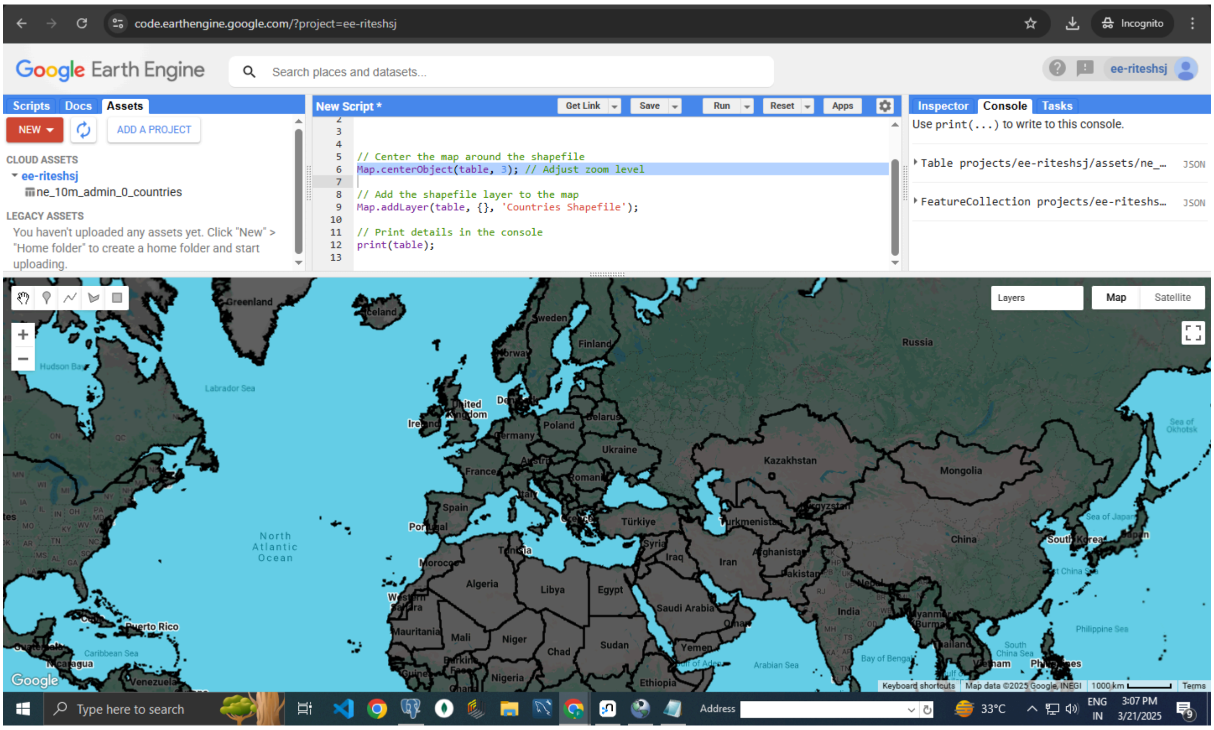


In the code editor :

