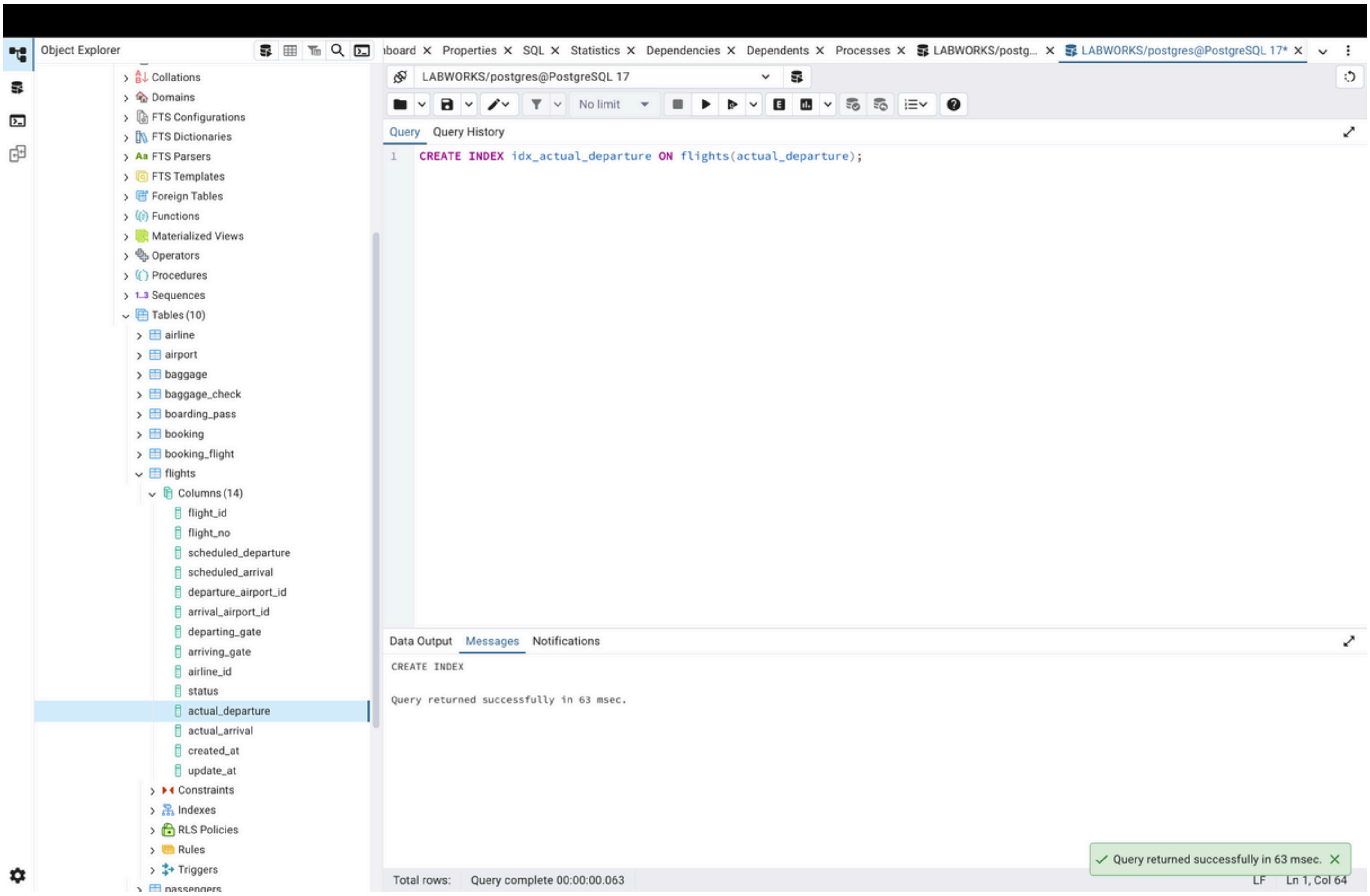
Laboratory work 7

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



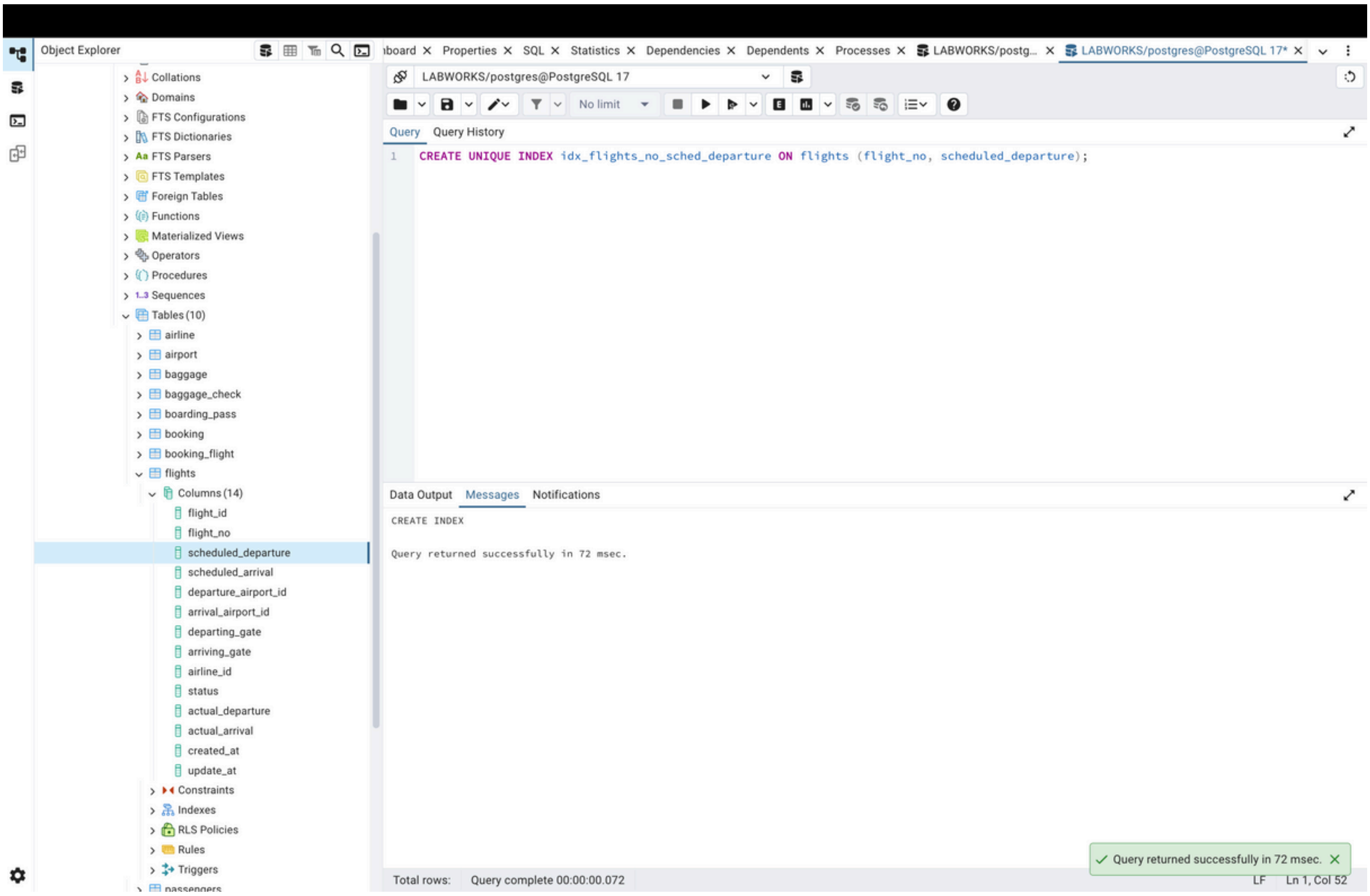
```
CREATE INDEX idx_actual_departure ON flights(actual_departure);
```

```
CREATE INDEX

Query returned successfully in 63 msec.
```

