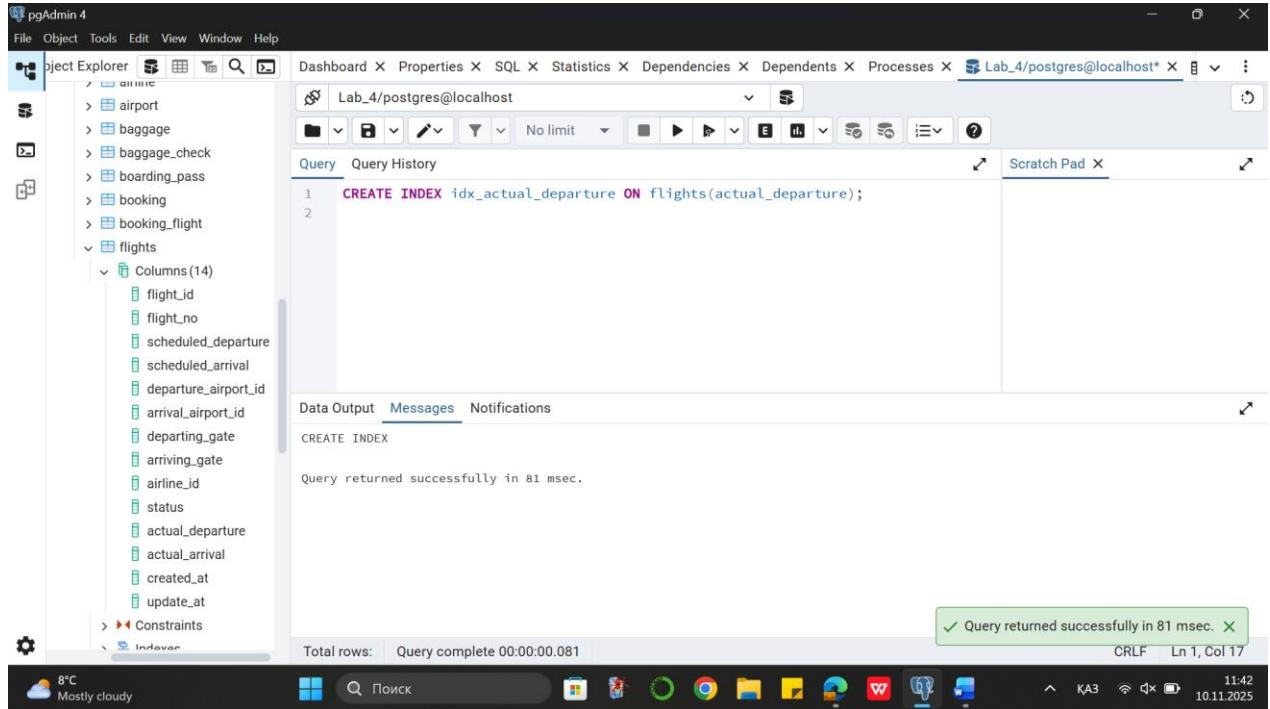


# Laboratory work 7

1. Create an index on the actual\_departure column in the flights table.



The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'flights' table, the 'Indexes' node is expanded. A new index is being created in the Query Editor:

```
CREATE INDEX idx_actual_departure ON flights(actual_departure);
```

The Messages tab shows the successful execution of the query:

