

## Group Project

### Team Members:

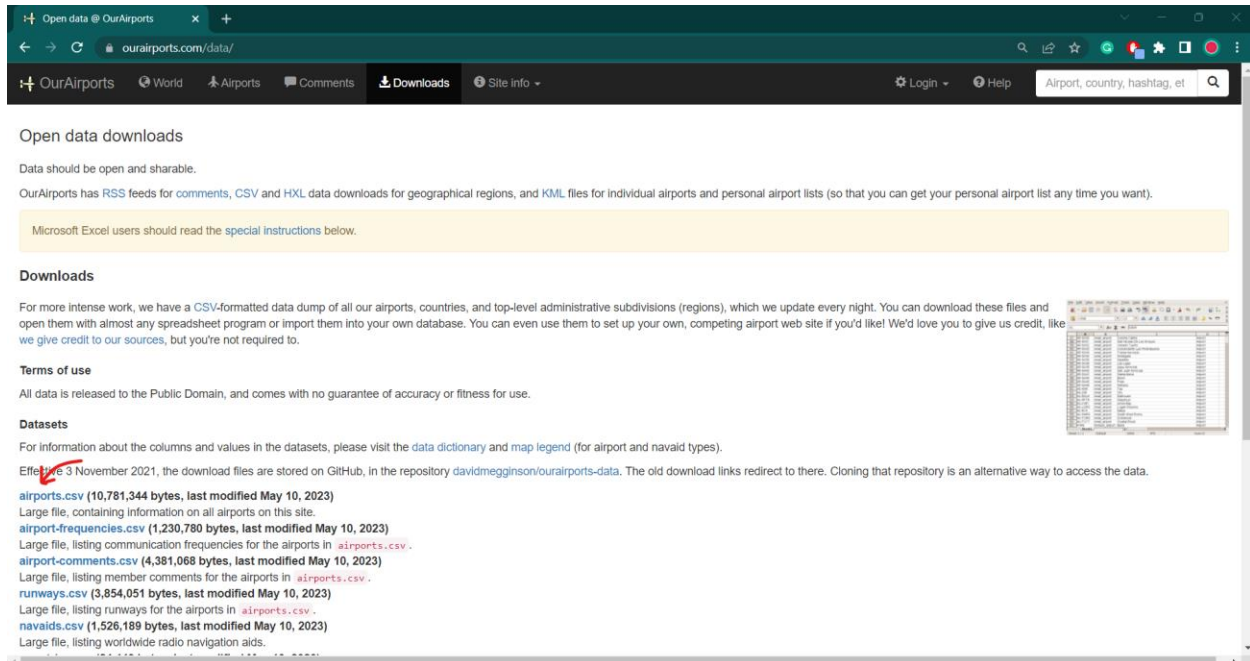
Pruthvi Raj Pudi

Vamshi Krishna Bangaru

Adusumalli Lokesh

### Our Dataset Source:

We're using data from OurAirports (<https://ourairports.com/data/>), a platform where users share and download geospatial data related to airports worldwide. The file, airports.csv, contains comprehensive geospatial data for all the airports listed on the platform, updated as of May 9, 2023. It's in a CSV format, which is easily imported into various software for further spatial analysis.



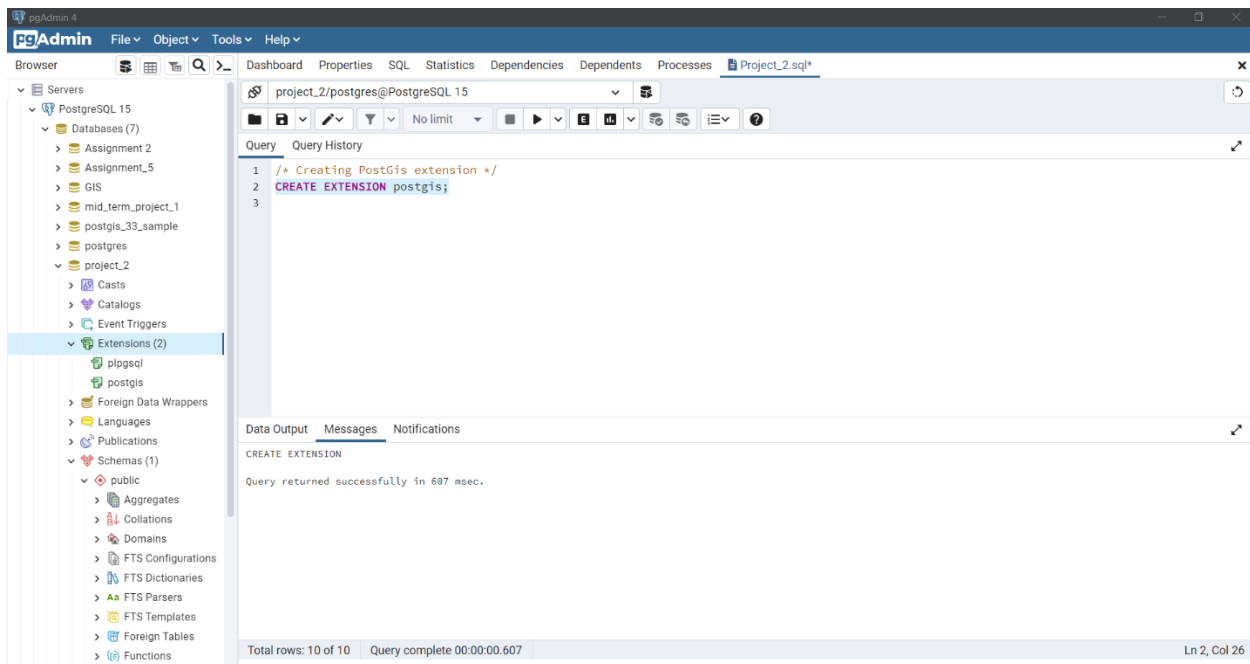
# CS 623 - Database Management Systems

## Group Project

### Creating the PostGIS Extension:

**CREATE EXTENSION postgis;**

This command is used to add the PostGIS extension to your PostgreSQL database. If it's not already installed, this command will install it.