Total rows: Query complete 00:00:00.063

Query returned successfully in 63 msec.

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



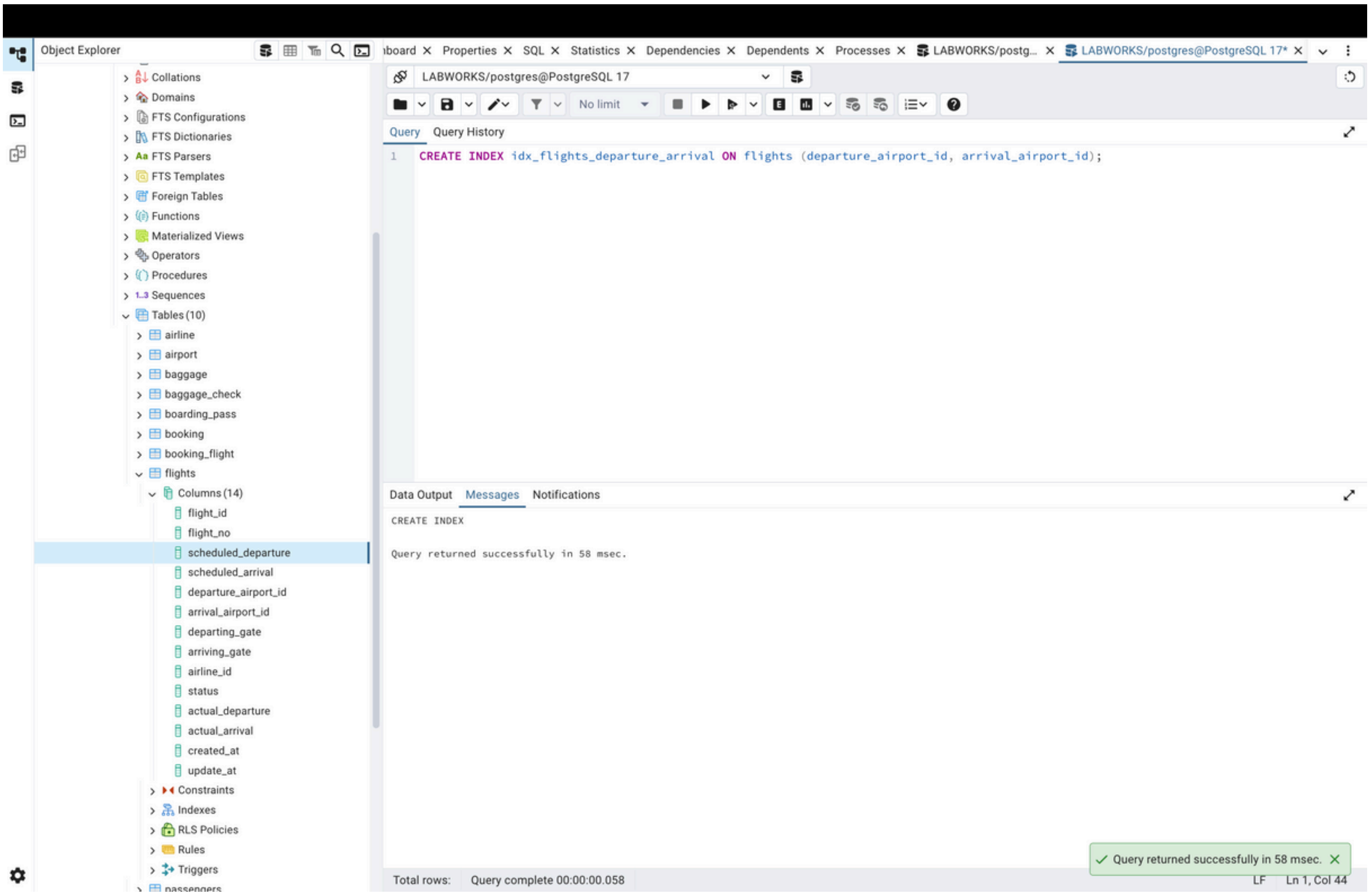
```
CREATE UNIQUE INDEX idx_flights_no_sched_departure ON flights (flight_no, scheduled_departure);
```

```
CREATE INDEX

Query returned successfully in 72 msec.
```

✓ Query returned successfully in 72 msec.

