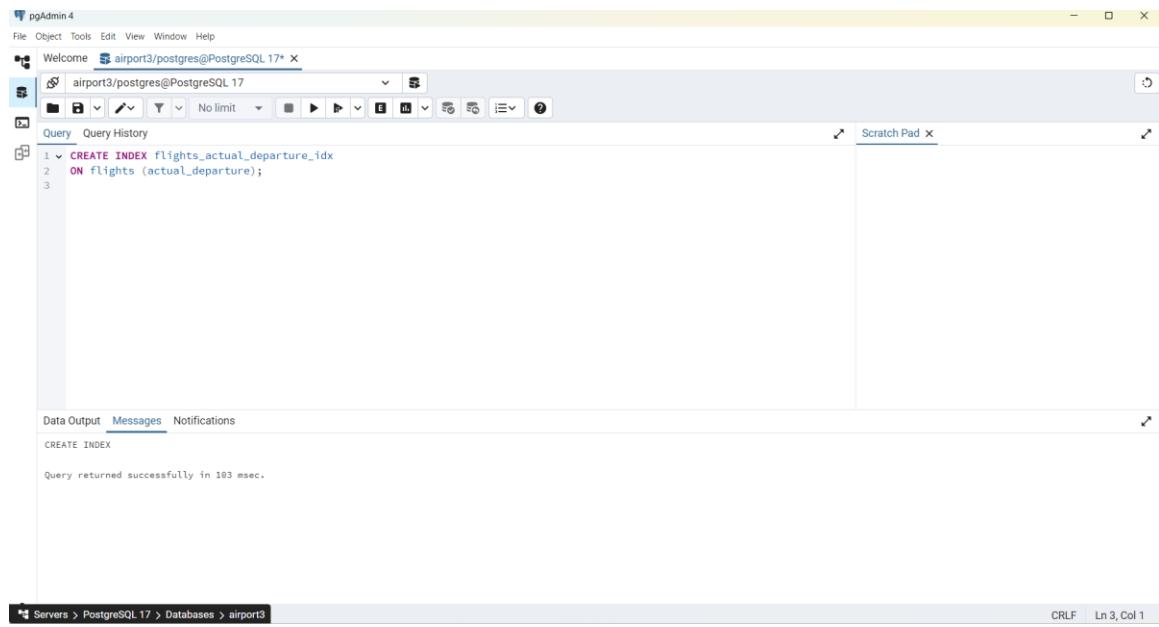


## Laboratory Work 7

### 1. Create an index on actual\_departure in flights



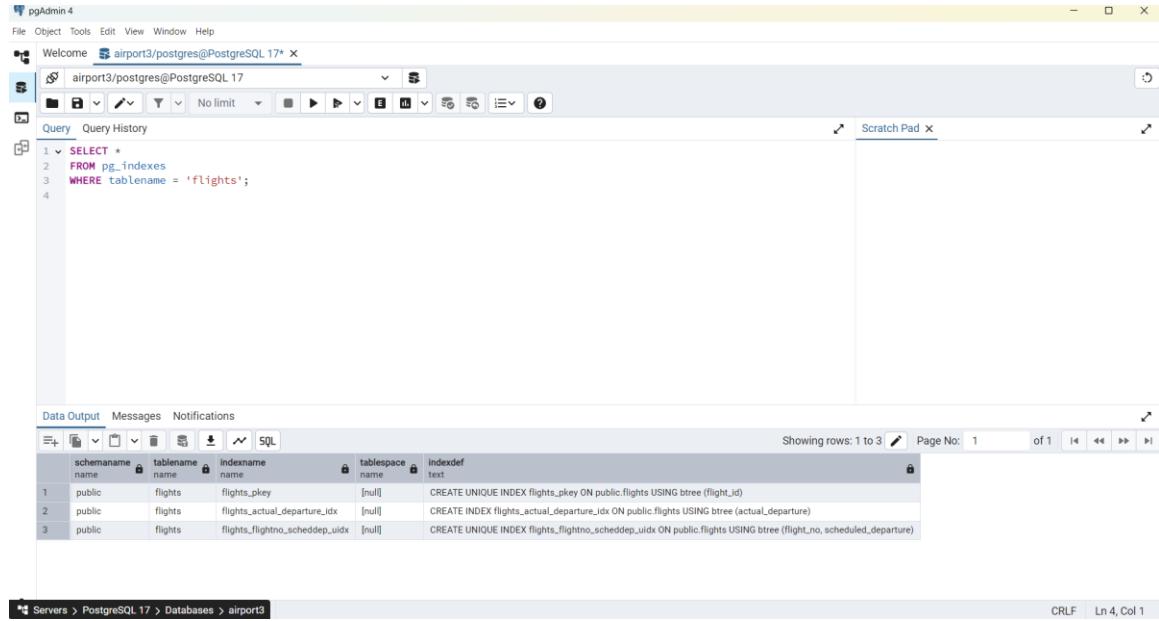
The screenshot shows the pgAdmin 4 interface with a query editor window. The query is:

```
1 v CREATE INDEX flights_actual_departure_idx
2 ON flights (actual_departure);
3
```