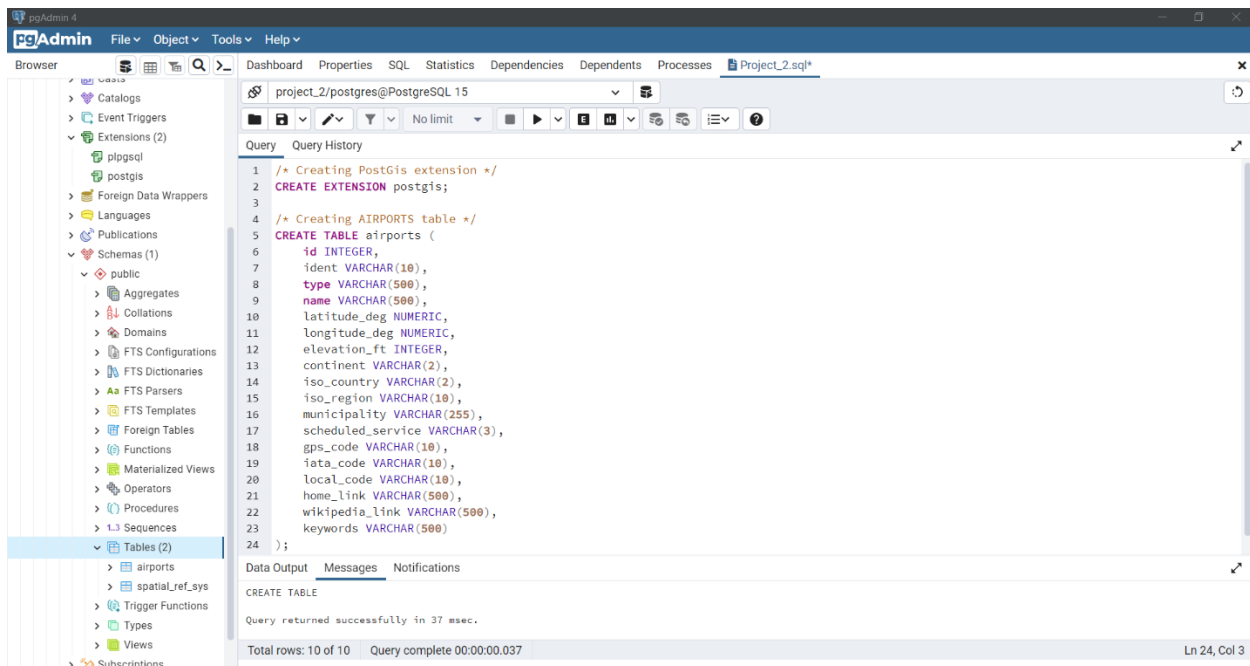


PostGIS is an extension of PostgreSQL and is the common standard for vector geodata in open source databases. It adds support for geographic objects, allowing the database to perform spatial queries and analyses.

## Group Project

### Creating the Airports Table:

```
CREATE TABLE airports (  
    id INTEGER,  
    ident VARCHAR(10),  
    type VARCHAR(500),  
    name VARCHAR(500),  
    latitude_deg NUMERIC,  
    longitude_deg NUMERIC,  
    elevation_ft INTEGER,  
    continent VARCHAR(2),  
    iso_country VARCHAR(2),  
    iso_region VARCHAR(10),  
    municipality VARCHAR(255),  
    scheduled_service VARCHAR(3),  
    gps_code VARCHAR(10),  
    iata_code VARCHAR(10),  
    local_code VARCHAR(10),  
    home_link VARCHAR(500),  
    wikipedia_link VARCHAR(500),  
    keywords VARCHAR(500));
```



The CREATE TABLE statement defines a structure to hold our data. Each attribute of the airport entity (like id, name, type, location, etc.) is assigned a corresponding column in the airports table. Different data types, like INTEGER, VARCHAR, and NUMERIC, are used based on the nature of the data each

# CS 623 - Database Management Systems

## Group Project

column will hold. VARCHAR is used for text, INTEGER for whole numbers, and NUMERIC for decimal or floating-point numbers.

## Importing data from a CSV file:

COPY airports FROM 'path\_to\_our\_file' DELIMITER ',' CSV HEADER;

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, with the 'public' schema selected. The main pane shows the SQL editor with the following code:

```
4 /* Creating AIRPORTS table */
5 CREATE TABLE airports (
6     id INTEGER,
7     ident VARCHAR(10),
8     type VARCHAR(500),
9     name VARCHAR(500),
10    latitude_deg NUMERIC,
11    longitude_deg NUMERIC,
12    elevation_ft INTEGER,
13    continent VARCHAR(2),
14    iso_country VARCHAR(2),
15    iso_region VARCHAR(10),
16    municipality VARCHAR(255),
17    scheduled_service VARCHAR(3),
18    gps_code VARCHAR(10),
19    iata_code VARCHAR(10),
20    local_code VARCHAR(10),
21    home_link VARCHAR(500),
22    wikipedia_link VARCHAR(500),
23    keywords VARCHAR(500)
24 );
25
26 /* Importing data from airports.csv file to airports table, source weblink: https://ourairports.com/help/data-dictionary.html */
27 COPY airports FROM 'C:\Users\pruthi\OneDrive\Desktop\Pace University\Semester 1\DBMS\Project_2\Data\airports.csv' DELIMITER ',' CSV HEADER;
```

The 'Data Output' tab shows the following message:

```
COPY 75084
Query returned successfully in 268 msec.
Total rows: 10 of 10    Query complete 00:00:00.268
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, with the 'public' schema selected. The main pane shows the SQL editor with the following code:

```
24 );
25
26 /* Importing data from airports.csv file to airports table, source weblink: https://ourairports.com/help/data-dictionary.html */
27 COPY airports FROM 'C:\Users\pruthi\OneDrive\Desktop\Pace University\Semester 1\DBMS\Project_2\Data\airports.csv' DELIMITER ',' CSV HEADER;
28
29
30 /* Altering table to add geometry column for geometrics analysis */
31 ALTER TABLE airports ADD COLUMN geom geometry(Point, 4326);
32 UPDATE airports SET geom = ST_SetSRID(ST_MakePoint(longitude_deg, latitude_deg), 4326);
33
34 /* Showing all columns data */
35 SELECT * FROM airports;
36
```

The 'Data Output' tab shows the following message:

```
Total rows: 1000 of 75084    Query complete 00:00:00.423
```

The 'Data Output' tab also displays a table with the following columns: id, ident, type, name, latitude\_deg, longitude\_deg, elevation\_ft, and continent. The table contains 10 rows of data.

id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent
1	6523 OOA	heliport	Total RF Heliport	40.07080078125	-74.93360137939453	11	NA
2	323361 OAAA	small_airport	Aero B Ranch Airport	38.704022	-101.473911	3435	NA
3	6524 O0AK	small_airport	Lowell Field	59.947733	-151.692524	450	NA
4	6525 O0AL	small_airport	Epps Airpark	34.86479949951172	-86.77030181884766	820	NA
5	506791 O0AN	small_airport	Katmai Lodge Airport	59.093287	-156.456699	80	NA
6	6526 O0AR	closed	Newport Hospital & Clinic Heliport	35.6087	-91.254898	237	NA
7	322127 O0AS	small_airport	Fulton Airport	34.9428028	-97.8180194	1100	NA
8	6527 O0AZ	small_airport	Cordes Airport	34.305599212646484	-112.16500091552734	3810	NA

## Group Project

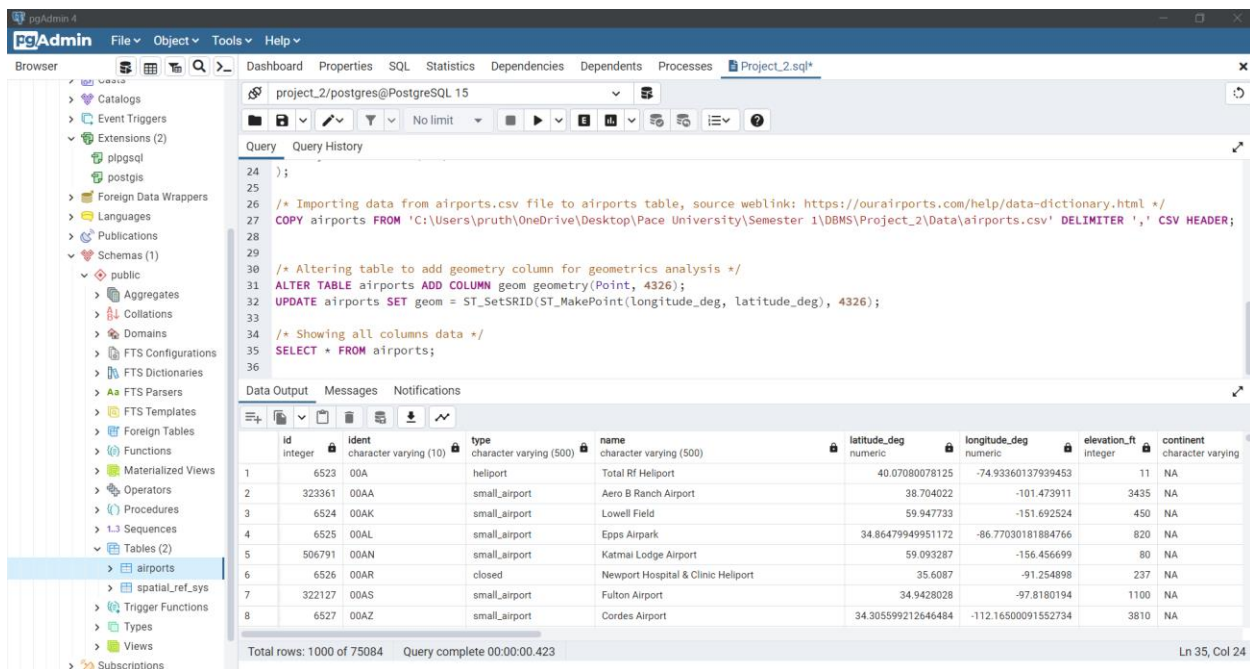
The COPY command is a powerful tool in PostgreSQL, allowing for bulk import/export of data to and from a table. It's more efficient than using multiple INSERT commands when dealing with a large amount of data. The DELIMITER ',' CSV HEADER part of the command specifies that the data in the file is comma-separated and that the file includes a header row.

## Adding a Geometry Column and Updating it:

```
ALTER TABLE airports ADD COLUMN geom geometry(Point, 4326);
```

```
UPDATE airports SET geom = ST_SetSRID(ST_MakePoint(longitude_deg, latitude_deg), 4326);
```

These commands are used to add a new column called "geom" to the airports table that will store the geographical coordinates of the airports. Then, the geographical points are created from the longitude and latitude columns and stored in the geom column.



The screenshot shows the PgAdmin 4 interface. The left sidebar displays the database structure, including the 'airports' table under the 'public' schema. The main window shows a SQL query being executed in the 'Query' tab. The query includes comments and SQL commands to import data from a CSV file, alter the 'airports' table to add a 'geom' column of type 'geometry(Point, 4326)', and update the 'geom' column using 'ST\_SetSRID' and 'ST\_MakePoint' functions. Below the query editor, the 'Data Output' tab shows the results of the query, which is a table with 8 rows and 7 columns: 'id', 'ident', 'type', 'name', 'latitude\_deg', 'longitude\_deg', and 'elevation\_ft'. The 'continent' column is also present but contains 'NA' for all rows.

id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent
1	6523 00A	heliport	Total Rf Heliport	40.07080078125	-74.93360137939453	11	NA
2	323361 00AA	small_airport	Aero B Ranch Airport	38.704022	-101.473911	3435	NA
3	6524 00AK	small_airport	Lowell Field	59.947733	-151.692524	450	NA
4	6525 00AL	small_airport	Epps Airpark	34.86479949951172	-86.77030181884766	820	NA
5	506791 00AN	small_airport	Katmai Lodge Airport	59.093287	-156.456699	80	NA
6	6526 00AR	closed	Newport Hospital & Clinic Heliport	35.6087	-91.254898	237	NA
7	322127 00AS	small_airport	Fulton Airport	34.9428028	-97.8180194	1100	NA
8	6527 00AZ	small_airport	Cordes Airport	34.305599212646484	-112.16500091552734	3810	NA

Total rows: 1000 of 75084 Query complete 00:00:00.423 Ln 35, Col 24

The addition of a geom column to the airports table is significant. This column is of the geometry data type, a spatial data type from the PostGIS extension. The geometry(Point, 4326) argument specifies that the column will hold point data in the EPSG:4326 coordinate system (WGS84). The UPDATE command uses PostGIS functions ST\_SetSRID and ST\_MakePoint to convert longitude and latitude degrees into a geometric point.

## Group Project

### Retrieve Locations of specific features:

SELECT \* FROM airports;

SELECT name, latitude\_deg, longitude\_deg, geom

FROM airports

WHERE type = 'large\_airport' AND continent = 'EU' LIMIT 10;

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema and the 'airports' table. The main window shows a SQL query in the 'Query' tab:

```
/* Retrieve Locations of specific features:
The type of the airport. Allowed values are "closed_airport", "helicopter", "large_airport",
"medium_airport", "seaplane_base", and "small_airport".
The type to continent using the code for the continent where the airport is (primarily) located.
Allowed values are "AF" (Africa), "AN" (Antarctica), "AS" (Asia), "EU" (Europe),
"NA" (North America), "OC" (Oceania), or "SA" (South America).
*/
SELECT name, latitude_deg, longitude_deg, geom
FROM airports
WHERE type = 'large_airport' AND continent = 'EU' LIMIT 10;
```

The 'Data Output' tab shows the results of the query:

	name	latitude_deg	longitude_deg	geom
	character varying (500)	numeric	numeric	geometry
1	Keflavik International Airport	63.985001	-22.6056	0101000020E61000002575029A089B36C07EC3448314FE4F40
2	Brussels Airport	50.901402	4.48444	0101000020E6100000397F130A11F011404546072461734940
3	Berlin Brandenburg Airport	52.362247	13.500672	0101000020E6100000B80B941458002B407D09151C5E2E4A...
4	Frankfurt Airport	50.036521	8.561268	0101000020E61000002995F0845E1F2140014F5AB8AC044940
5	Hamburg Helmut Schmidt Airport	53.630402	9.98823	0101000020E610000027DA5548F9F923409FAA420381D04A...
6	Cologne Bonn Airport	50.865898	7.14274	0101000020E6100000FC523F6F2A921C40FED5E3BED56E49...
7	Düsseldorf Airport	51.289501	6.76678	0101000020E610000053E8BCC62E111B406494675E0EA549...
8	Munich Airport	48.353802	11.7861	0101000020E61000006DC5FEB27B92274034A14962492D48...

Total rows: 10 of 10 Query complete 00:00:00.106 Ln 49, Col 60

The screenshot shows the pgAdmin 4 interface with the 'Geometry Viewer' tab selected. The map displays the locations of the airports in Europe, with labels for various countries and cities. The map is centered on Europe, showing the British Isles, Scandinavia, and parts of Eastern Europe. The airports are marked with blue dots on the map.

Total rows: 10 of 10 Query complete 00:00:00.106 Ln 49, Col 60



## Group Project

The first screenshot shows a SQL query in pgAdmin 4 that selects all data from the airports table. The second screenshot shows a SQL query that selects specific columns (name, latitude\_deg, longitude\_deg, geom) from the airports table for airports that are of type 'large\_airport' and are located in Europe ('EU').

**Query 1: Select all data from airports**