Total rows: Query complete 00:00:00.072 LF Ln 1, Col 52

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



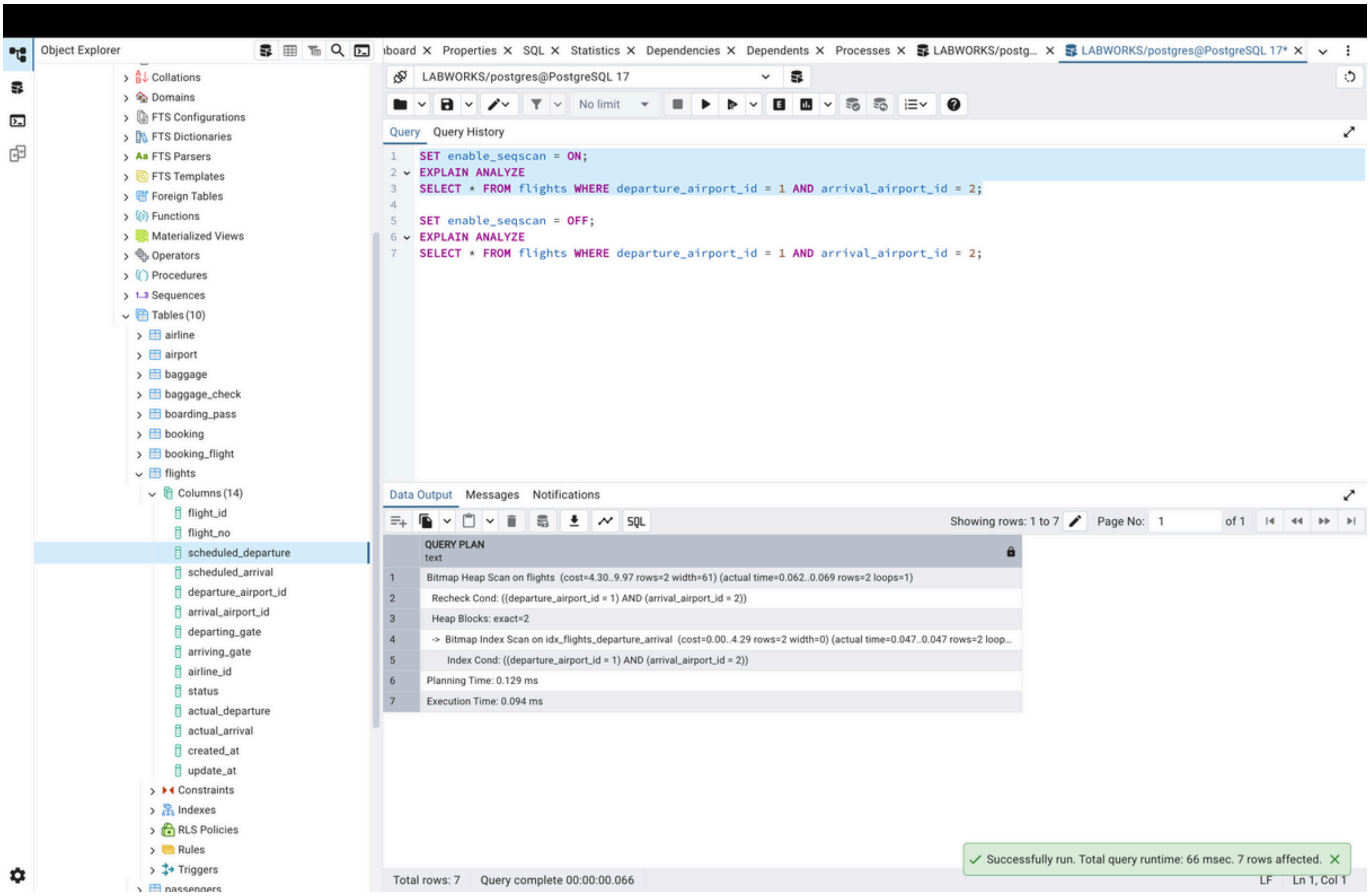
```
CREATE INDEX idx_flights_departure_arrival ON flights (departure_airport_id, arrival_airport_id);
```

```
CREATE INDEX

Query returned successfully in 58 msec.
```

✓ Query returned successfully in 58 msec.