The results pane shows the output of the query:

```
CREATE INDEX
Query returned successfully in 103 msec.
```

### 2. Create unique index flight\_no+scheduled\_departure



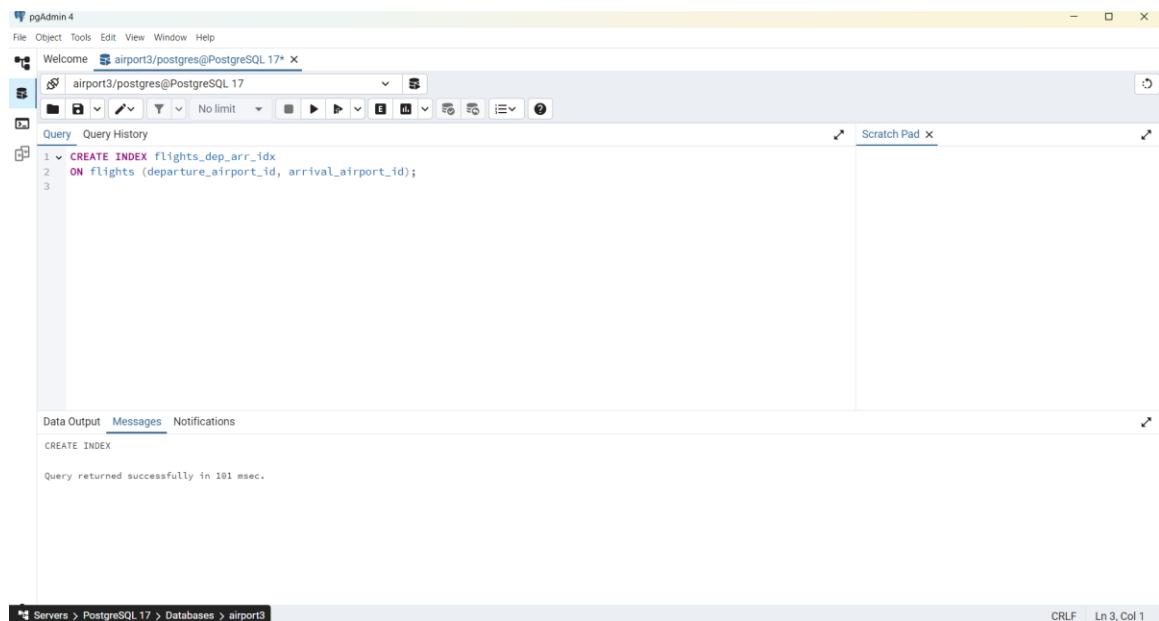
The screenshot shows the pgAdmin 4 interface with a query editor window. The query is:

```
1 v SELECT *
2 FROM pg_indexes
3 WHERE tablename = 'flights';
4
```

The results pane shows the output of the query:

schemaname	tablename	indexname	tablespace	Indexdef
1 public	flights	flights_pkey	[null]	CREATE UNIQUE INDEX flights_pkey ON public.flights USING btree (flight_id)
2 public	flights	flights_actual_departure_idx	[null]	CREATE INDEX flights_actual_departure_idx ON public.flights USING btree (actual_departure)
3 public	flights	flights_flightno_scheddep_idx	[null]	CREATE UNIQUE INDEX flights_flightno_scheddep_idx ON public.flights USING btree (flight_no, scheduled_departure)

### 3. Create composite index departure\_airport\_id + arrival\_airport\_id



The screenshot shows the pgAdmin 4 interface with a single query window open. The query is:

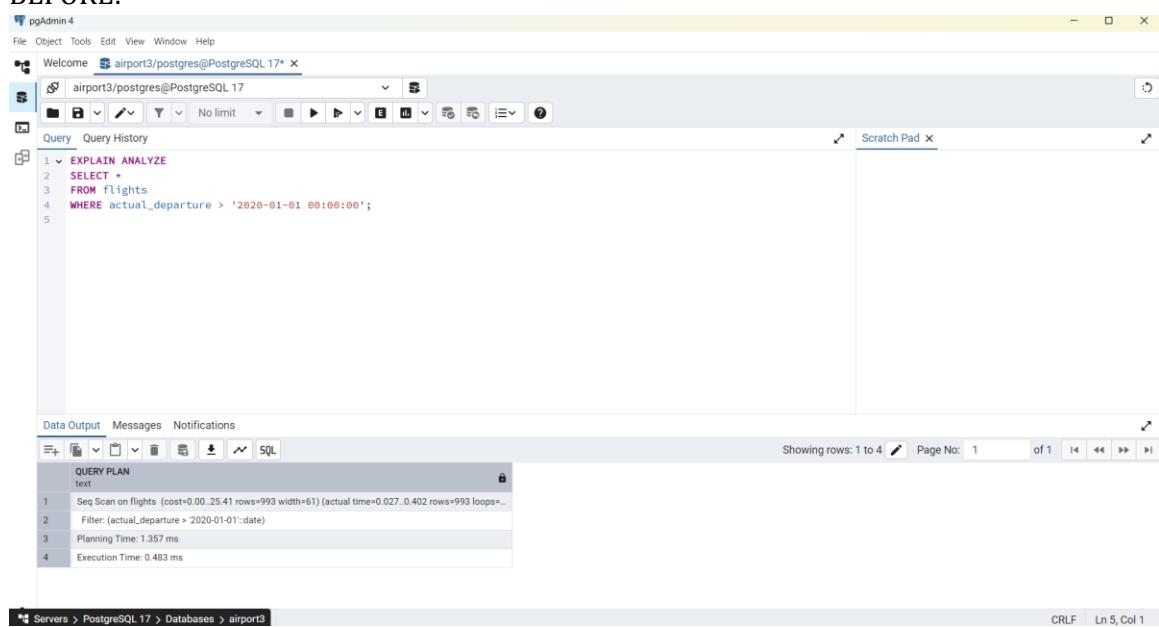
```
1 ✓ CREATE INDEX flights_dep_arr_idx
2   ON flights (departure_airport_id, arrival_airport_id);
3
```

The 'Messages' tab at the bottom shows the result:

```
CREATE INDEX
Query returned successfully in 101 msec.
```

### 4. Evaluate performance before/after indexes

BEFORE:



The screenshot shows the pgAdmin 4 interface with a single query window open. The query is:

```
1 ✓ EXPLAIN ANALYZE
2   SELECT *
3     FROM flights
4    WHERE actual_departure > '2020-01-01 00:00:00';
5
```

The 'Data Output' tab at the bottom displays the query plan:

QUERY PLAN	text
1	Seq Scan on flights (cost=0.00..25.41 rows=993 width=61) (actual time=0.027..0.402 rows=993 loops=1)
2	Filter: (actual_departure > 2020-01-01::date)
3	Planning Time: 1.357 ms
4	Execution Time: 0.483 ms

AFTER:

The screenshot shows the pgAdmin 4 interface with a query window containing the following SQL code:

```
EXPLAIN ANALYZE
SELECT *
FROM flights
WHERE actual_departure > '2020-01-01 00:00:00';
```

Below the query window is a "Data Output" tab showing the execution plan:

QUERY PLAN	text
1	Seq Scan on flights (cost=0.00..25.41 rows=993 width=61) (actual time=0.049..0.682 rows=993 loops=1)
2	Filter: (actual_departure > 2020-01-01::date)
3	Planning Time: 4.116 ms
4	Execution Time: 0.823 ms

The status bar at the bottom indicates "CRLF Ln 5, Col 1".

## 5. Use EXPLAIN ANALYZE for airport filters

The screenshot shows the pgAdmin 4 interface with a query window containing the following SQL code:

```
EXPLAIN ANALYZE
SELECT *
FROM flights
WHERE departure_airport_id = 30
AND arrival_airport_id = 86;
```

Below the query window is a "Data Output" tab showing the execution plan:

QUERY PLAN	text
1	Index Scan using flights_dep_arr_idx on flights (cost=0.28..8.29 rows=1 width=61) (actual time=0.157..0.158 rows=0 loops=1)
2	Index Cond: (departure_airport_id = 30) AND (arrival_airport_id = 86)
3	Planning Time: 2.457 ms
4	Execution Time: 0.239 ms

The status bar at the bottom indicates "CRLF Ln 6, Col 1".