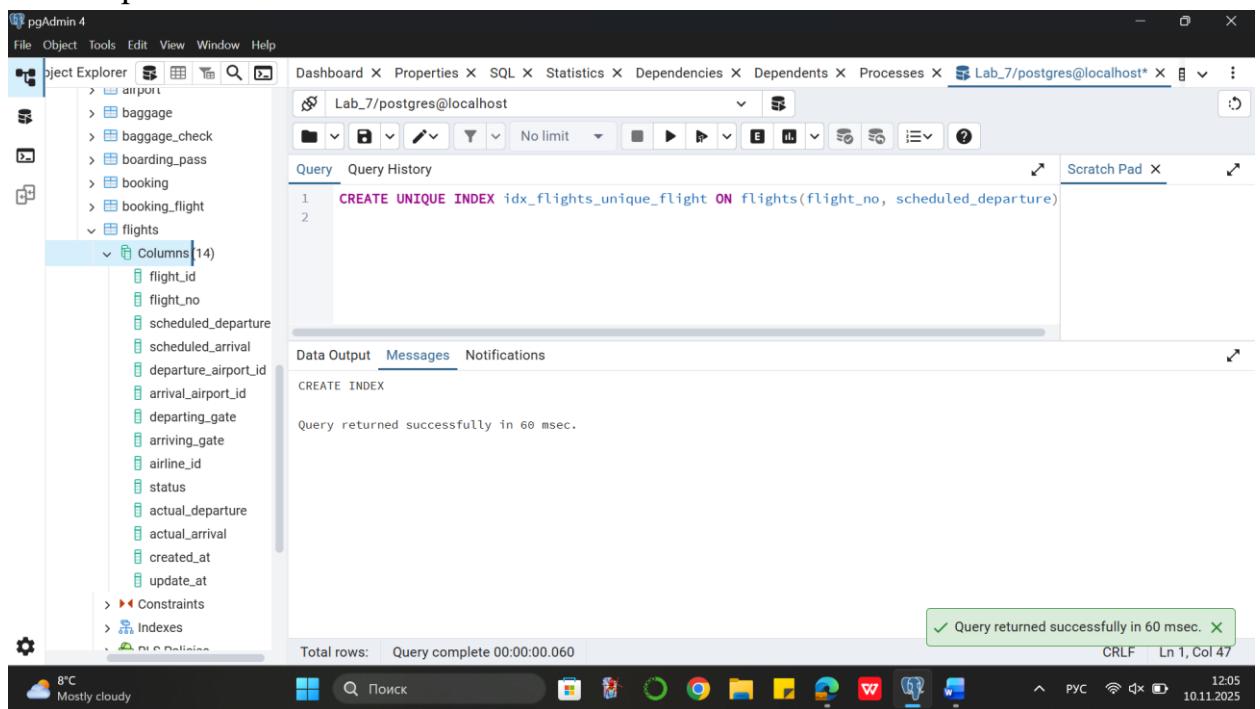
CREATE INDEX  
Query returned successfully in 81 msec.

Total rows: 0 Query complete 00:00:00.081

Query returned successfully in 81 msec. CRLF Ln 1, Col 17

11:42 10.11.2025

2. Create a unique index to ensure flight\_no and scheduled\_departure combinations are unique.



The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'flights' table, the 'Indexes' node is selected. A unique index is being created in the Query Editor:

```
CREATE UNIQUE INDEX idx_flights_unique_flight ON flights(flight_no, scheduled_departure)
```

The Messages tab shows the successful execution of the query:

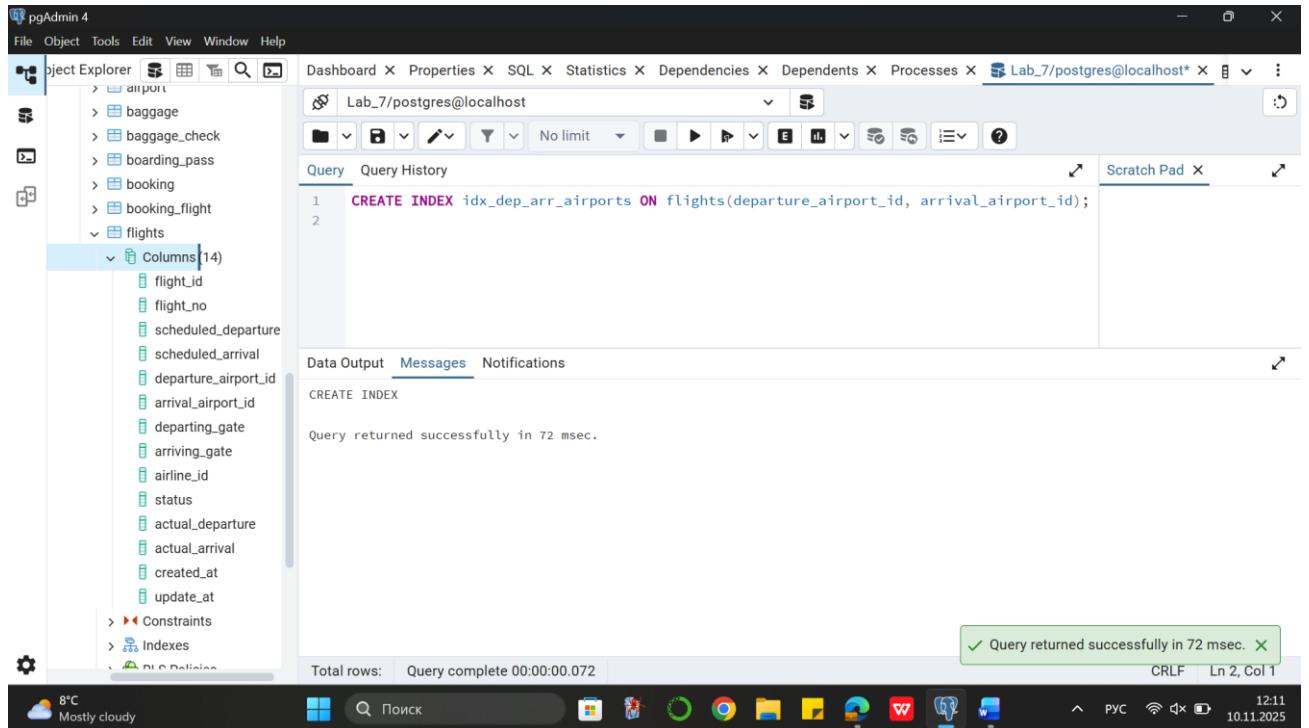
CREATE INDEX  
Query returned successfully in 60 msec.