Total rows: Query complete 00:00:00.058 LF Ln 1, Col 44

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



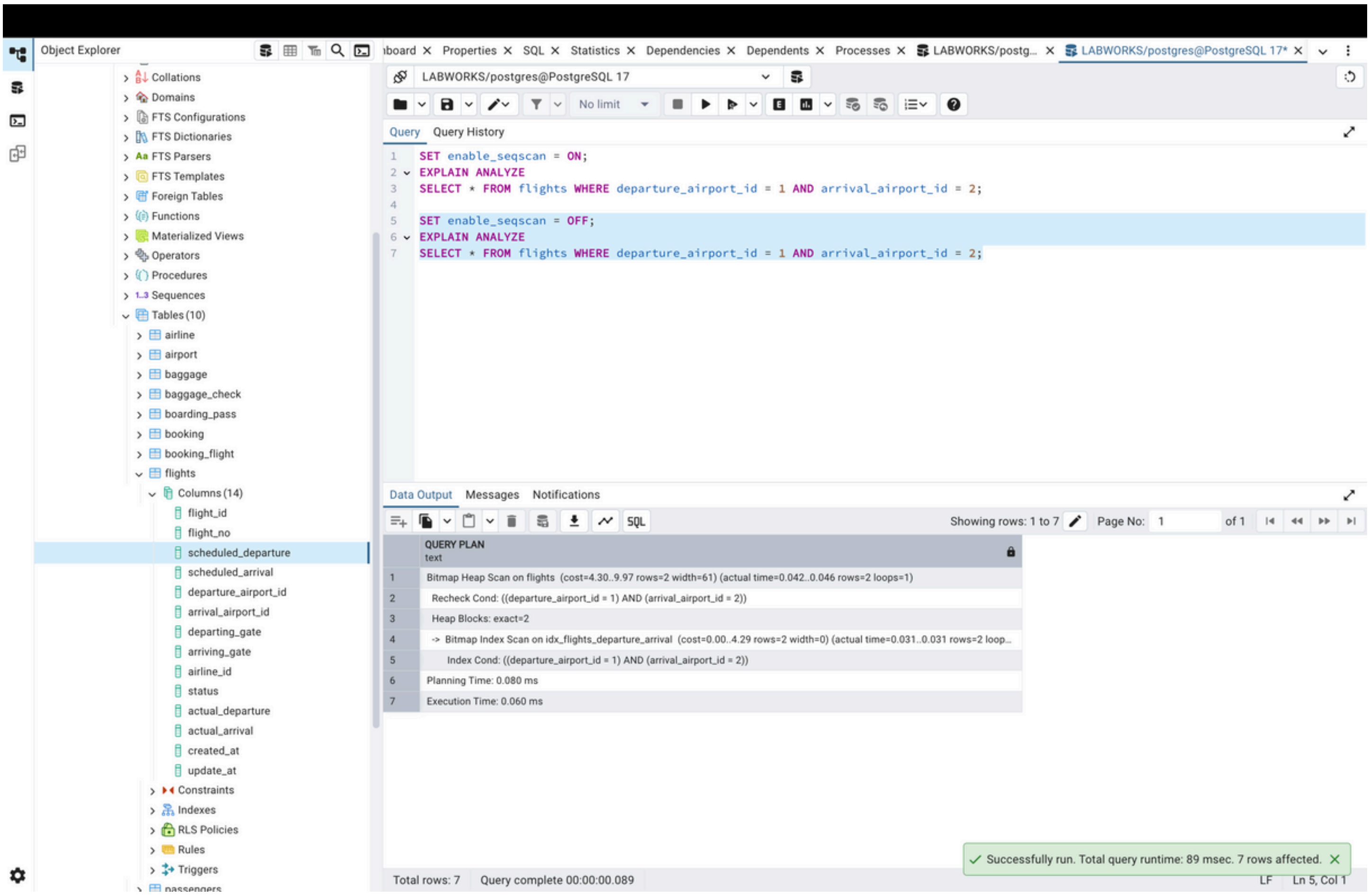
```sql
SET enable_seqscan = ON;
EXPLAIN ANALYZE
SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;

SET enable_seqscan = OFF;
EXPLAIN ANALYZE
SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;
```

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=0.062..0.069 rows=2 loops=1)
2	Recheck Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
3	Heap Blocks: exact=2
4	-> Bitmap Index Scan on idx_flights_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.047..0.047 rows=2 loop...
5	Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
6	Planning Time: 0.129 ms
7	Execution Time: 0.094 ms

✓ Successfully run. Total query runtime: 66 msec. 7 rows affected.

Total rows: 7 Query complete 00:00:00.066 LF Ln 1, Col 1



```sql
SET enable_seqscan = ON;
EXPLAIN ANALYZE
SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;

SET enable_seqscan = OFF;
EXPLAIN ANALYZE
SELECT * FROM flights WHERE departure_airport_id = 1 AND arrival_airport_id = 2;
```