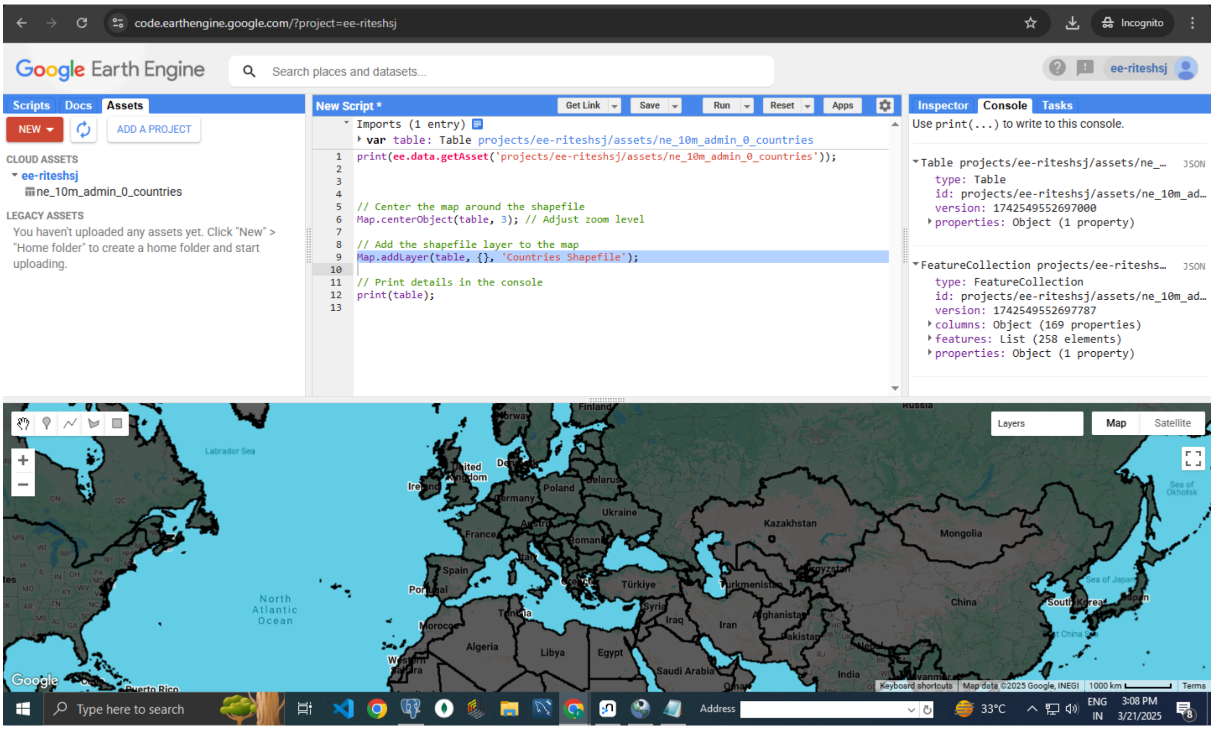
```
print(ee.data.getAsset('projects/ee-riteshsj/assets/ne_10m_admin_0_countries'));
```



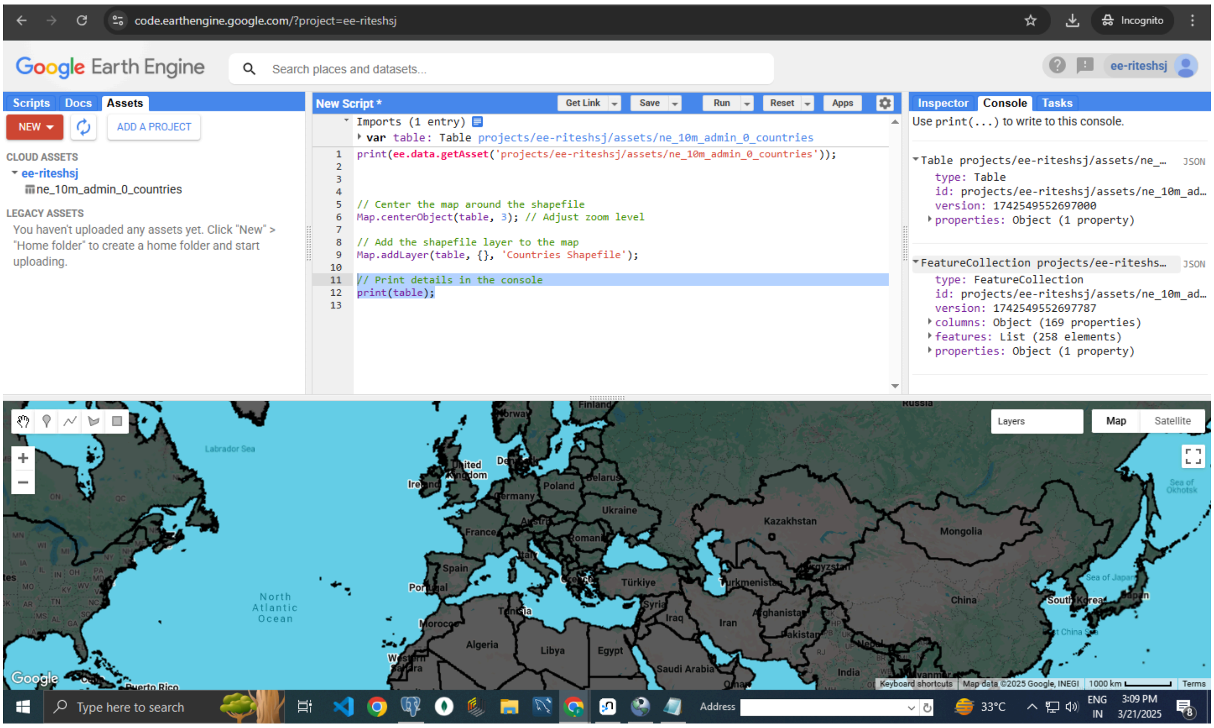
```
// Center the map around the shapefile
Map.centerObject(table, 3); // Adjust zoom level
```