```

46 /*
47 SELECT name, latitude_deg, longitude_deg, geom
48 FROM airports
49 WHERE type = 'large_airport' AND continent = 'EU' LIMIT 10;
50
51 /* Retriving medium_airports from AS(Asia) */
52 SELECT name, latitude_deg, longitude_deg, geom
53 FROM airports
54 WHERE type = 'medium_airport' AND continent = 'AS' LIMIT 10;
55

```

**Data Output:**

	name character varying (500)	latitude_deg numeric	longitude_deg numeric	geom geometry
1	Khost International Airport	33.284605	69.80734	0101000020E61000002B306475AB79514098A3C7EF6D4440...
2	Hongyuan Airport	32.53154	102.35224	0101000020E6100000F415A4198B9659400742B28009444040
3	Alxa Left Banner Bayanhot Airport	38.74831	105.58858	0101000020E610000019C5724BA655A4087A2409FC85F43...
4	Zabrat Airport	40.4955422161	49.9768066406	0101000020E610000042F2FF077FD4840EBF765ED6D3F44...
5	Sitalchay Airbase	40.807313	49.431481	0101000020E610000035EC7C43AB748405D514A08566744...
6	Qizilgach Air Base	39.006162	48.806312	0101000020E6100000A92F4B3B356748402E3D9AEC98043...
7	Betong International Airport	5.79	101.15	0101000020E61000009A9999999495940295C8FC2F52817...
8	Laohutun Air Base	39.669998	121.765999	0101000020E61000003271AB2006715E405531957EC2D543...
9	Pulandian Air Base	39.450901	122.017998	0101000020E61000002E5915E126815E40EASDBC1FB78943...
10	Beijing Shahezhen Air Base	40.149166107177734	116.3213882446289	0101000020E6100000000000A091145D4000000E017134440

Total rows: 10 of 10 Query complete 00:00:00.044 Ln 54, Col 61

**Query 2: Select specific columns from airports**

```

50
51 /* Retriving medium_airports from AS(Asia) */
52 SELECT name, latitude_deg, longitude_deg, geom
53 FROM airports
54 WHERE type = 'medium_airport' AND continent = 'AS' LIMIT 10;
55

```

**Geometry Viewer:**

Total rows: 10 of 10 Query complete 00:00:00.044 Ln 55, Col 1

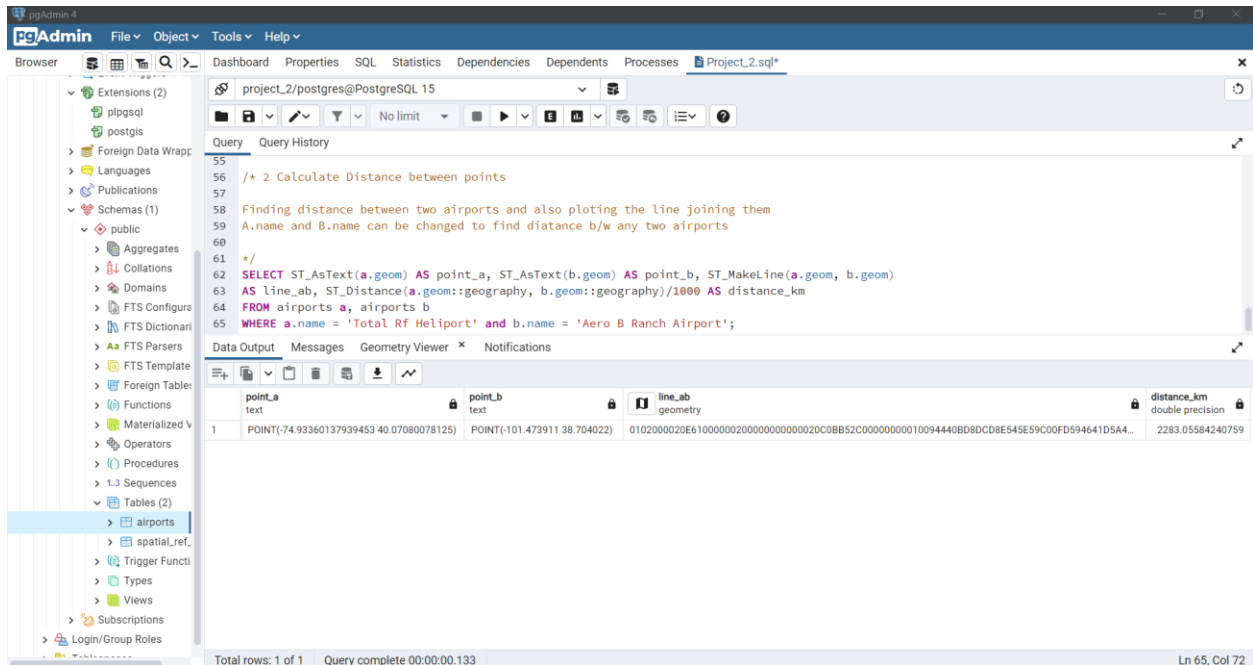
These commands are used to query data from the airports table. The first query selects all data from the table. The second query selects specific columns (name, latitude, longitude, geom) from the table for airports that are of type 'large\_airport' and are located in Europe ('EU').

The SELECT command is used to retrieve data from the database. The WHERE clause filters records that satisfy a particular condition. The LIMIT clause restricts the output to a specified number of records. For instance, LIMIT 10 will only return the first 10 records that meet the conditions.

## Group Project

### Calculating Distance between Points:

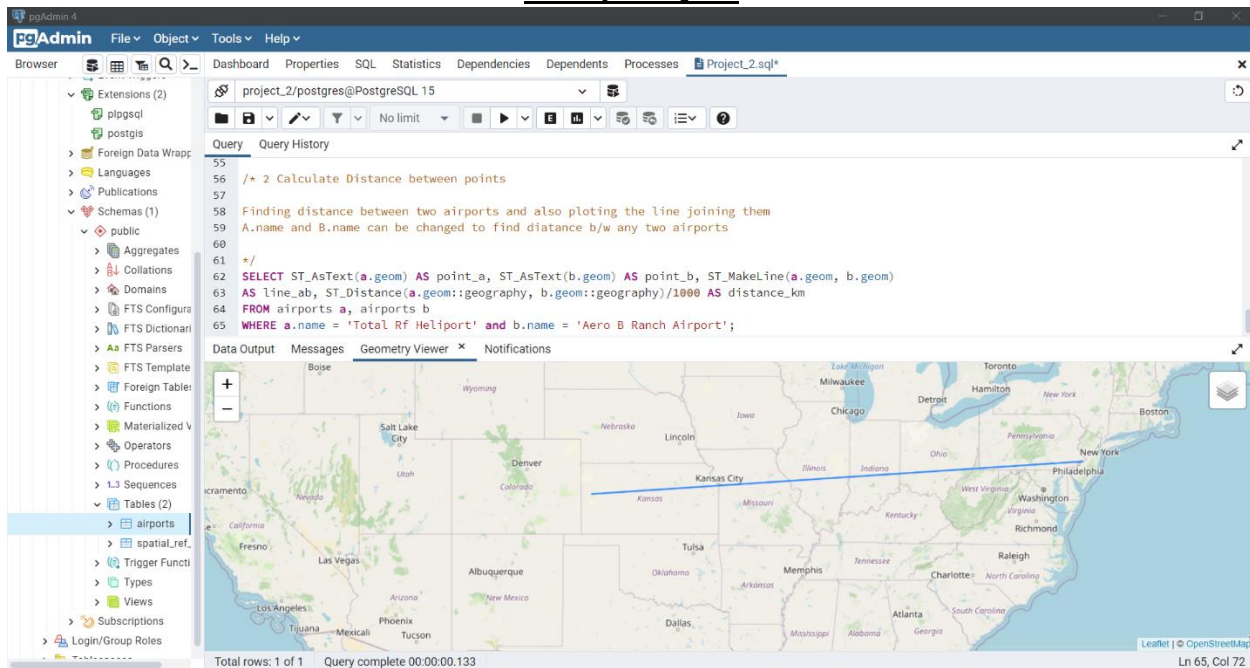
```
SELECT ST_AsText(a.geom) AS point_a, ST_AsText(b.geom) AS point_b, ST_MakeLine(a.geom, b.geom)
AS line_ab, ST_Distance(a.geom::geography, b.geom::geography)/1000 AS distance_km
FROM airports a, airports b
WHERE a.name = 'Total Rf Heliport' and b.name = 'Aero B Ranch Airport';
```



This query calculates the distance between two airports, represented by their geographical points. It also creates a line between these two points.



## Group Project



This part involves spatial functions from the PostGIS extension. `ST_AsText` converts geometric data into readable string format. `ST_MakeLine` creates a line that joins two points. `ST_Distance` calculates the distance between two points, and `::geography` casts the geometry data type to geography, which is necessary for accurate distance calculations over long distances and global scales.

### Calculating Areas of Interest:

