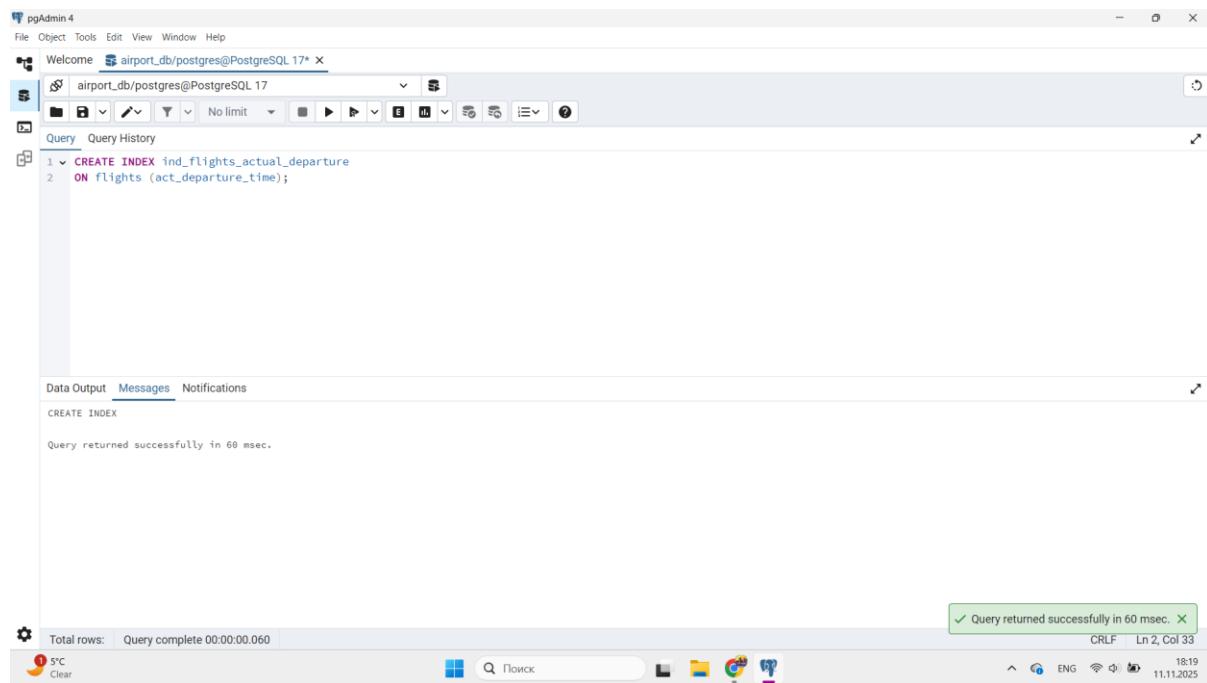


1. Create an index on the actual_departure column in the flights table.



The screenshot shows the pgAdmin 4 interface. A query window is open with the following SQL command:

```
CREATE INDEX ind_flights_actual_departure  
ON flights (act_departure_time);
```