Total rows: 0 Query complete 00:00:00.060

Query returned successfully in 60 msec. CRLF Ln 1, Col 47

12:05 10.11.2025

### 3. Create a composite index on the departure\_airport\_id and arrival\_airport\_id columns.



The screenshot shows the pgAdmin 4 interface. In the Object Explorer on the left, under the 'flights' table, the 'Columns' section is selected, showing 14 columns: flight\_id, flight\_no, scheduled\_departure, scheduled\_arrival, departure\_airport\_id, arrival\_airport\_id, departing\_gate, arriving\_gate, airline\_id, status, actual\_departure, actual\_arrival, created\_at, and update\_at. In the central SQL tab, the following query is run:

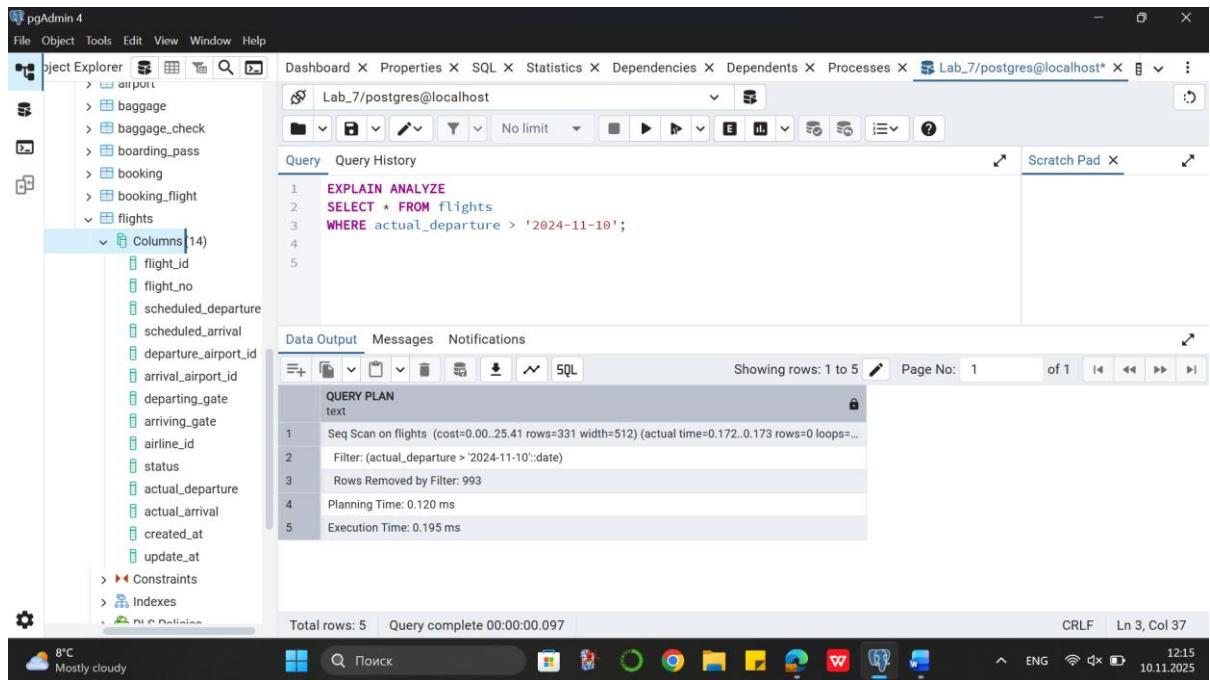
```
CREATE INDEX idx_dep_arr_airports ON flights(departure_airport_id, arrival_airport_id);
```

The 'Messages' tab shows the output:

```
CREATE INDEX
Query returned successfully in 72 msec.
```

A green success message at the bottom right of the interface also indicates: "Query returned successfully in 72 msec."