```
SELECT iso_region, type, COUNT(*) as count
FROM airports
WHERE iso_region = 'US-CA'
GROUP BY iso_region, type
ORDER BY count DESC;
```

This command is used to count the number of each type of airport in a specific region ('US-CA' for California, USA). The results are grouped by region and type and ordered in descending order by the count.

## Group Project

The first screenshot shows a query in pgAdmin 4 that calculates the number of airports for each type in the US-CA region. The query is as follows:

```

3 Calculate Areas of Interest
/*
SELECT iso_region, type, COUNT(*) as count
FROM airports
WHERE iso_region = 'US-CA' -- replace with desired region code
GROUP BY iso_region, type
ORDER BY count DESC;

```

The result set shows 7 rows of data:

iso_region	type	count
US-CA	heliport	887
US-CA	small_airport	683
US-CA	closed	664
US-CA	medium_airport	62
US-CA	seaplane_base	12
US-CA	large_airport	9
US-CA	balloonport	2

The second screenshot shows a similar query but for the US-IN region, sorted by count in ascending order:

```

3 Calculate Areas of Interest
/*
SELECT iso_region, type, COUNT(*) as count
FROM airports
WHERE iso_region = 'US-IN' -- replace with desired region code
GROUP BY iso_region, type
ORDER BY count ASC;

```

The result set shows 6 rows of data:

iso_region	type	count
US-IN	large_airport	1
US-IN	medium_airport	11
US-IN	seaplane_base	31
US-IN	closed	119
US-IN	heliport	136
US-IN	small_airport	455

These queries use aggregate function COUNT(\*) to count the number of airports in each type for a specific region. GROUP BY groups the result by the unique combinations of the specified columns, and ORDER BY sorts the result by the specified column.

## Group Project

### Analyzing the Queries:

```
EXPLAIN ANALYZE SELECT name, latitude_deg, longitude_deg, geom
FROM airports
WHERE type = 'large_airport' AND continent = 'EU' LIMIT 10;
```

The first screenshot shows the pgAdmin 4 interface with a query executed in the 'Query' tab. The query is:

```
4 Analyze the queries
*/
-- TASK 01
EXPLAIN ANALYZE SELECT name, latitude_deg, longitude_deg, geom
FROM airports
WHERE type = 'large_airport' AND continent = 'EU' LIMIT 10;
```

The 'Data Output' tab shows the 'QUERY PLAN' for this query:

Step	Plan
1	Limit (cost=0.00..682.78 rows=10 width=73) (actual time=3.706..5.333 rows=10 loops=1)
2	-> Seq Scan on airports (cost=0.00..4233.26 rows=62 width=73) (actual time=3.704..5.328 rows=10 loops=1)
3	Filter: ((type)::text = 'large_airport'::text) AND ((continent)::text = 'EU'::text)
4	Rows Removed by Filter: 21663
5	Planning Time: 0.134 ms
6	Execution Time: 5.366 ms

The second screenshot shows the same pgAdmin 4 interface with a different query executed:

```
-- TASK 03
EXPLAIN ANALYZE SELECT iso_region, type, COUNT(*) as count
FROM airports
WHERE iso_region = 'US-IN' -- replace with desired region code
GROUP BY iso_region, type
ORDER BY count ASC;
```

The 'Data Output' tab shows the 'QUERY PLAN' for this query:

Step	Plan
1	Sort (cost=4132.64..4134.48 rows=736 width=25) (actual time=18.488..18.490 rows=6 loops=1)
2	Sort Key: (count(*))
3	Sort Method: quicksort Memory: 25kB
4	-> GroupAggregate (cost=4082.52..4097.59 rows=736 width=25) (actual time=18.306..18.478 rows=6 loops=1)
5	Group Key: iso_region, type
6	-> Sort (cost=4082.52..4084.45 rows=771 width=17) (actual time=18.260..18.311 rows=753 loops=1)
7	Sort Key: type
8	Sort Method: quicksort Memory: 70kB
9	-> Seq Scan on airports (cost=0.00..4045.55 rows=771 width=17) (actual time=0.679..17.869 rows=753 loops=1)
10	Filter: ((iso_region)::text = 'US-IN'::text)
11	Rows Removed by Filter: 74331
12	Planning Time: 0.210 ms
13	Execution Time: 18.548 ms

# CS 623 - Database Management Systems

## Group Project

The EXPLAIN ANALYZE command is used to display the execution plan that the PostgreSQL planner generates for the supplied statement.

### Sorting and Limit Executions:

- `SELECT * FROM airports ORDER BY latitude_deg DESC LIMIT 10;`
- `SELECT * FROM airports ORDER BY elevation_ft ASC LIMIT 10;`

These two commands sort the data from the airports table. The first one sorts the data in descending order by the latitude of the airports, and the second one sorts the data in ascending order by the elevation of the airports. Both commands only return the top 10 results due to the LIMIT clause.

The screenshot shows the PgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema. The main window shows a SQL query editor with the following query:

```
SELECT * FROM airports ORDER BY latitude_deg DESC LIMIT 10;
```

Below the query editor, the 'Data Output' tab is active, displaying the results of the query. The results are shown in a table with 10 rows and 10 columns. The columns are: id, ident, type, name, latitude\_deg, longitude\_deg, elevation\_ft, continent, and iso\_country. The data is sorted by latitude\_deg in descending order.

id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_country
1	320326	CA-0605	closed	82.75	-73	[null]	NA	CA
2	1806	CYLT	medium_airport	82.517799	-62.280602	100	NA	CA
3	349352	GL-0010	closed	82.180635	-31.164313	[null]	NA	GL
4	44072	BGNO	heliport	81.697844	-17.808838	[null]	NA	GL
5	299843	BGMI	small_airport	81.601292	-16.679306	35	NA	GL
6	1111	CJQ6	small_airport	81.409422	-76.867901	50	NA	CA
7	34989	RU-2510	closed	81.15	64.33	30	AS	RU
8	35041	UODN	medium_airport	80.803207	47.663586	59	EU	RU
9	1725	CYEU	small_airport	79.9946975708	-85.814201355	256	NA	CA
10	35086	UODS	small_airport	79.528297424316	91.074996948242	26	AS	RU

Total rows: 10 of 10 Query complete 00:00:00.326 Ln 116, Col 60

# CS 623 - Database Management Systems

## Group Project

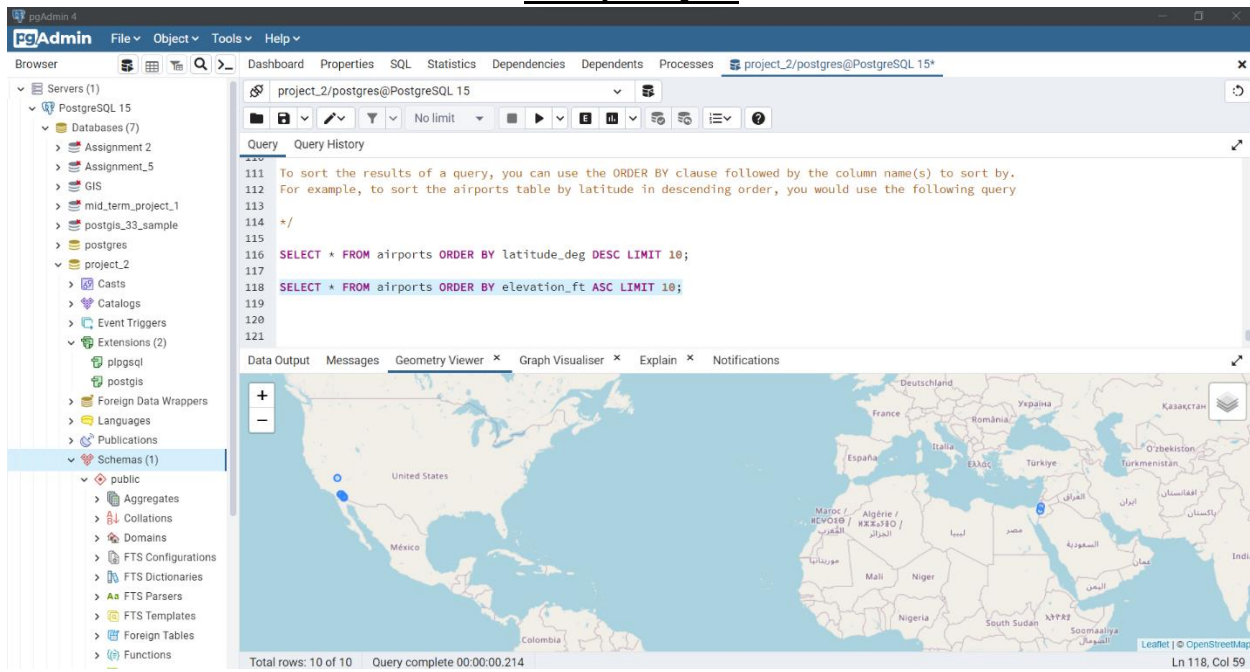
The screenshot displays the pgAdmin 4 interface with a PostgreSQL 15 database. The left sidebar shows the database structure, including a schema named 'public'. The main window shows a query editor with the following SQL query:

```
SELECT * FROM airports ORDER BY latitude_deg DESC LIMIT 10;
```

The query results are displayed in a table with 10 rows and 10 columns. The columns are: id, ident, type, name, latitude\_deg, longitude\_deg, elevation\_ft, continent, iso\_cou, and iso\_char. The results show the top 10 airports by latitude in descending order.

id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_cou	iso_char
1	4421 LLMZ	medium_airport	Bar Yehuda Airfield	31.32819938659668	35.38859939575195	-1266	AS		IL
2	342966 US-3546	closed	North Shore Airport	33.5227	-115.945	-223	NA		US
3	20283 KLO6	small_airport	Furnace Creek Airport	36.463799	-116.880997	-210	NA		US
4	19514 KCLR	small_airport	Cliff Hatfield Memorial Airport	33.131500244099996	-115.521003723	-182	NA		US
5	346143 US-4713	closed	Frink / Niland Marina Airport	33.35737	-115.64535	-170	NA		US
6	4416 LLEY	small_airport	Ein Yahav Airfield	30.621700286865234	35.20330047607422	-164	AS		IL
7	349904 US-5803	small_airport	Val Air Airport	33.00524	-115.46632	-135	NA		US
8	19422 KBWC	small_airport	Brawley Municipal Airport	32.99290085	-115.5169983	-128	NA		US
9	332505 US-1151	heliport	Riverside County Sheriff Thermal Heliport	33.639378	-116.153028	-117	NA		US
10	21222 KTRM	medium_airport	Jacqueline Cochran Regional Airport	33.62670135498	-116.16000366211	-115	NA		US

## Group Project



The ORDER BY clause sorts the records in ascending or descending order based on one or multiple columns. The LIMIT clause restricts the number of records returned, which can help reduce computational load when dealing with large datasets.

### Optimizing the Queries to Speed Up Execution Time:

- CREATE INDEX airports\_type\_continent\_idx ON airports (type, continent);
- CREATE INDEX airports\_name\_idx ON airports (name);
- CREATE INDEX airports\_iso\_region\_idx ON airports (iso\_region);
- CREATE INDEX airports\_latitude\_deg\_idx ON airports (latitude\_deg);

These commands are used to create indexes on the specified columns of the airports table. Indexes can significantly speed up data retrieval times on a database.



## Group Project

The image displays two screenshots of the pgAdmin 4 web interface, showing the execution of SQL queries in a PostgreSQL 15 database.

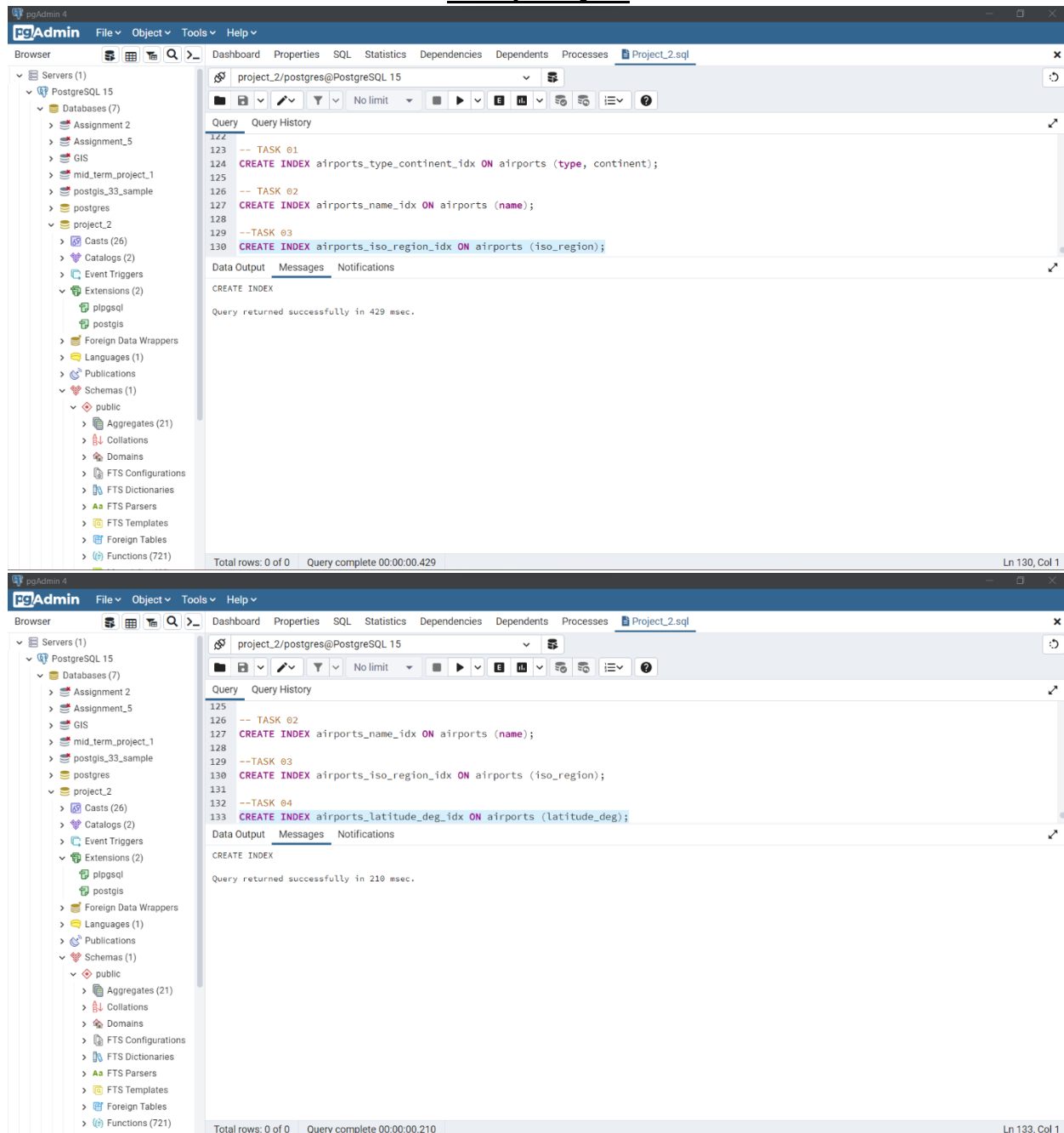