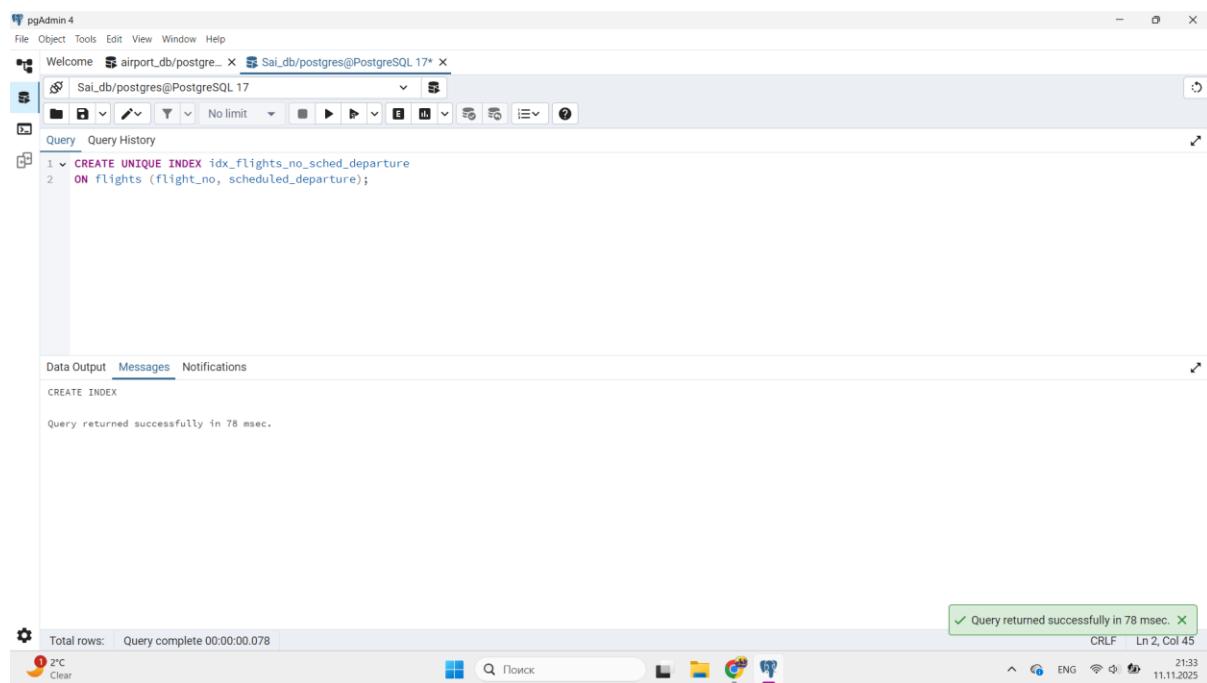
The 'Messages' tab shows the output:

CREATE INDEX

Query returned successfully in 60 msec.

The status bar at the bottom right indicates: Total rows: Query complete 00:00:00.060, CRLF, Ln 2, Col 33, 18:19, 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 is open with the following SQL command:

```
CREATE UNIQUE INDEX idx_flights_no_sched_departure  
ON flights (flight_no, scheduled_departure);
```

The 'Messages' tab shows the output:

CREATE INDEX

Query returned successfully in 78 msec.

The status bar at the bottom right indicates: Total rows: Query complete 00:00:00.078, CRLF, Ln 2, Col 45, 21:33, 11.11.2025.

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

The screenshot shows the pgAdmin 4 interface. At the top, there's a menu bar with File, Object, Tools, Edit, View, Window, Help. Below the menu is a toolbar with various icons. There are two tabs open in the main area: 'Welcome' and 'Sai_db/postgres@PostgreSQL 17*'. The current tab, 'Sai_db/postgres@PostgreSQL 17*', contains a query editor with the following SQL code:

```
1 ✓ CREATE INDEX idx_flights_departure_arrival
2   ON flights (departure_airport_id, arrival_airport_id);
```

Below the query editor, there are three tabs: Data Output, Messages, and Notifications. The 'Data Output' tab is selected and shows the message: 'CREATE INDEX'. Underneath it, it says 'Query returned successfully in 68 msec.'.

At the bottom of the window, there's a status bar with the message 'Query complete 00:00:00.068', the file path 'C:\Users\2\PycharmProjects\airline\airline\models.py', and a timestamp '21:34 11.11.2025'.

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

```
SET enable_seqscan = ON;
SET enable_indexscan = OFF;
EXPLAIN ANALYZE
SELECT *
FROM flights
WHERE departure_airport_id = 3
    AND arrival_airport_id = 7;
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

QUERY PLAN text

```
1 Seq Scan on flights (cost=0.00..27.89 rows=2 width=61) (actual time=0.164..0.309 rows=1 loops=...)
2   Filter: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
3   Rows Removed by Filter: 992
4 Planning Time: 1.556 ms
5 Execution Time: 0.334 ms
```

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

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 137 msec.