### 4. Evaluate the difference in query performance with and without indexes. Measure performance differences. Without indexes:



The screenshot shows the pgAdmin 4 interface. In the Object Explorer on the left, under the 'flights' table, the 'Columns' section is selected. In the central SQL tab, the following query is run:

```
EXPLAIN ANALYZE
SELECT * FROM flights
WHERE actual_departure > '2024-11-10';
```

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

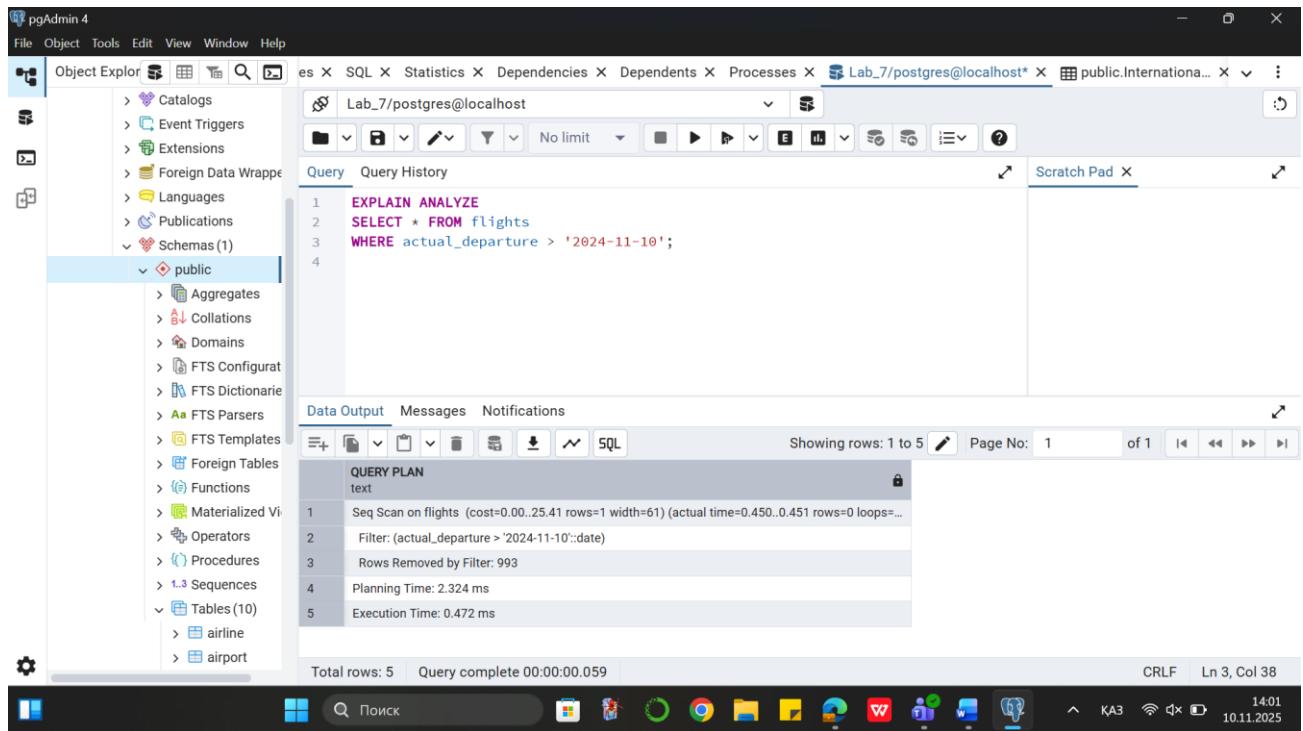
```
text
1 Seq Scan on flights (cost=0.00..25.41 rows=331 width=512) (actual time=0.172..0.173 rows=0 loops=...)
2 Filter: (actual_departure > '2024-11-10'::date)
3 Rows Removed by Filter: 993
4 Planning Time: 0.120 ms
5 Execution Time: 0.195 ms
```

The 'Messages' tab shows the output:

```
Showing rows: 1 to 5
Page No: 1
of 1
12:15
CRLF Ln 3, Col 37
```

A green success message at the bottom right of the interface indicates: "Query returned successfully in 00:00:00.097".

With indexes:



```
EXPLAIN ANALYZE
SELECT * FROM flights
WHERE actual_departure > '2024-11-10';
```