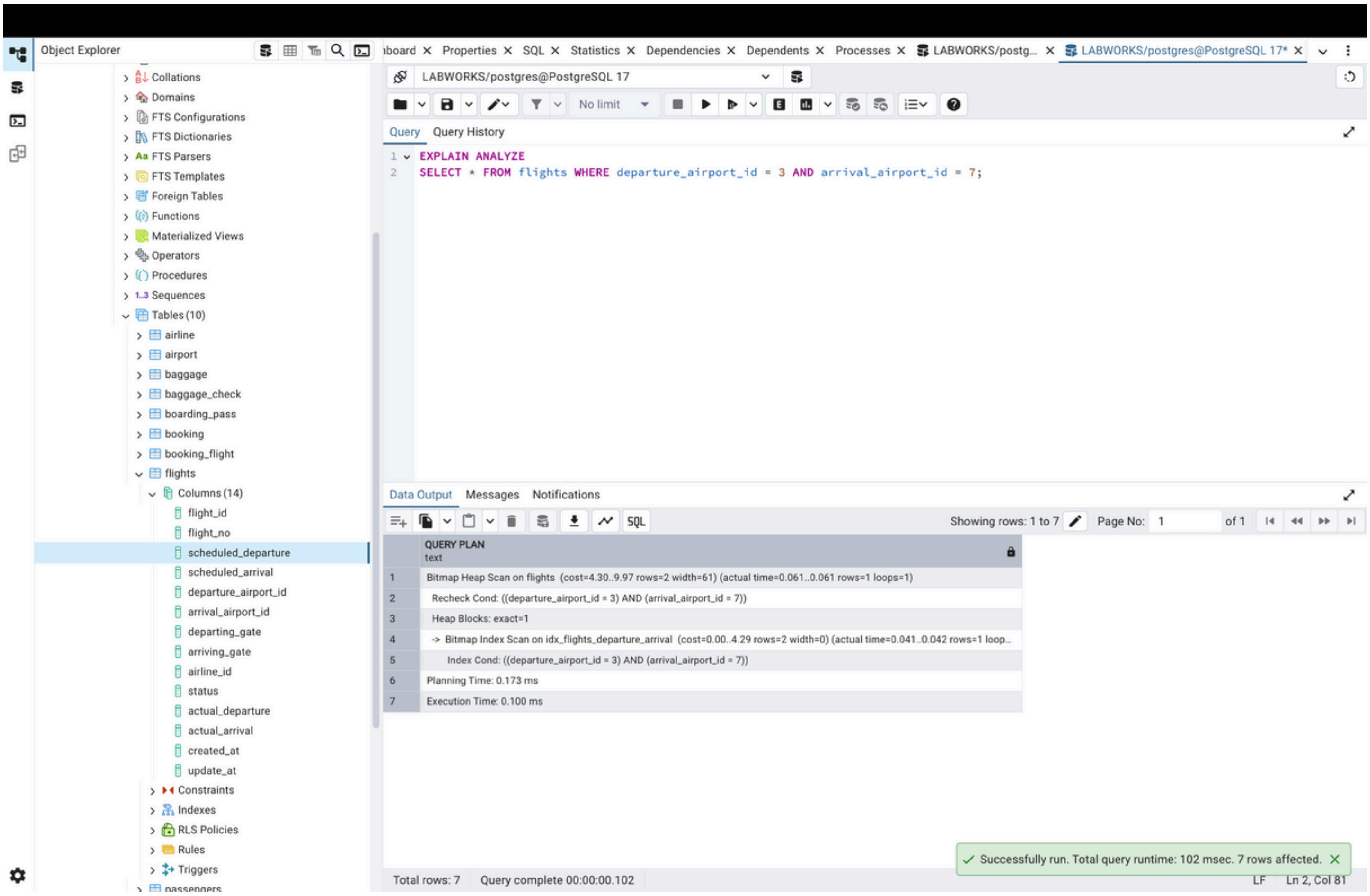
Showing rows: 1 to 7 Page No: 1 of 1

	QUERY PLAN text
1	Bitmap Heap Scan on flights (cost=4.30..9.97 rows=2 width=61) (actual time=0.042..0.046 rows=2 loops=1)
2	Recheck Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
3	Heap Blocks: exact=2
4	-> Bitmap Index Scan on idx_flights_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.031..0.031 rows=2 loop...
5	Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
6	Planning Time: 0.080 ms
7	Execution Time: 0.060 ms

✓ Successfully run. Total query runtime: 89 msec. 7 rows affected.