```
Total rows: 5 Query complete 00:00:00.084
```

Query returned successfully in 137 msec.

The screenshot shows the pgAdmin 4 interface. In the top navigation bar, the 'File' and 'Object' tabs are visible. Below the navigation bar, there are two tabs: 'Welcome' and 'Sai_db/postgres@PostgreSQL 17*'. The current tab is 'Sai_db/postgres@PostgreSQL 17*'. The main area contains a SQL editor with the following query:

```
SET enable_seqscan = OFF;
SET enable_indexscan = ON;
EXPLAIN ANALYZE
SELECT *
FROM flights
WHERE departure_airport_id = 3
    AND arrival_airport_id = 7;
```

Below the SQL editor, there is a 'Data Output' tab selected, followed by 'Messages' and 'Notifications'. The 'Data Output' tab displays the execution plan and results. The 'QUERY PLAN' section shows the following steps:

- 1 Bitmap Heap Scan on flights (cost=4.30..9.97 rows=2 width=61) (actual time=0.097..0.098 rows=1 loops=1)
- 2 Recheck Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
- 3 Heap Blocks: exact=1
- 4 -> Bitmap Index Scan on idx_flights_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.075..0.075 rows=1 loops=1)
- 5 Index Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
- 6 Planning Time: 2.776 ms
- 7 Execution Time: 0.140 ms

At the bottom of the pgAdmin window, a status bar indicates 'Total rows: 7' and 'Query complete 00:00:00.082'. To the right of the status bar, a message box says 'Successfully run. Total query runtime: 82 msec. 7 rows affected.' with a green checkmark icon. The status bar also shows 'CRLF Ln 8, Col 30', '21:52', 'РУС', and '11.11.2023'.

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

pgAdmin 4

File Object Tools Edit View Window Help

Welcome Sai_db/postgres@PostgreSQL 17*

Sai_db/postgres@PostgreSQL 17

Query History

```
1 v SELECT indexname, indexdef
2   FROM pg_indexes
3 WHERE tablename = 'flights';
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

indexname	indexdef
flights_pkey	CREATE UNIQUE INDEX flights_pkey ON public.flights USING btree (flight_id)
idx_flights_no_sched_departure	CREATE UNIQUE INDEX idx_flights_no_sched_departure ON public.flights USING btree (flight_no, scheduled_departure)
idx_flights_departure_arrival	CREATE INDEX idx_flights_departure_arrival ON public.flights USING btree (departure_airport_id, arrival_airport_id)
idx_flights_actual_departure	CREATE INDEX idx_flights_actual_departure ON public.flights USING btree (actual_departure)

Total rows: 4 Query complete 00:00:00.109

Successfully run. Total query runtime: 109 msec. 4 rows affected.

22:01 CRLF Ln 3, Col 29

pgAdmin 4

File Object Tools Edit View Window Help

Welcome Sai_db/postgres@PostgreSQL 17*

Sai_db/postgres@PostgreSQL 17

Query History

```
1 v EXPLAIN ANALYZE
2 SELECT *
3   FROM flights
4 WHERE departure_airport_id = 3
5   AND arrival_airport_id = 7;
```

Data Output Messages Notifications

Showing rows: 1 to 7 Page No: 1 of 1

QUERY PLAN	text
1	Bitmap Heap Scan on flights (cost=4.30..9.97 rows=2 width=61) (actual time=0.040..0.041 rows=1 loops=1)
2	Recheck Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on idx_flights_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.025..0.026 rows=1 loops=1)
5	Index Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
6	Planning Time: 0.200 ms
7	Execution Time: 0.098 ms

Total rows: 7 Query complete 00:00:00.108

Successfully run. Total query runtime: 108 msec. 7 rows affected.

22:05 CRLF Ln 5, Col 30

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.

pgAdmin 4

File Object Tools Edit View Window Help

Welcome Sai_db/postgres@PostgreSQL 17*

Sai_db/postgres@PostgreSQL 17

Query History

```
1 ✓ CREATE UNIQUE INDEX idx_passengers_passport
2   ON passengers (passport_number);
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 116 msec.

Total rows: Query complete 00:00:00.116

Sai_db/postgres@PostgreSQL 17

Query History

```
1 ✓ SELECT indexname, indexdef
2   FROM pg_indexes
3   WHERE tablename = 'passengers';
```

Data Output Messages Notifications

Indexname	indexdef
passenger_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
idx_passengers_passport	CREATE UNIQUE INDEX idx_passengers_passport ON public.passengers USING btree (passport_number)

Showing rows: 1 to 2 Page No: 1 of 1

Total rows: 2 Query complete 00:00:00.130

Sai_db/postgres@PostgreSQL 17

Successfully run. Total query runtime: 130 msec. 2 rows affected.

CRLF Ln 3, Col 32

The image contains three vertically stacked screenshots of the pgAdmin 4 interface, version 4.17. Each screenshot shows a query being run against a PostgreSQL database.

Screenshot 1:

```

INSERT INTO passengers (first_name, last_name, passport_number)
VALUES ('Adil', 'Kassym', 'KZ9988776');

```

Data Output:

INSERT 0 1
Query returned successfully in 73 msec.

Screenshot 2:

```

INSERT INTO passengers (first_name, last_name, passport_number)
VALUES ('Aruzhan', 'Tursyn', 'KZ9988776');

```

Data Output:

Total rows: 1 Query complete 00:00:00.073
ERROR: повторяющееся значение ключа нарушает ограничение уникальности "idx_passengers_passport"
Ключ "(passport_number)=(KZ9988776)" уже существует.
ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "idx_passengers_passport"
SQL state: 23505
Detail: Ключ "(passport_number)=(KZ9988776)" уже существует.

Screenshot 3:

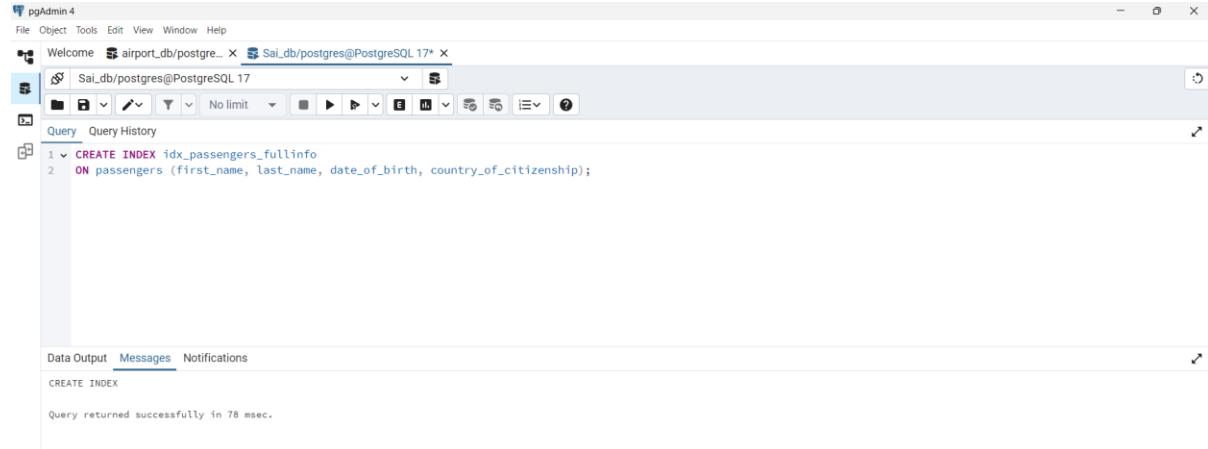
Data Output:

Total rows: 1 Query complete 00:00:00.079
ERROR: повторяющееся значение ключа нарушает ограничение уникальности "idx_passengers_passport"
Ключ "(passport_number)=(KZ9988776)" уже существует.
ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "idx_passengers_passport"
SQL state: 23505
Detail: Ключ "(passport_number)=(KZ9988776)" уже существует.

Explain in your own words what is going on in the output?

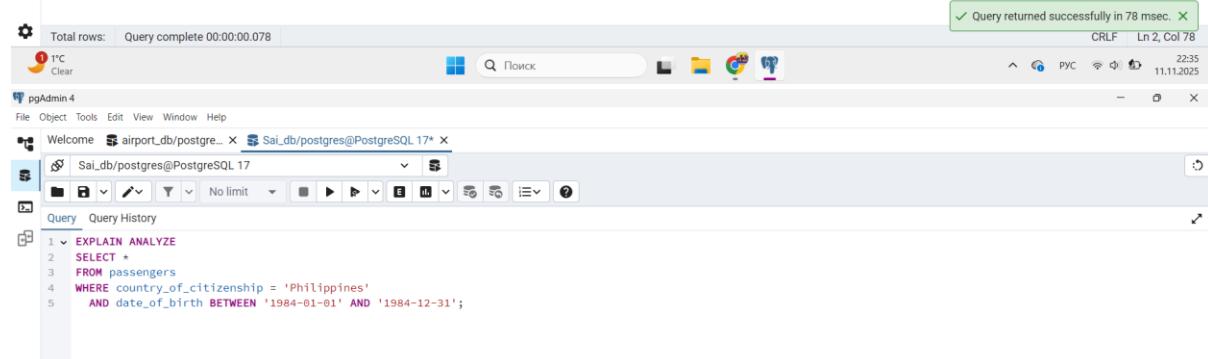
When we inserted the first passenger, the database accepted it — because the passport number was new. When we tried to insert the second passenger with the same passport number, PostgreSQL showed an error. This happened because we created a **unique index** on the `passport_number` column. It means every passenger must have a **different** passport number. So the database **stopped** the second insert to keep the data correct and avoid duplicates.

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.



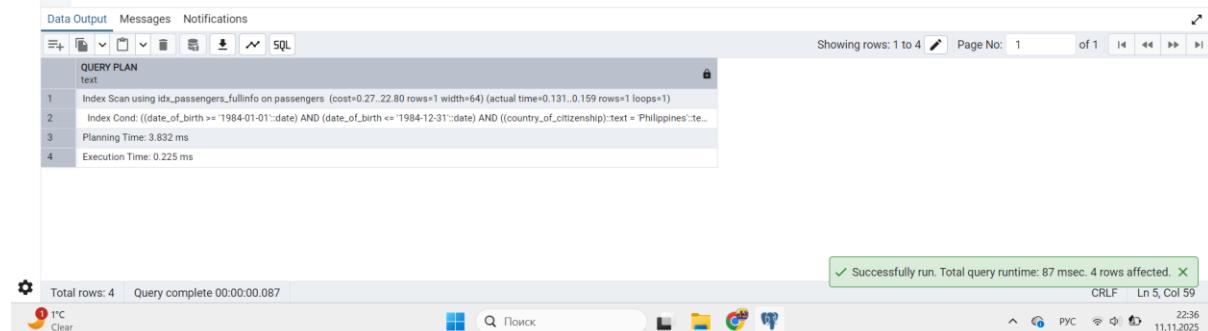
```
CREATE INDEX idx_passengers_fullinfo
ON passengers (first_name, last_name, date_of_birth, country_of_citizenship);
```

Query returned successfully in 78 msec.



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

Showing rows: 1 to 4 Page No: 1 of 1 14 44 >> >>



Successfully run. Total query runtime: 87 msec. 4 rows affected.

PostgreSQL used the composite index when we searched for a passenger by birth year and citizenship. The query was much faster because the database did not scan the whole table – it used the index to find only matching rows. If there is no index or the query does not match its columns, PostgreSQL does a sequential scan and becomes slower.

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

pgAdmin 4

File Object Tools Edit View Window Help

Welcome Sai_db/postgres@PostgreSQL 17*

Sai_db/postgres@PostgreSQL 17

Query History