## 6. Unique index on passport\_number + test inserts

The screenshot shows the pgAdmin 4 interface. The top menu bar includes File, Object, Tools, Edit, View, Window, and Help. The title bar says "pgAdmin 4" and "Welcome airport3/postgres@PostgreSQL 17\*". The main window has a toolbar at the top with various icons. Below the toolbar is a "Query" tab and a "Scratch Pad" tab. The "Query" tab contains the following SQL code:

```
1 v CREATE UNIQUE INDEX passengers_passport_uidx
2 ON passengers (passport_number);
```

Below the query editor is a "Data Output" tab. It displays the message "CREATE INDEX" followed by "Query returned successfully in 103 msec.". At the bottom of the pgAdmin window, there is a navigation bar with "Servers > PostgreSQL 17 > Databases > airports" and status information "CRLF Ln 3, Col 1".

pgAdmin 4

Welcome airport3/postgres@PostgreSQL 17\*

airport3/postgres@PostgreSQL 17

Query History

```
1 v INSERT INTO passengers (passenger_id, first_name, last_name, date_of_birth, passport_number)
2 VALUES (201, 'Adam', 'Smith', '1984-05-02', 'PP123456');
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 166 msec.

Servers > PostgreSQL 17 > Databases > airport3

CRLF Ln 2, Col 11

Welcome airport3/postgres@PostgreSQL 17\*

airport3/postgres@PostgreSQL 17

Query History

```
1 v INSERT INTO passengers (passenger_id, first_name, last_name, date_of_birth, passport_number)
2 VALUES (202, 'David', 'Jones', '1985-08-11', 'PP123456');
```

Data Output Messages Notifications

ERROR: повторяющееся значение ключа нарушает ограничение уникальности "passengers\_passport\_uidx"  
Ключ "(passport\_number)=(PP123456)" уже существует.

ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "passengers\_passport\_uidx"  
SQL state: 23505  
Detail: Ключ "(passport\_number)=(PP123456)" уже существует.

Servers > PostgreSQL 17 > Databases > airport3

CRLF Ln 2, Col 10

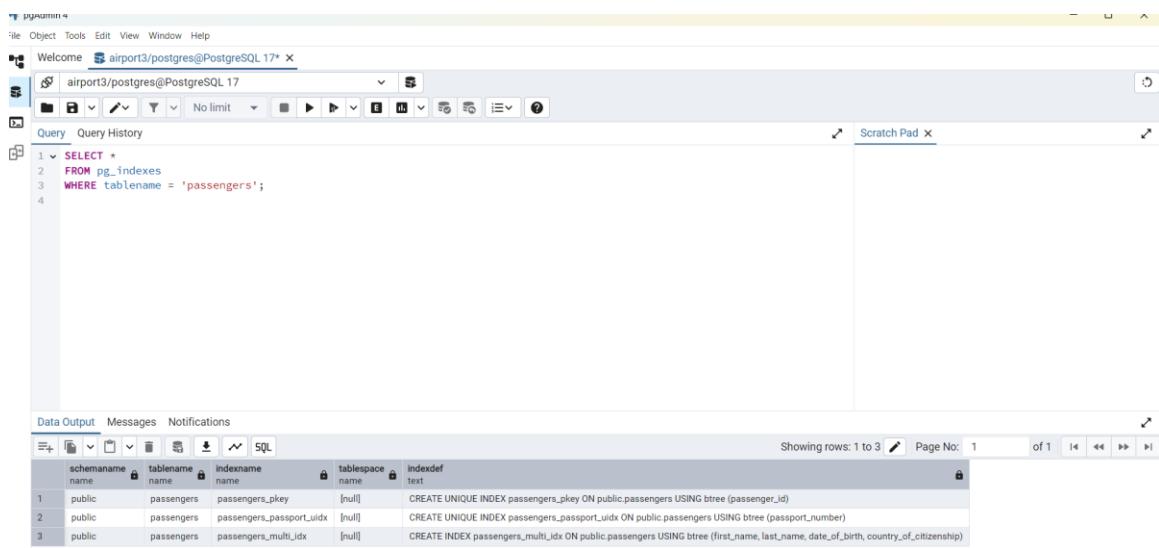
## 7. Composite index on passenger fields + EXPLAIN ANALYZE

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

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

PostgreSQL does not use the composite index because the query filters by the last columns of the index, not by the first ones, so it performs a sequential scan instead.

## 8. List indexes & then delete them

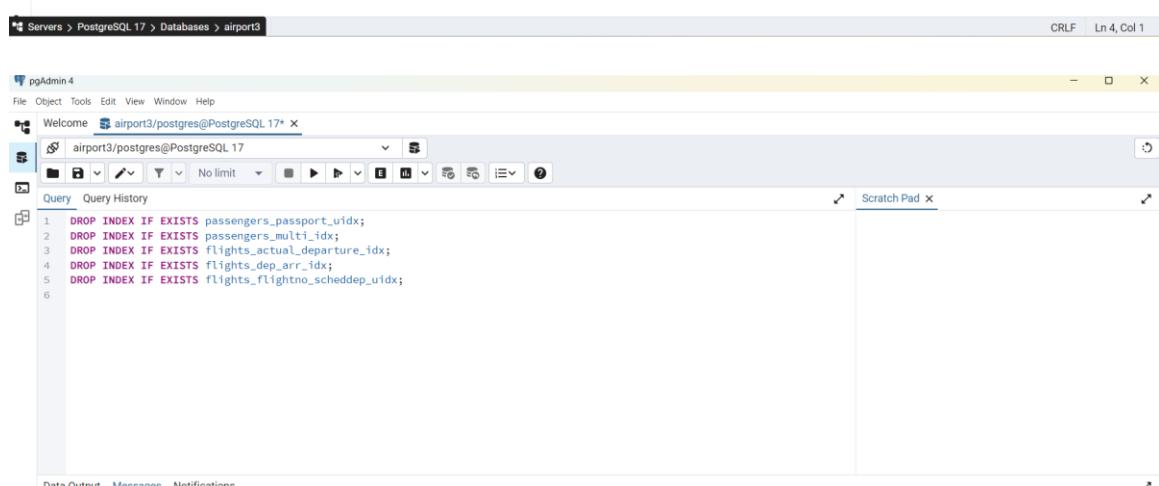


The screenshot shows the pgAdmin 4 interface with the 'Indexes' tab selected for the 'passenger' table. The query window contains the following SQL:

```
1 SELECT *  
2 FROM pg_indexes  
3 WHERE tablename = 'passenger';
```

The results table shows three indexes:

schemaname	tablename	indexname	tablespace	indexdef
1 public	passenger	passenger_pkey	[null]	CREATE UNIQUE INDEX passenger_pkey ON public.passenger USING btree (passenger_id)
2 public	passenger	passenger_passport_uidx	[null]	CREATE UNIQUE INDEX passenger_passport_uidx ON public.passenger USING btree (passport_number)
3 public	passenger	passenger_multi_idx	[null]	CREATE INDEX passenger_multi_idx ON public.passenger USING btree (first_name, last_name, date_of_birth, country_of_citizenship)

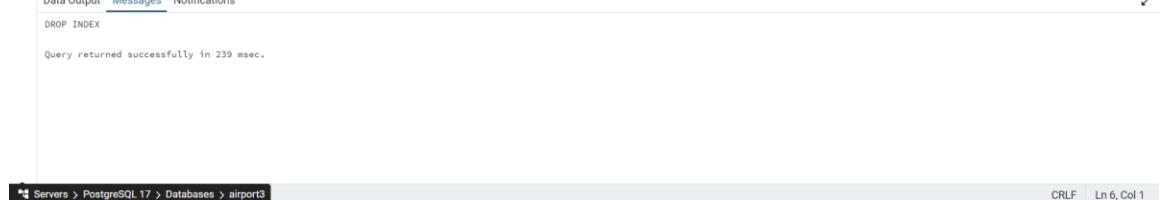
  


The screenshot shows the pgAdmin 4 interface with the 'Indexes' tab selected for the 'flights' table. The query window contains the following SQL:

```
1 DROP INDEX IF EXISTS passengers_passport_uidx;  
2 DROP INDEX IF EXISTS passengers_multi_idx;  
3 DROP INDEX IF EXISTS flights_actual_departure_idx;  
4 DROP INDEX IF EXISTS flights_dep_arr_idx;  
5 DROP INDEX IF EXISTS flights_flightno_scheddep_uidx;
```

The results table shows five indexes:

schemaname	tablename	indexname	tablespace	indexdef
1 public	flights	flights_actual_departure_idx	[null]	CREATE INDEX flights_actual_departure_idx ON public.flights (actual_departure)
2 public	flights	flights_dep_arr_idx	[null]	CREATE INDEX flights_dep_arr_idx ON public.flights (dep_airport, arr_airport)
3 public	flights	flights_flightno_scheddep_uidx	[null]	CREATE INDEX flights_flightno_scheddep_uidx ON public.flights (flightno, scheddep)
4 public	flights	flights_pkey	[null]	CREATE PRIMARY INDEX flights_pkey ON public.flights (flightno) USING btree
5 public	flights	flights_passenger_uidx	[null]	CREATE INDEX flights_passenger_uidx ON public.flights (passenger_id)

The screenshot shows the pgAdmin 4 interface with the 'Messages' tab selected. The query window contains the following message:

```
DROP INDEX
```

The status bar indicates "Query returned successfully in 239 msec."