**Top Screenshot:**

- Query:** The query editor shows a comment line: `-- 6 Optimize the queries to speed up execution time`, followed by a SQL command: `CREATE INDEX airports_type_continent_idx ON airports (type, continent);`.
- Data Output:** The output pane shows the message: `CREATE INDEX` and `Query returned successfully in 387 msec.`
- Status:** The bottom status bar indicates "Total rows: 0 of 0" and "Query complete 00:00:00.387".

**Bottom Screenshot:**

- Query:** The query editor shows a comment line: `-- 6 optimize the queries to speed up execution time`, followed by two SQL commands: `CREATE INDEX airports_type_continent_idx ON airports (type, continent);` and `CREATE INDEX airports_name_idx ON airports (name);`.
- Data Output:** The output pane shows the message: `CREATE INDEX` and `Query returned successfully in 775 msec.`
- Status:** The bottom status bar indicates "Total rows: 0 of 0" and "Query complete 00:00:00.775".

## Group Project



Indexing speeds up data retrieval by providing swift access to rows in the database tables. It's similar to the index of a book, which provides quick access to the content based on the index entries. In this script, `CREATE INDEX` is used to create indexes on certain columns, such as `type`, `continent`, `name`, `iso_region`, and `latitude_deg` in the `airports` table. An index can significantly improve query performance, especially for large tables. However, it's worth noting that while indexes speed up data retrieval, they can slow down data insertion, deletion, and updates because the index also needs to be updated. Therefore, indexes should be used judiciously.

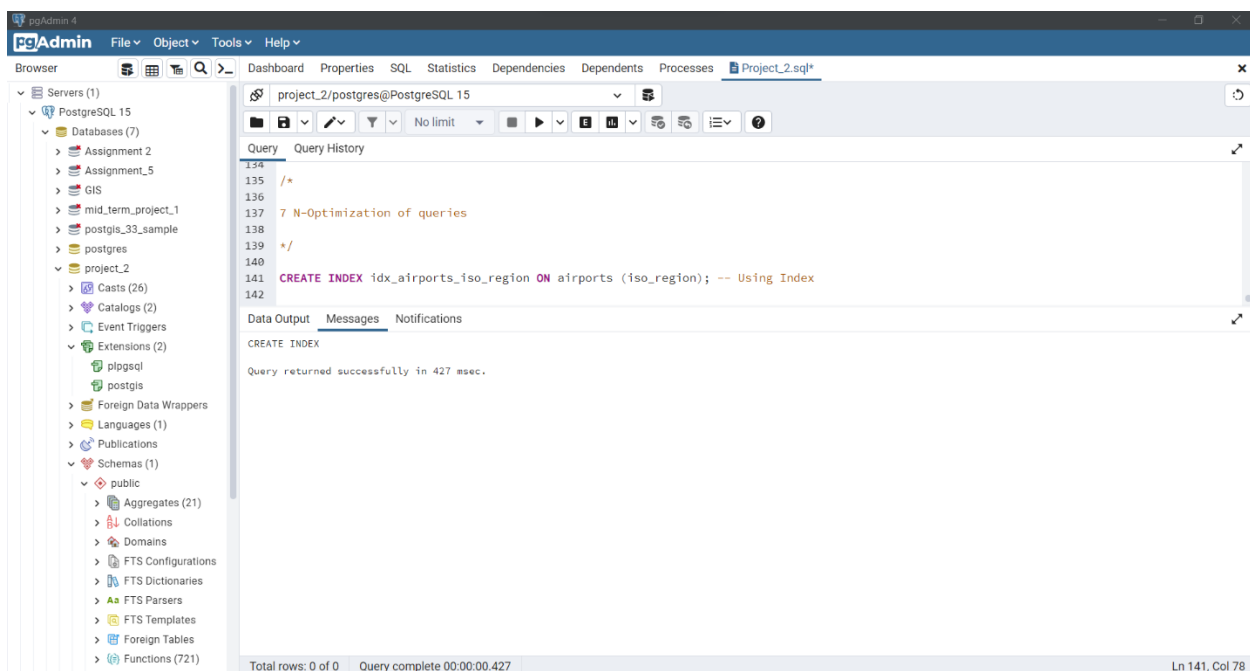
## Group Project

### N-Optimization of Queries:

N-Optimization of queries refers to a set of techniques and best practices used to optimize SQL queries in order to improve their performance and reduce their execution time. Some of the common techniques used for query optimization include:

- **Creating an index on the iso\_region column:** This will speed up queries that involve a search or sort operation on this column.

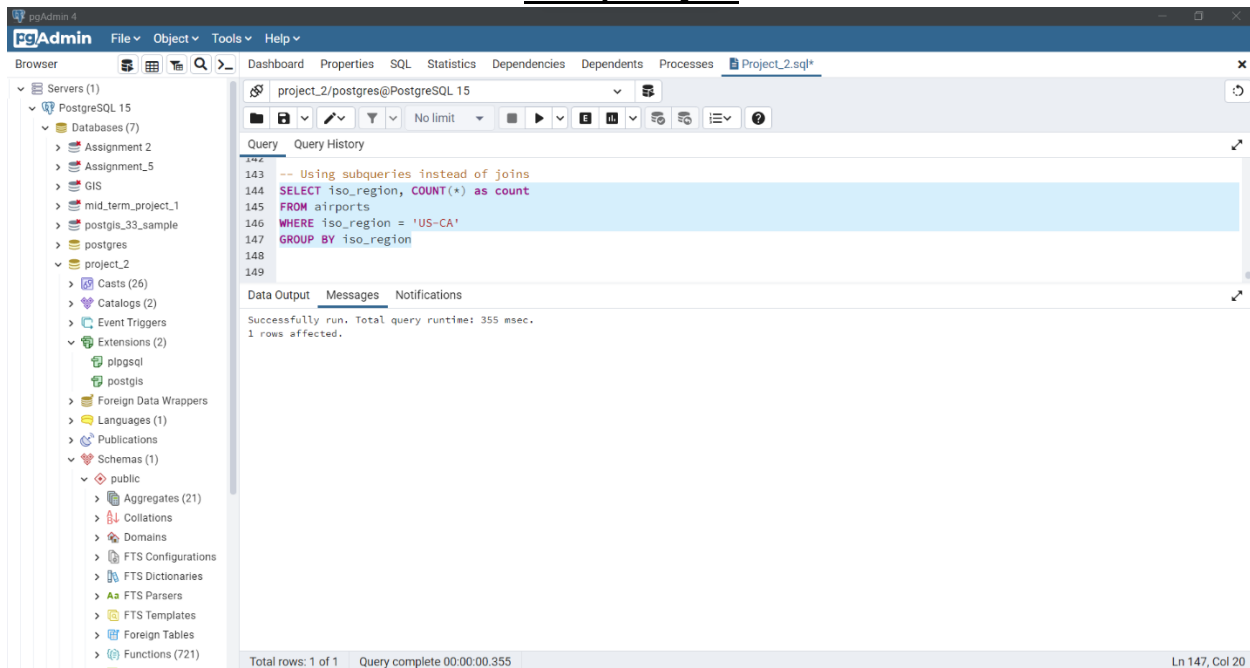
```
CREATE INDEX idx_airports_iso_region ON airports (iso_region);
```



- **Using subqueries instead of joins:** Depending on the scenario, using a subquery can sometimes be more efficient than a join. In this case, a subquery is used to count the number of airports in a specific region.