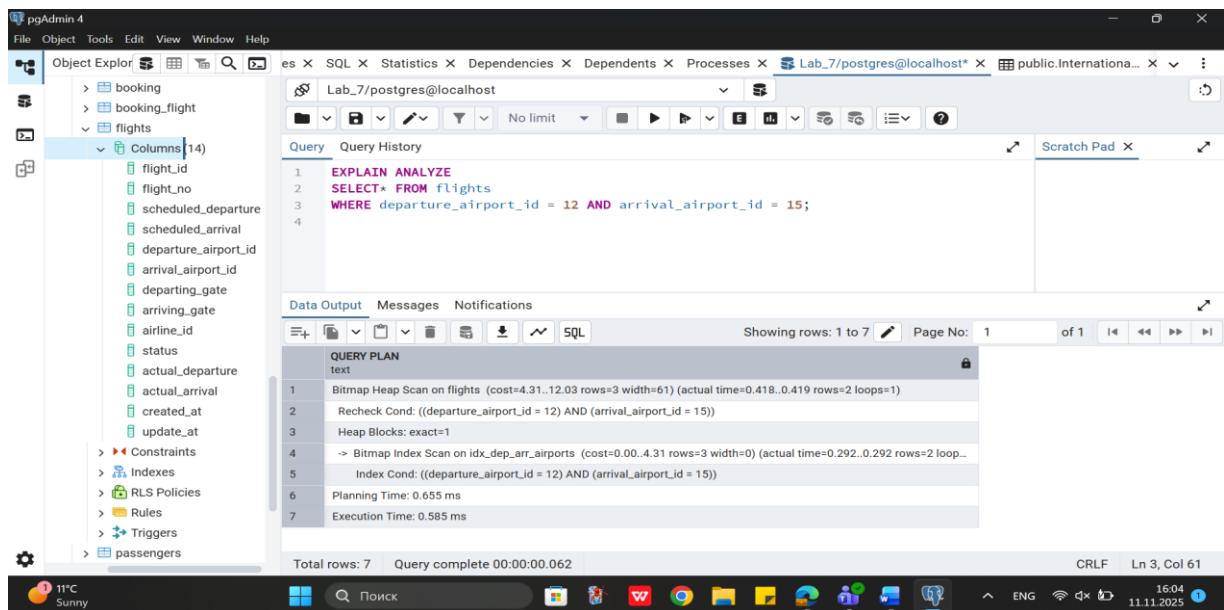
QUERY PLAN  
text

- Seq Scan on flights (cost=0.00..25.41 rows=1 width=61) (actual time=0.450..0.451 rows=0 loops=1)
- Filter: (actual\_departure > '2024-11-10'::date)
- Rows Removed by Filter: 993
- Planning Time: 2.324 ms
- Execution Time: 0.472 ms

Total rows: 5 Query complete 00:00:00.059

After the indexes were created, the query execution time decreased. Before creating the index, PostgreSQL used Seq Scan, which takes longer. After creating the index, PostgreSQL started using Index Scan, which selects only the necessary rows by indexes. This reduces the search time, especially for large tables.

## 5. Use EXPLAIN ANALYZE to check index usage in a query filtering by departure\_airport and arrival\_airport.



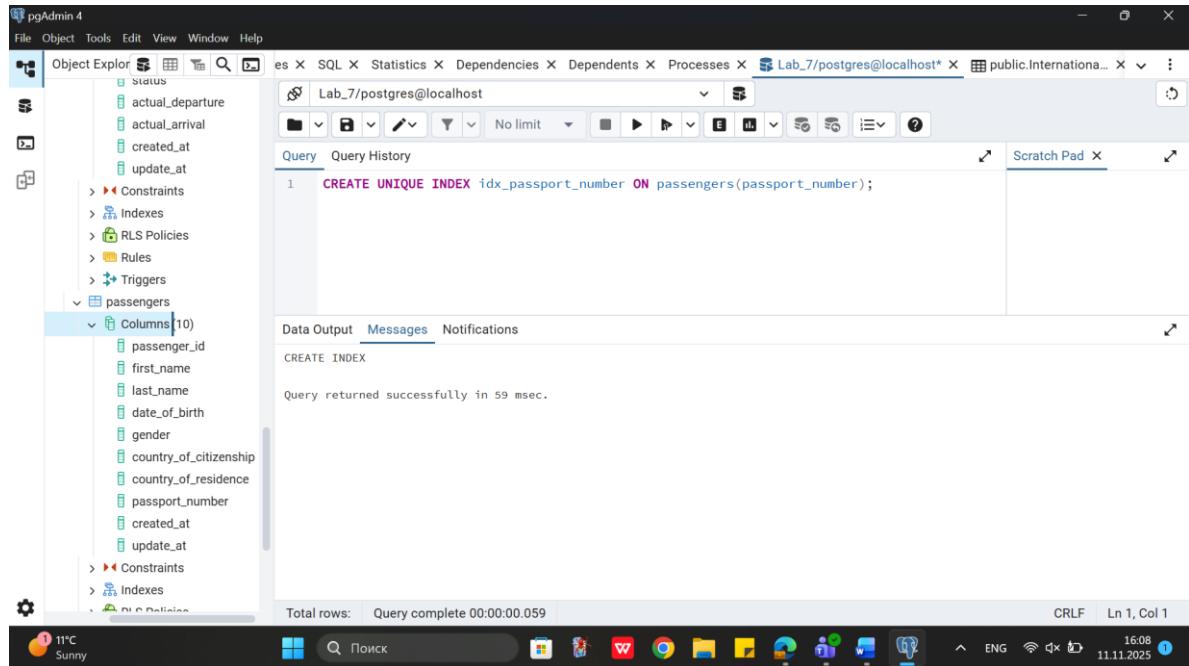
```
EXPLAIN ANALYZE
SELECT * FROM flights
WHERE departure_airport_id = 12 AND arrival_airport_id = 15;
```