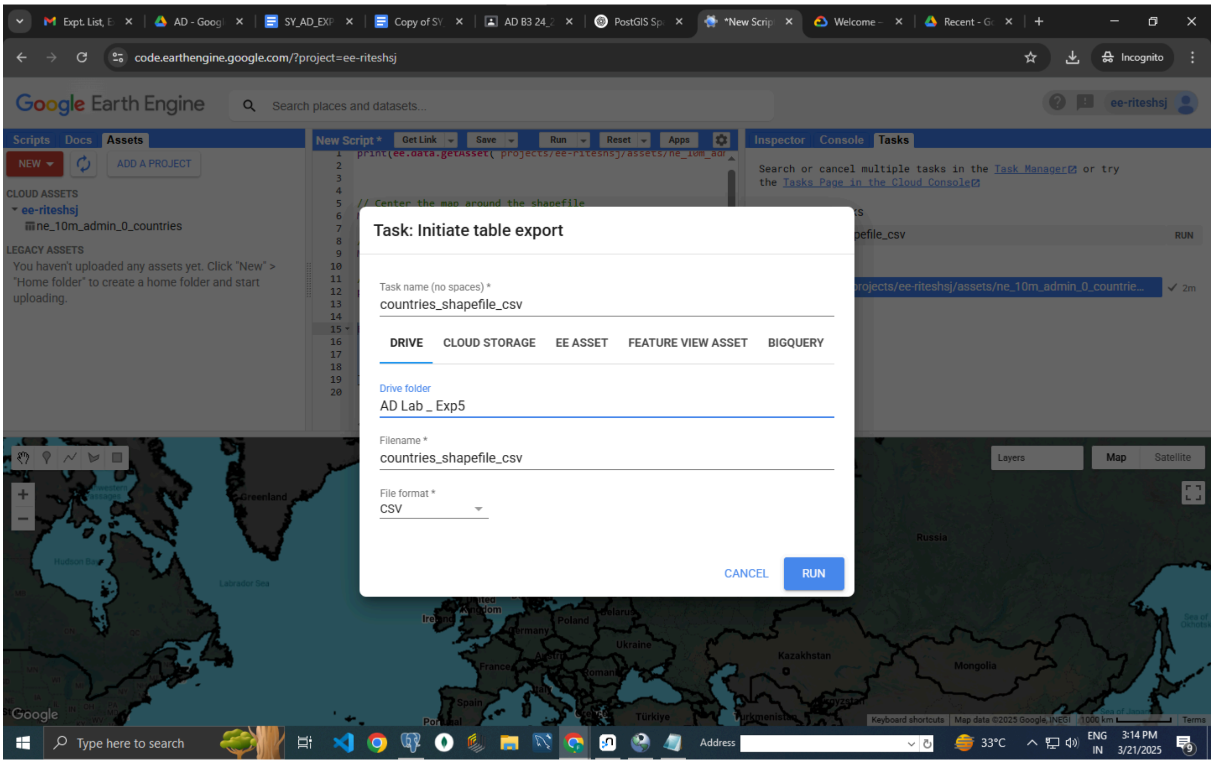

```
// Add the shapefile layer to the map
Map.addLayer(table, {}, 'Countries Shapefile');
```