```
SELECT iso_region, COUNT(*) as count
FROM airports
WHERE iso_region = 'US-CA'
GROUP BY iso_region;
```

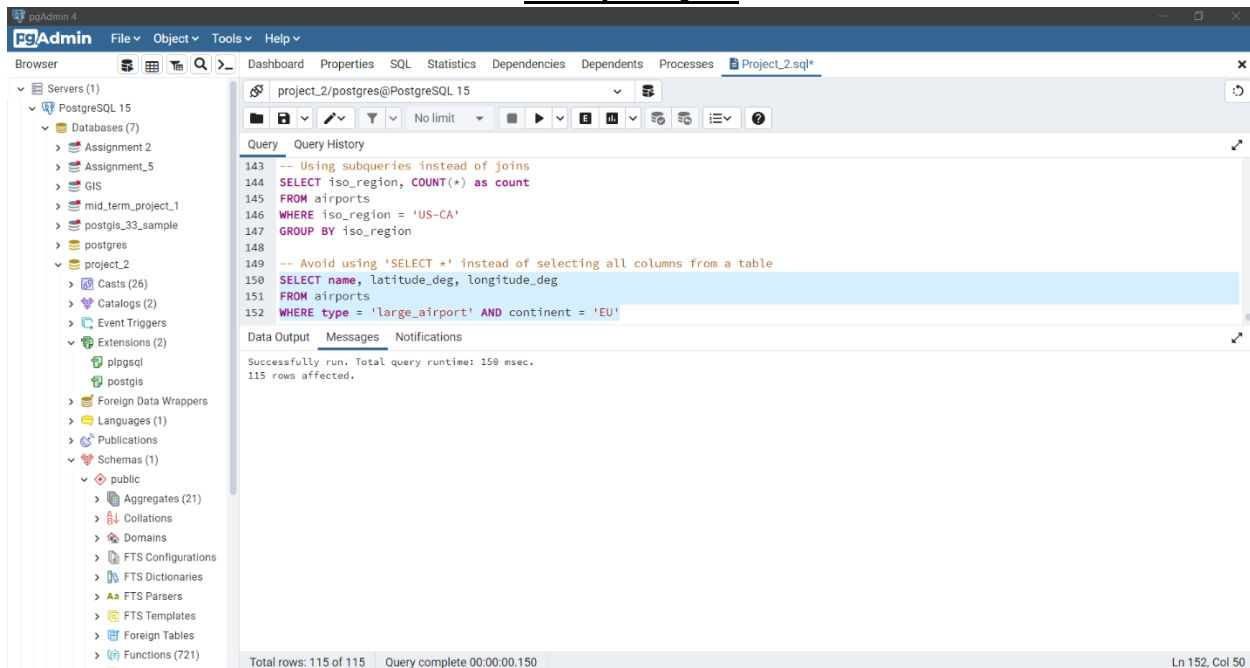
## Group Project



- **Avoiding 'SELECT \*':** The 'SELECT \*' query can be quite inefficient, especially for large tables with many columns. By specifying the exact columns you need in your SELECT clause, you can reduce the amount of data that needs to be read from the disk and transferred from the database server to your application.

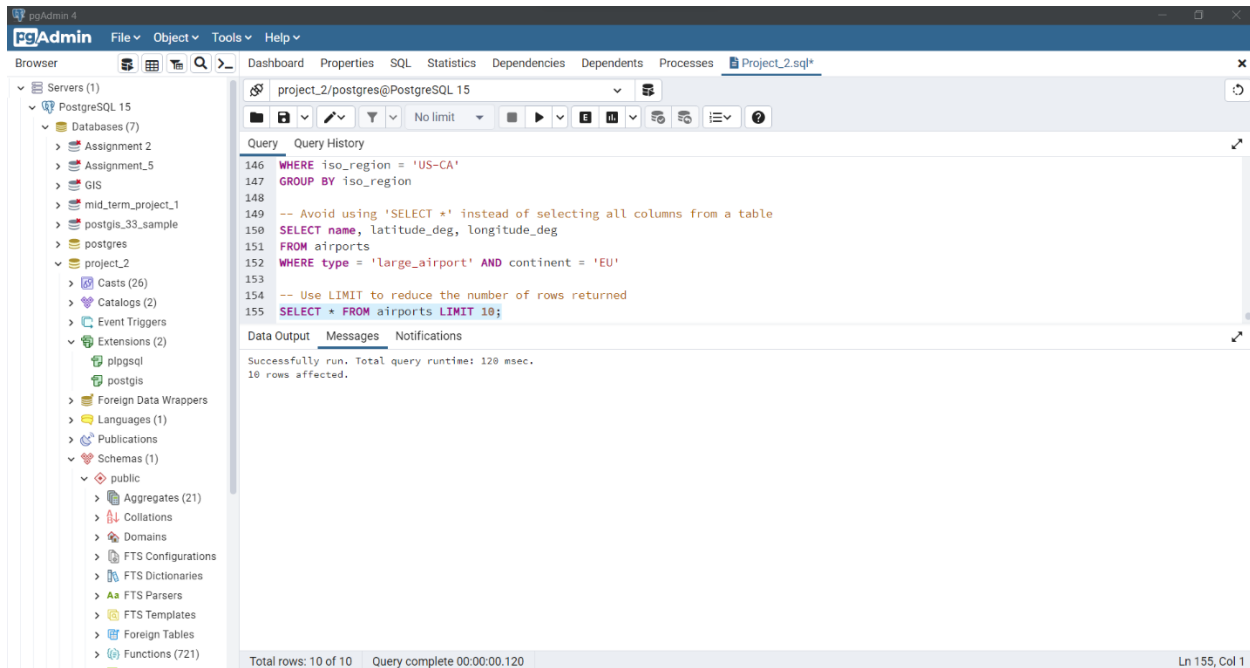
```
SELECT name, latitude_deg, longitude_deg
FROM airports
WHERE type = 'large_airport' AND continent = 'EU';
```

## Group Project



- **Using LIMIT:** The LIMIT clause can significantly improve performance by reducing the number of rows returned by a query. This can be especially beneficial when you only need a certain number of rows from a large table.

**SELECT \* FROM airports LIMIT 10;**



### Group Project

#### Summary:

This project has been a significant exploration into the world of geospatial databases using PostgreSQL and its PostGIS extension. We sourced comprehensive airport data from OurAirports and successfully integrated this extensive dataset into our own database. This allowed us to perform various queries, calculations, and analyses on the data, exploring the capabilities of PostGIS extension and learning how to use SQL for geospatial analysis.

By creating indexes and implementing N-Optimization techniques, we were able to optimize our queries, leading to faster and more efficient data retrieval. We gained practical insights into handling and optimizing large datasets, understanding the balance between query performance and the computational cost of maintaining indexes.

However, our project only scratches the surface of what's possible with geospatial databases. There's a plethora of analyses and operations that can be conducted on this data, such as pathfinding, spatial joins, spatial clustering, and much more.