Total rows: 7 Query complete 00:00:00.089 LF Ln 5, Col 1

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 = 3 AND arrival_airport_id = 7;
```

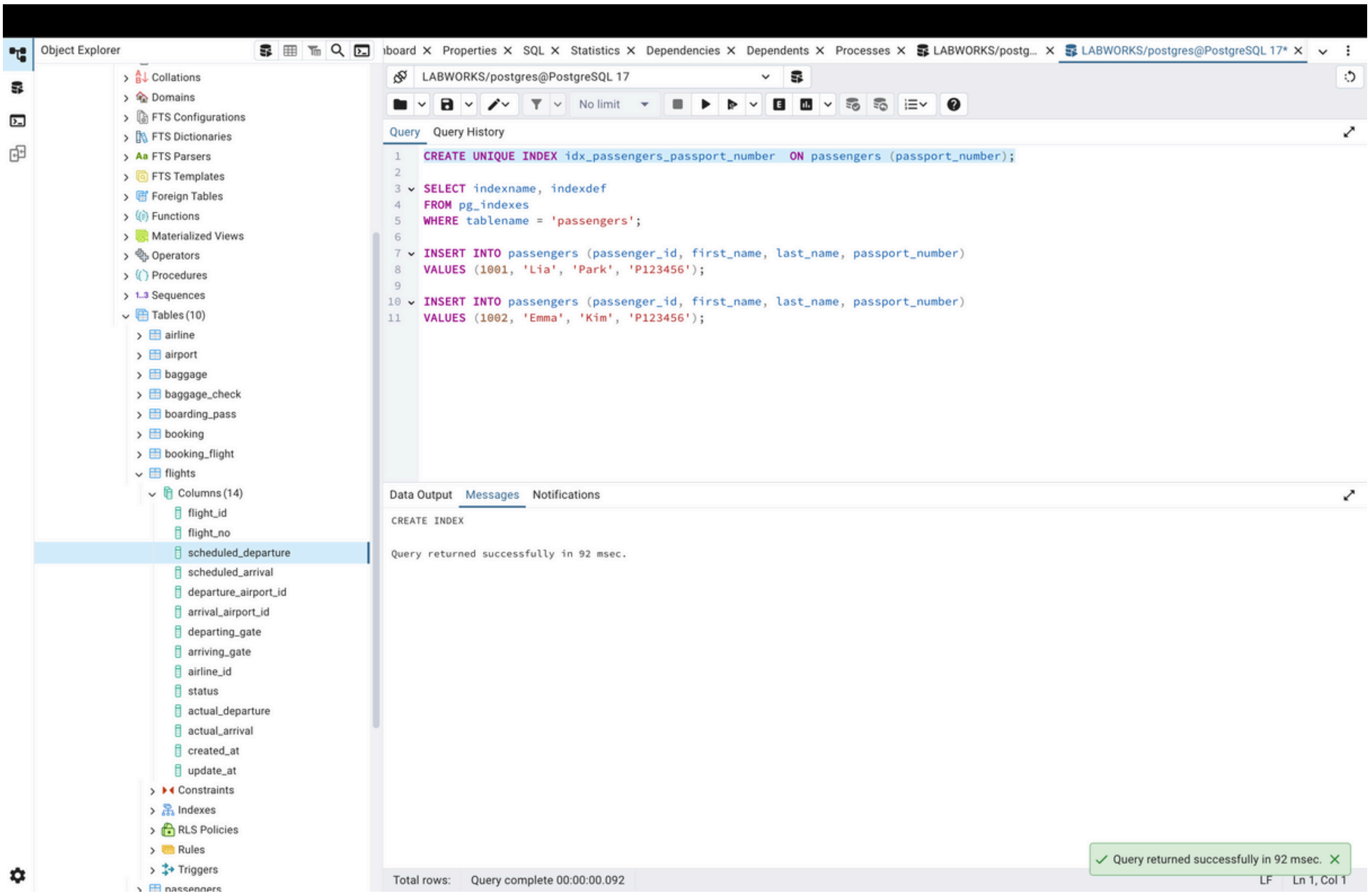
	QUERY PLAN text
1	Bitmap Heap Scan on flights (cost=4.30..9.97 rows=2 width=61) (actual time=0.061..0.061 rows=1 loops=1)
2	Recheck Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on idx_flights_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.041..0.042 rows=1 loop...
5	Index Cond: ((departure_airport_id = 3) AND (arrival_airport_id = 7))
6	Planning Time: 0.173 ms
7	Execution Time: 0.100 ms

✓ Successfully run. Total query runtime: 102 msec. 7 rows affected.

Total rows: 7 Query complete 00:00:00.102

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?

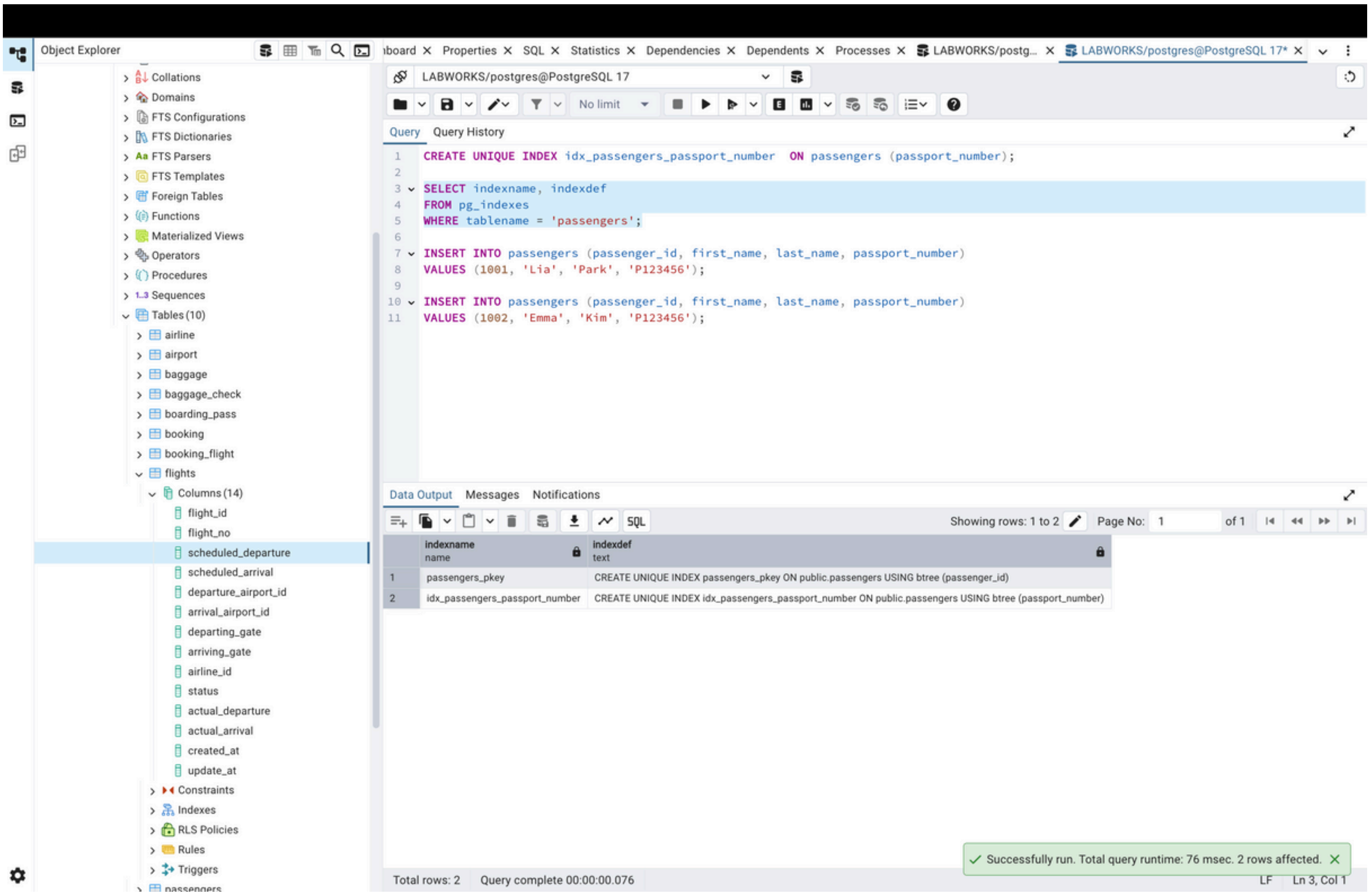


```sql
CREATE UNIQUE INDEX idx_passengers_passport_number  ON passengers (passport_number);

SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passengers';