QUERY PLAN  
text

- Bitmap Heap Scan on flights (cost=4.31..12.03 rows=3 width=61) (actual time=0.418..0.419 rows=2 loops=1)
- Recheck Cond: ((departure\_airport\_id = 12) AND (arrival\_airport\_id = 15))
- Heap Blocks: exact=1
- > Bitmap Index Scan on idx\_dep\_arr\_airports (cost=0.00..4.31 rows=3 width=0) (actual time=0.292..0.292 rows=2 loops=1)
- Index Cond: ((departure\_airport\_id = 12) AND (arrival\_airport\_id = 15))
- Planning Time: 0.655 ms
- Execution Time: 0.585 ms

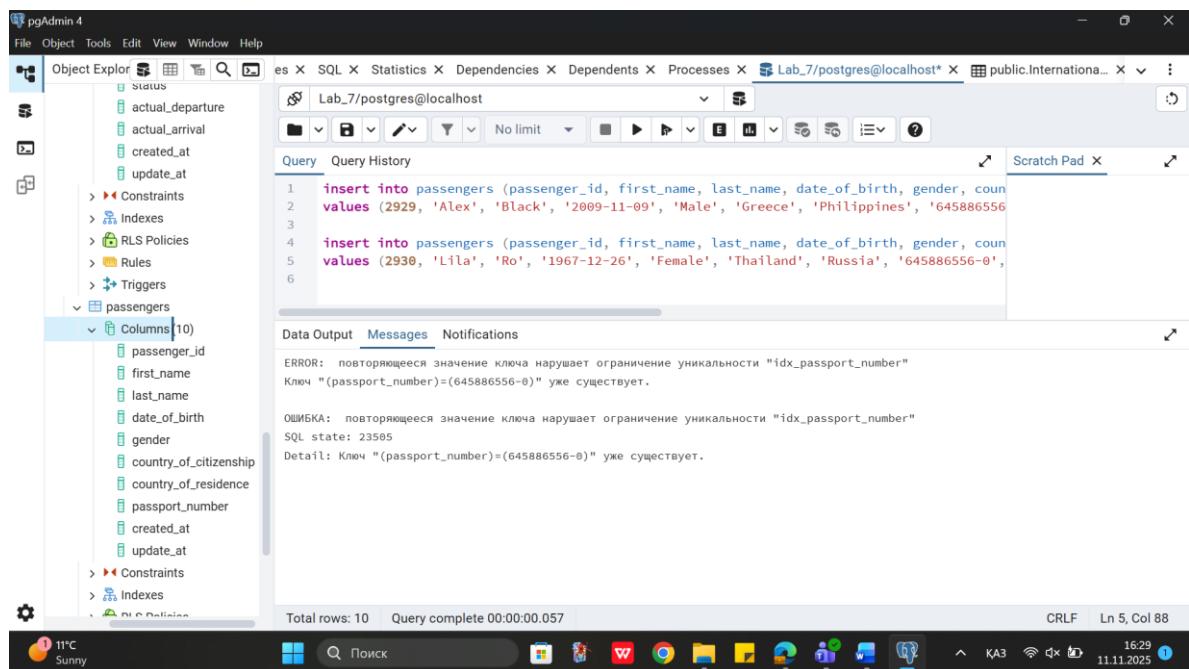
Total rows: 7 Query complete 00:00:00.062

6. Create a unique index for the passport\_number of the Passengers table. Check if the index was created or not. Insert into the table two new passengers. Explain in your own words what is going on in the output?



```
CREATE UNIQUE INDEX idx_passport_number ON passengers(passport_number);
```

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'passengers' table, the 'Indexes' node is expanded, showing the newly created 'idx\_passport\_number'. The 'Query' tab contains the SQL command to create the index. The 'Messages' tab shows the confirmation: 'CREATE INDEX' and 'Query returned successfully in 59 msec.'.