```
1 SELECT indexname, indexdef
2 FROM pg_indexes
3 WHERE tablename = 'passengers';
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1 of 1

indexname	indexdef
passenger_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
idx_passengers_pass...	CREATE UNIQUE INDEX idx_passengers_passport ON public.passengers USING btree (passport_number)
idx_passengers_fullinfo	CREATE INDEX idx_passengers_fullinfo ON public.passengers USING btree (first_name, last_name, date_of_birth, country_of_citizenship)

Total rows: 3 Query complete 00:00:00.155

Сucceeded Successfully run. Total query runtime: 155 msec. 3 rows affected.

CRLF Ln 3, Col 32

22:41 11.11.2025

pgAdmin 4

File Object Tools Edit View Window Help

Welcome Sai_db/postgres@PostgreSQL 17*

Sai_db/postgres@PostgreSQL 17

Query History

```
1 DROP INDEX idx_passengers_passport;
2 DROP INDEX idx_passengers_fullinfo;
```

Data Output Messages Notifications

DROP INDEX

Query returned successfully in 116 msec.

Total rows: 0 Query complete 00:00:00.116

Сucceeded Query returned successfully in 116 msec.

CRLF Ln 2, Col 36

22:42 11.11.2025

pgAdmin 4

File Object Tools Edit View Window Help

Welcome Sai_db/postgres@PostgreSQL 17*

Sai_db/postgres@PostgreSQL 17