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');
```

```
CREATE INDEX

Query returned successfully in 92 msec.
```

✓ Query returned successfully in 92 msec. ✕

Total rows: Query complete 00:00:00.092 LF Ln 1, Col 1



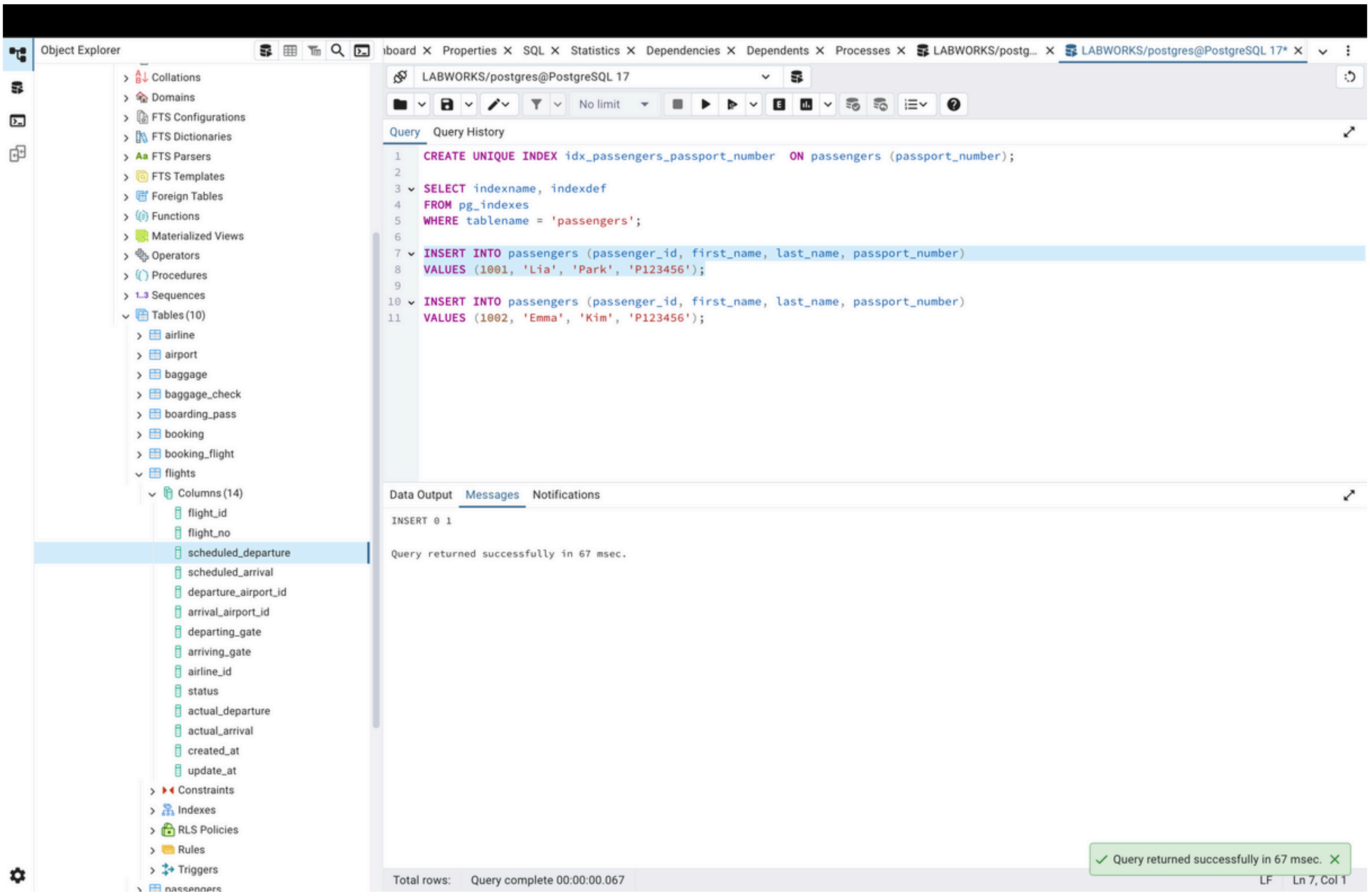
```sql
CREATE UNIQUE INDEX idx_passengers_passport_number  ON passengers (passport_number);

SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passengers';

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');
```

Showing rows: 1 to 2

	indexname name	indexdef text
1	passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
2	idx_passengers_passport_number	CREATE UNIQUE INDEX idx_passengers_passport_number ON public.passengers USING btree (passport_number)

✓ Successfully run. Total query runtime: 76 msec. 2 rows affected. ✕

Total rows: 2 Query complete 00:00:00.076 LF Ln 3, Col 1

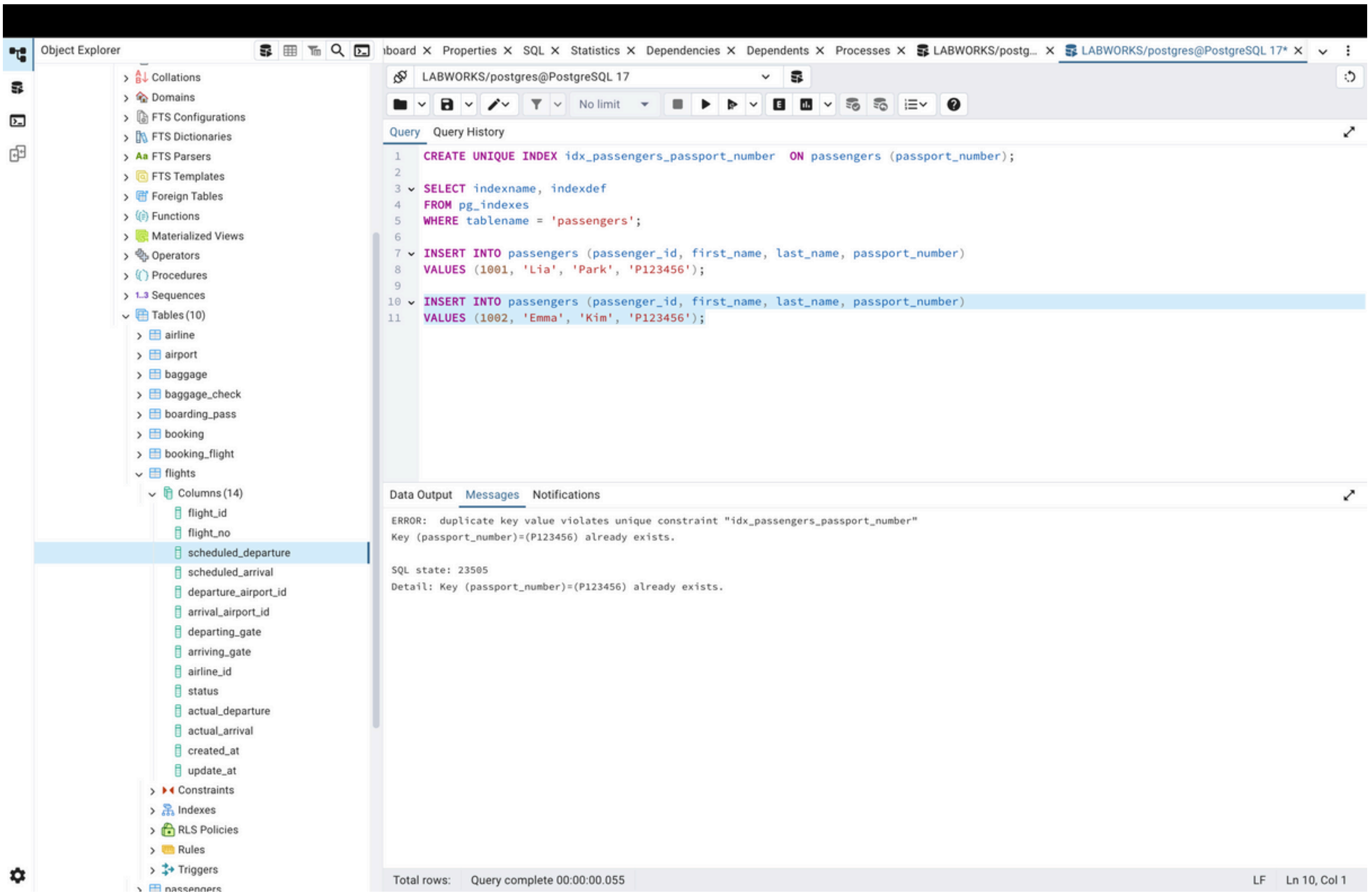