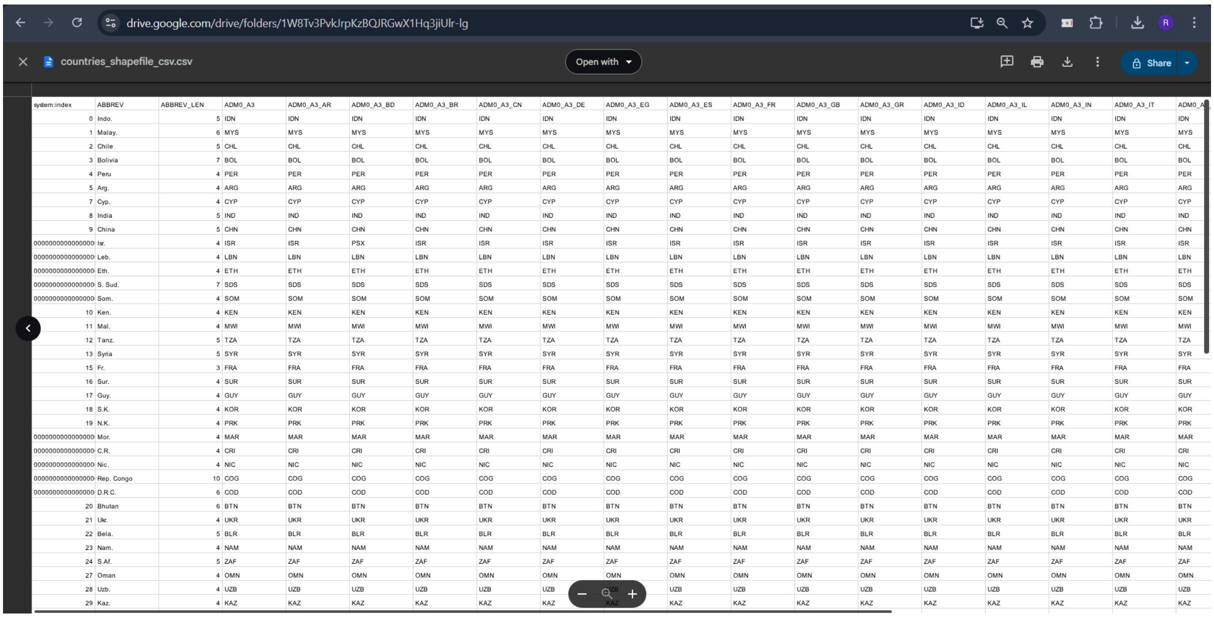

```
// Print details in the console
print(table);
```




```
Export.table.toDrive({
  collection: table,  // Use 'table' instead of 'shapefile'
  description: 'countries_shapefile_csv',
  fileFormat: 'CSV'
});
```



This script enables visualization and export of the shapefile data in Google Earth Engine into a CSV file to our drive.



