

- 1) Create an index on the actual_departure column in the flights table.

The screenshot shows the pgAdmin 4 interface. A query window titled 'lab_7/postgres@PostgreSQL 17*' contains the SQL command:

```
1 create index f_actual_departure on flights(actual_departure)
```

The 'Messages' tab at the bottom displays an error message:

```
ERROR: relation "f_actual_departure" already exists
```

SQL state: 42P07

The system tray at the bottom left shows an 'AWO1' icon with '+0.13%' and a battery level of 100%. The taskbar at the bottom right shows various application icons and the date/time: 15:59 11.11.2025.

- 2) Create a unique index to ensure flight_no and scheduled_departure combinations are unique.

The screenshot shows the pgAdmin 4 interface. A query window titled 'lab_7/postgres@PostgreSQL 17*' contains the SQL command:

```
1 v CREATE UNIQUE INDEX f_no_sch_dep
2 ON flights (flight_no, scheduled_departure);
```

The 'Messages' tab at the bottom displays an error message:

```
ERROR: could not create unique index "f_no_sch_dep"
Key (flight_no, scheduled_departure)=(US-KS, 2023-09-04) is duplicated.
```

SQL state: 23505
Detail: Key (flight_no, scheduled_departure)=(US-KS, 2023-09-04) is duplicated.

The system tray at the bottom left shows an 'Air Moderate Tomorrow' icon and a battery level of 100%. The taskbar at the bottom right shows various application icons and the date/time: 17:02 11.11.2025.

- 3) Create a composite index on the departure_airport_id and arrival_airport_id columns.

pgAdmin 4

Welcome lab_7/postgres@PostgreSQL 17*

```
1 ✓ create index ind_fl_airport
2   on flights(departure_airport_id, arrival_airport_id)
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 67 msec.

Total rows: Query complete 00:00:00.067

Watchlist +154%

Query History Scratch Pad

✓ Query returned successfully in 67 msec. CRLF Ln 2, Col 1

17:07 11.11.2025

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

pgAdmin 4

Welcome lab_7/postgres@PostgreSQL 17*

```
1 ✓ EXPLAIN (ANALYZE, BUFFERS, TIMING)
2   SELECT *
3     FROM flights
4    WHERE airline_id = 101
5      AND departure_airport_id = 10
6      AND actual_departure >= DATE '2025-11-01';
7
8
9 ✓ CREATE INDEX IF NOT EXISTS idx_flights_airline_dep_act
10   ON flights (airline_id, departure_airport_id, actual_departure);
11
12 ANALYZE flights;
13
14
15 ✓ EXPLAIN (ANALYZE, BUFFERS, TIMING)
16   SELECT *
17     FROM flights
18    WHERE airline_id = 101
19      AND departure_airport_id = 10
20      AND actual_departure >= DATE '2025-11-01';
21
```

Data Output Messages Notifications

QUERY PLAN

1	Index Scan using idx_flights_airline_dep_act on flights (cost=0.28..8.30 rows=1 width=61) (actual time=0.004..0.005 rows=0 loops=..)
2	Index Cond: ((airline_id = 101) AND (departure_airport_id = 10) AND (actual_departure >= 2025-11-01..date))
3	Buffers: shared hit=2
4	Planning:
5	Buffers: shared hit=55
6	Planning Time: 0.188 ms

Total rows: 7 Query complete 00:00:00.125

✓ Successfully run. Total query runtime: 125 msec. 7 rows affected. CRLF Ln 13, Col 1

S&P 500 +1.54% Помощь 17:44 11.11.2025

5. Use EXPLAIN ANALYZE to check index usage in a query filtering by departure_airport and arrival_airport.

```
1 v create index if not exists fl_airport
2   on flights(departure_airport_id, arrival_airport_id);
3
4 analyze flights;
5
6 v explain(analyze)
7 select *
8 from flights
9 where departure_airport_id = 10
10   and arrival_airport_id = 25;
11
12
13
14
15
16
17
18
19
```

QUERY PLAN
text

```
1 Index Scan using fl_airport on flights (cost=0.28 8.29 rows=1 width=61) (actual time=0.188..0.188 rows=0 loops=...)
2 Index Cond: ((departure_airport_id = 10) AND (arrival_airport_id = 25))
3 Planning Time: 0.421 ms
4 Execution Time: 0.223 ms
```

Total rows: 4 Query complete 00:00:00.080

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?

```
1 v CREATE unique INDEX idx_passport_number
2   ON passengers (passport_number);
3
4 v SELECT indexname, indexdef
5   FROM pg_indexes
6   WHERE tablename = 'passengers';
7
8 v SELECT *
9   FROM passengers;
10
11 v INSERT INTO passengers (passenger_id, first_name, last_name, date_of_birth, gender, country_of_citizenship, country_of_res
12   VALUES (3454, 'Ali', 'Turgan', '2000-12-12', 'Male', 'Kazakhstan', 'USA', '438547349', '2025-11-11', '2025-11-11');
13 v INSERT INTO passengers (passenger_id, first_name, last_name, date_of_birth, gender, country_of_citizenship, country_of_res
14   VALUES (3456, 'Ali', 'Turgan', '2000-12-12', 'Male', 'Kazakhstan', 'USA', '382385347-3', '2025-11-11', '2025-11-11');
```

ERROR: duplicate key value violates unique constraint "idx_passport_number"
Key (passport_number)=(382385347-3) already exists.

SQL state: 23505
Detail: Key (passport_number)=(382385347-3) already exists.

Total rows: 0 Query complete 00:00:00.052

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 top-left pane, there is a tree view with a node labeled 'Query History'. In the main central area, a SQL query is written:

```
1 ✓ create index idx_passengers
2   on passengers(first_name, last_name, date_of_birth, country_of_citizenship)
3
4 select * from passengers
5 where extract(year from date_of_birth) = 1984
6   and country_of_citizenship = 'Philippines'
```

Below the query, a large blue button with a play icon is visible, indicating the query is ready to be executed. At the bottom of the interface, there is a status bar showing 'Total rows: 1' and 'Query complete 00:00:00.124'.

The bottom part of the screenshot shows a Windows taskbar with various icons and system information like battery level (0.25%), network (ENG), and time (20:44, 11.11.2025).

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 top-left pane, there is a tree view with a node labeled 'Query History'. In the main central area, a SQL query is written:

```
1 ✓ select *
2   from pg_indexes
3   where tablename= 'passengers'
4
5 drop index idx_passport_number
```

At the bottom of the interface, there is a status bar showing 'Total rows: 1' and 'Query complete 00:00:00.101'.

The bottom part of the screenshot shows a Windows taskbar with various icons and system information like battery level (Air: Moderate Tomorrow), network (ENG), and time (20:51, 11.11.2025). A green message box at the bottom right says 'Query returned successfully in 101 msec.'