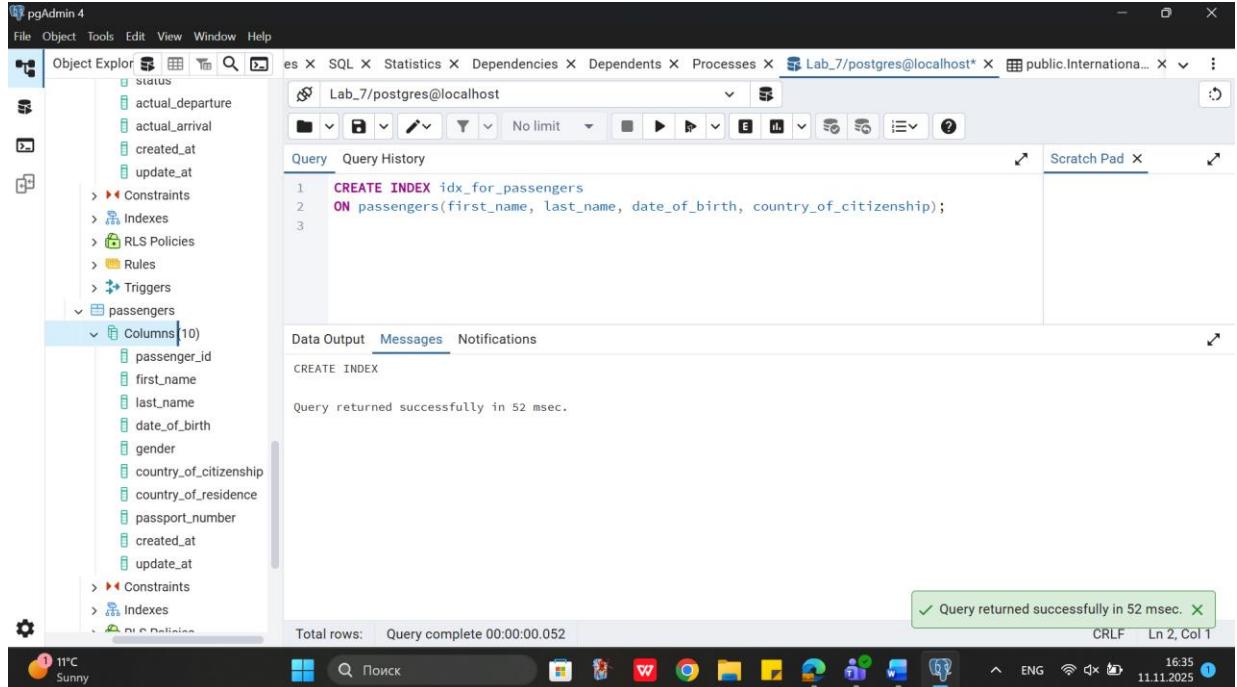
  


```
insert into passengers (passenger_id, first_name, last_name, date_of_birth, gender, country_of_citizenship, country_of_residence, passport_number, created_at, update_at)
values (2929, 'Alex', 'Black', '2009-11-09', 'Male', 'Greece', 'Philippines', '645886556-0',
       2930, 'Lila', 'Ro', '1967-12-26', 'Female', 'Thailand', 'Russia', '645886556-0');
```

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'passengers' table, the 'Indexes' node is expanded, showing the newly created 'idx\_passport\_number'. The 'Query' tab contains two INSERT statements. The 'Messages' tab displays an error message: 'ERROR: повторяющееся значение ключа нарушает ограничение уникальности "idx\_passport\_number" Ключ "(passport\_number)=(645886556-0)" уже существует.' (Error: A repeating key value violates the uniqueness constraint "idx\_passport\_number". Key "(passport\_number)=(645886556-0)" already exists.)

When trying to insert a passenger with an existing passport number, PostgreSQL returned an error. This proves that the unique idx\_passport\_number index has been successfully created and is working correctly. It prevents duplicate passport numbers in the passengers table.

7. Create an index for the Passengers table. Use for that first name, last name, date of birth and country of citizenship. Then, write a SQL query to find a passenger who was born in Philippines and was born in 1984 and check if the query uses indexes or not. Give the explanation of the results.



The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'passenger' table, a new index named 'idx\_for\_passenger' is being created with the following SQL command:

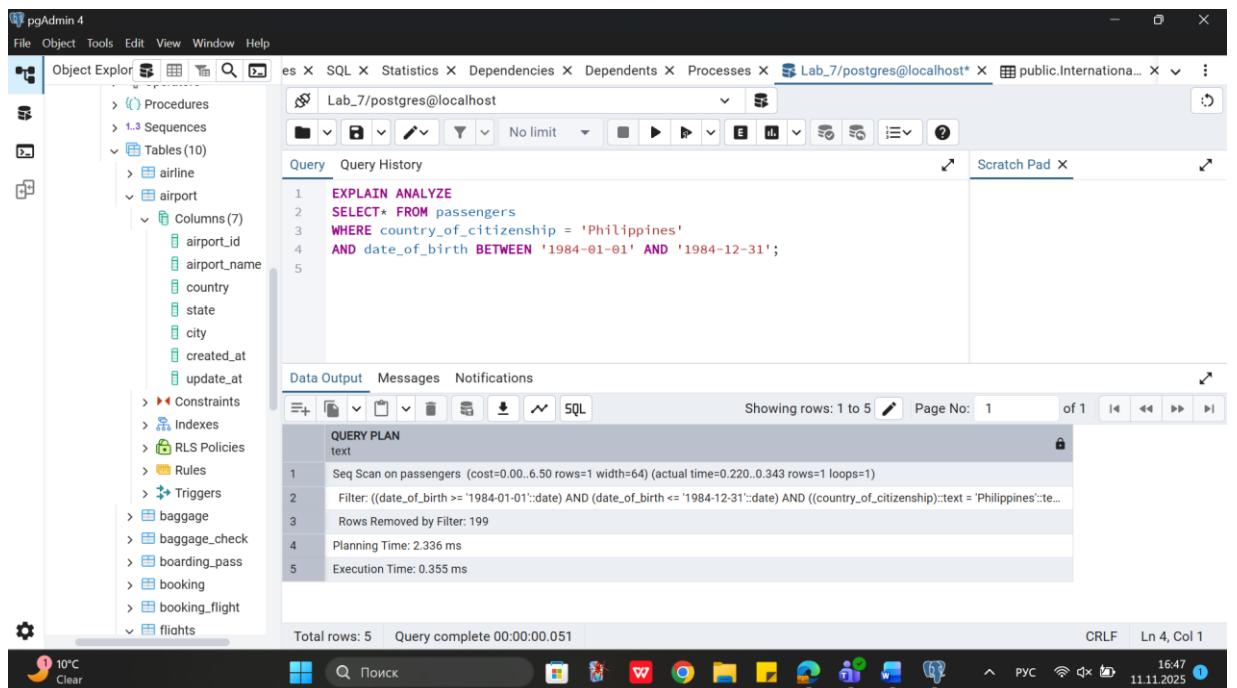
```
CREATE INDEX idx_for_passenger
ON passengers(first_name, last_name, date_of_birth, country_of_citizenship);
```

The 'Messages' tab shows the execution message:

```
CREATE INDEX
```

Query returned successfully in 52 msec.

Total rows: Query complete 00:00:00.052 CRLF Ln 2, Col 1



The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'airline' table, the 'airport' sub-table is selected. A query is run using EXPLAIN ANALYZE:

```
EXPLAIN ANALYZE
SELECT * FROM passengers
WHERE country_of_citizenship = 'Philippines'
AND date_of_birth BETWEEN '1984-01-01' AND '1984-12-31';
```

The 'Data Output' tab displays the query plan:

```
QUERY PLAN
text
1 Seq Scan on passengers (cost=0.00..6.50 rows=1 width=64) (actual time=0.220..0.343 rows=1 loops=1)
  Filter: ((date_of_birth >='1984-01-01':date) AND (date_of_birth <='1984-12-31':date) AND ((country_of_citizenship)::text = 'Philippines'::text))
  Rows Removed by Filter: 199
2 Planning Time: 2.336 ms
3 Execution Time: 0.355 ms
```

Total rows: 5 Query complete 00:00:00.051 CRLF Ln 4, Col 1

A composite index of 4 fields has been created. But in the query, filtering was performed on only two of these fields. The index was not used because the search did not start from the first index field.

8. Write a SQL query to list indexes for table Passengers. After delete the created indexes.

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'Tables' node, the 'passenger' table is selected. In the main pane, a query is run:

```
1 SELECT indexname, indexdef FROM pg_indexes WHERE tablename = 'passenger';
```

The results table shows three indexes:

indexname	indexdef
passenger_pkey	CREATE UNIQUE INDEX passenger_pkey ON public.passenger USING btree (passenger_id)
idx_passport_number	CREATE UNIQUE INDEX idx_passport_number ON public.passenger USING btree (passport_number)
idx_for_passenger	CREATE INDEX idx_for_passenger ON public.passenger USING btree (first_name, last_name, date_of_birth, country_of_citizenship)

Total rows: 3 | Query complete 00:00:01.133 | CRLF | Ln 1, Col 28

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'Tables' node, the 'passenger' table is selected. In the main pane, the following SQL statements are run:

```
1 DROP INDEX idx_passport_number;
2 DROP INDEX idx_for_passenger;
```

The 'Messages' tab shows the output:

DROP INDEX  
Query returned successfully in 103 msec.

Query returned successfully in 103 msec. | CRLF | Ln 3, Col 1