system:index	ABBREV	ABBREV_LEN	ADM0_A3	ADM0_A3_AR	ADM0_A3_BD	ADM0_A3_BR	ADM0_A3_CN	ADM0_A3_DE	ADM0_A3_EG	ADM0_A3_ES	ADM0_A3_FR	ADM0_A3_GB	ADM0_A3_GR	ADM0_A3_ID	ADM0_A3_IL	ADM0_A3_IN	ADM0_A3_IT	ADM0_A
0	Indo.	5	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN	IDN
1	Malay.	6	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS	MYS
2	Chile	5	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL	CHL
3	Bolivia	7	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL	BOL
4	Peru	4	PER	PER	PER	PER	PER	PER	PER	PER	PER	PER	PER	PER	PER	PER	PER	PER
5	Arg.	4	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG	ARG
7	Cyp.	4	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP	CYP
8	India	5	IND	IND	IND	IND	IND	IND	IND	IND	IND	IND	IND	IND	IND	IND	IND	IND
9	China	5	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN	CHN
0000000000000000	Isr.	4	ISR	ISR	PSX	ISR	ISR	ISR	ISR	ISR	ISR	ISR	ISR	ISR	ISR	ISR	ISR	ISR
0000000000000000	Leb.	4	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN	LBN
0000000000000000	Eth.	4	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH	ETH
0000000000000000	S. Sud.	7	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS	SDS
0000000000000000	Som.	4	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM	SOM
10	Ken.	4	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN	KEN
11	Mal.	4	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI	MWI
12	Tanz.	5	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA	TZA
13	Syria	5	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR	SYR
15	Fr.	3	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA	FRA
16	Sur.	4	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR	SUR
17	Guy.	4	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY	GUY
18	S.K.	4	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR	KOR
19	N.K.	4	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK	PRK
0000000000000000	Mor.	4	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR	MAR
0000000000000000	C.R.	4	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI	CRI
0000000000000000	Nic.	4	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC	NIC
0000000000000000	Rep. Congo	10	COG	COG	COG	COG	COG	COG	COG	COG	COG	COG	COG	COG	COG	COG	COG	COG
0000000000000000	D.R.C.	6	COD	COD	COD	COD	COD	COD	COD	COD	COD	COD	COD	COD	COD	COD	COD	COD
20	Bhutan	6	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN	BTN
21	Ukr.	4	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR	UKR
22	Bela.	5	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR	BLR
23	Nam.	4	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM	NAM
24	S.Af.	5	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF	ZAF
27	Oman	4	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN	OMN
28	Uzb.	4	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB	UZB
29	Kaz.	4	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ	KAZ

Questions:

1. Explain the spatial functions used for these queries in detail.

- Spatial queries are queries executed on spatial databases to retrieve, manipulate, or analyze geographic data based on spatial relationships.
- Unlike standard SQL queries that deal with alphanumeric data, spatial queries involve geometry-based operations such as distance, containment, intersection, and adjacency.
- These queries are essential in Geographic Information System (GIS) applications because they allow users to analyze spatial relationships, visualize geographic patterns, and make data-driven decisions.
- For example, spatial queries can be used to determine the nearest hospital to a location, identify areas within a specific radius of a point, or analyze land use patterns.
- Their importance lies in their ability to integrate spatial data with traditional databases, enabling advanced geospatial analysis for urban planning, environmental monitoring, disaster management, and many other applications.

2. **Explain any two applications of spatial databases.**
   - PostGIS extends PostgreSQL by adding support for spatial data types, spatial indexing, and a wide range of spatial functions.
   - It enables the storage, retrieval, and analysis of geographic data within a PostgreSQL database.
   - PostGIS introduces geometry and geography data types, allowing users to store points, lines, polygons, and multi-geometries.
   - It also provides spatial indexing mechanisms, such as R-tree-over-GiST, which improve the performance of spatial queries by optimizing search operations.
   - Additionally, PostGIS includes various spatial functions, such as ST_Intersects for checking spatial relationships, ST_Distance for measuring distances, and ST_Union for merging geometries.
   - These features make PostGIS a powerful tool for handling geospatial data, making PostgreSQL a fully functional spatial database.

Outcomes: CO2: Design advanced database systems using In-memory, Spatial and NOSQL databases and its implementation.

Conclusion: (Conclusion to be based on outcomes achieved)

From this experiment, I learned how PostGIS enhances PostgreSQL to handle spatial data efficiently. I explored various spatial queries, including spatial relationships and measurement functions, and understood how geospatial data is stored, queried, and visualized. Implementing these queries helped me understand their real-world applications, such as GIS mapping and spatial analysis. Additionally, using Google Earth Engine for visualization provided a practical perspective on handling shapefiles and analyzing geospatial datasets.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

1. Elmasri and Navathe, “Fundamentals of Database Systems”, Pearson Education

2. Raghu Ramakrishnan and Johannes Gehrke, “Database Management Systems” 3rd Edition, McGraw Hill,2002

3. Korth, Silberchatz, Sudarshan, “Database System Concepts” McGraw Hill

4. http://www.bostongis.com/PrinterFriendly.aspx?content_name=postgis_tut01