```sql
CREATE UNIQUE INDEX idx_passengers_passport_number  ON passengers (passport_number);

SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passengers';

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');
```

```
INSERT 0 1

Query returned successfully in 67 msec.
```

✓ Query returned successfully in 67 msec.

Total rows: Query complete 00:00:00.067 LF Ln 7, Col 1



```sql
CREATE UNIQUE INDEX idx_passengers_passport_number  ON passengers (passport_number);

SELECT indexname, indexdef
FROM pg_indexes
WHERE tablename = 'passengers';

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1001, 'Lia', 'Park', 'P123456');

INSERT INTO passengers (passenger_id, first_name, last_name, passport_number)
VALUES (1002, 'Emma', 'Kim', 'P123456');
```

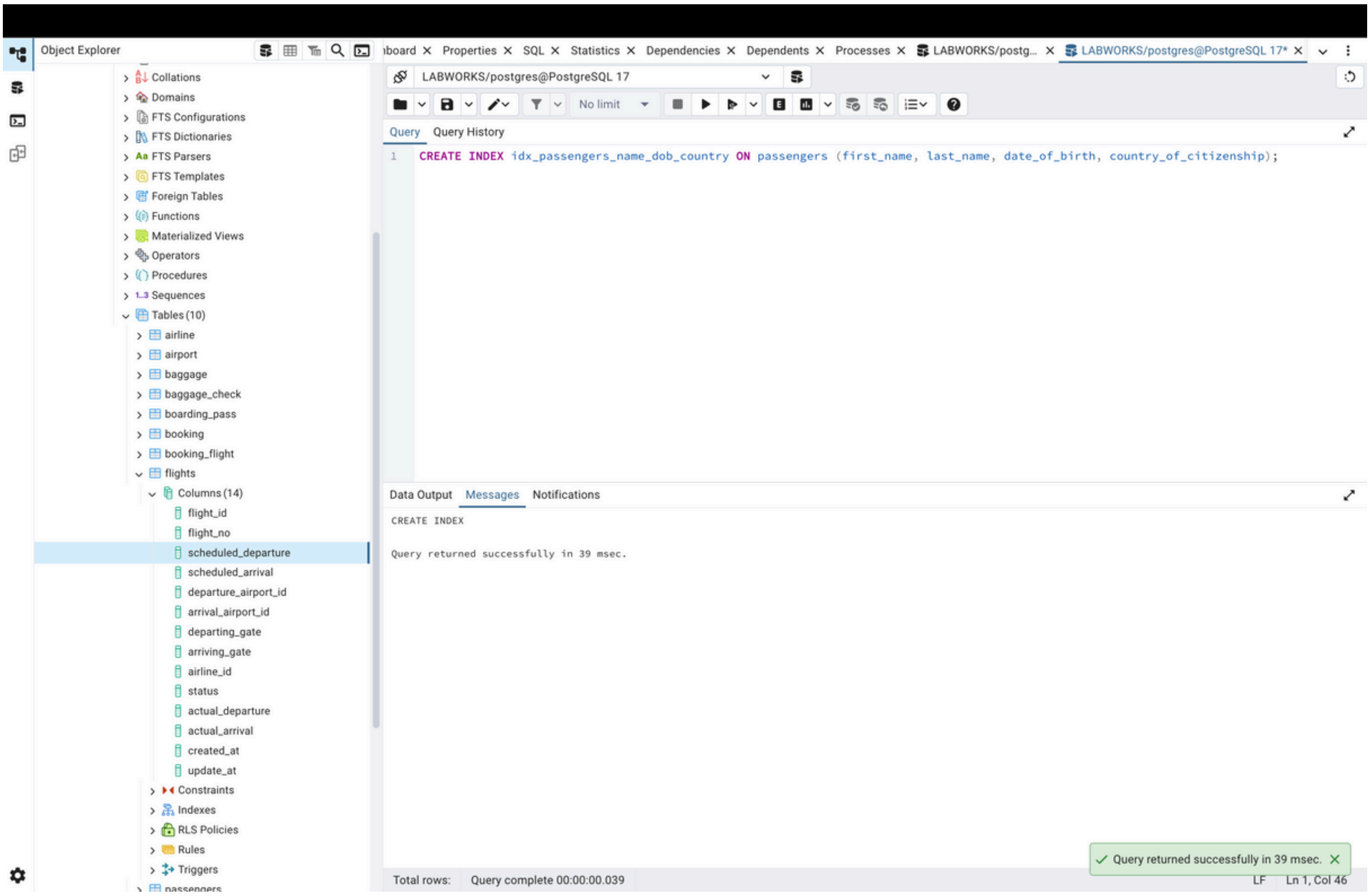
```
ERROR:  duplicate key value violates unique constraint "idx_passengers_passport_number"
Key (passport_number)=(P123456) already exists.

SQL state: 23505
Detail: Key (passport_number)=(P123456) already exists.
```

Total rows: Query complete 00:00:00.055 LF Ln 10, Col 1

The error means you tried to insert a new passenger with the same passport number as one that already exists.

Because the column has a unique index, PostgreSQL doesn’t allow duplicate passport numbers, so it stops the insertion and shows this error.

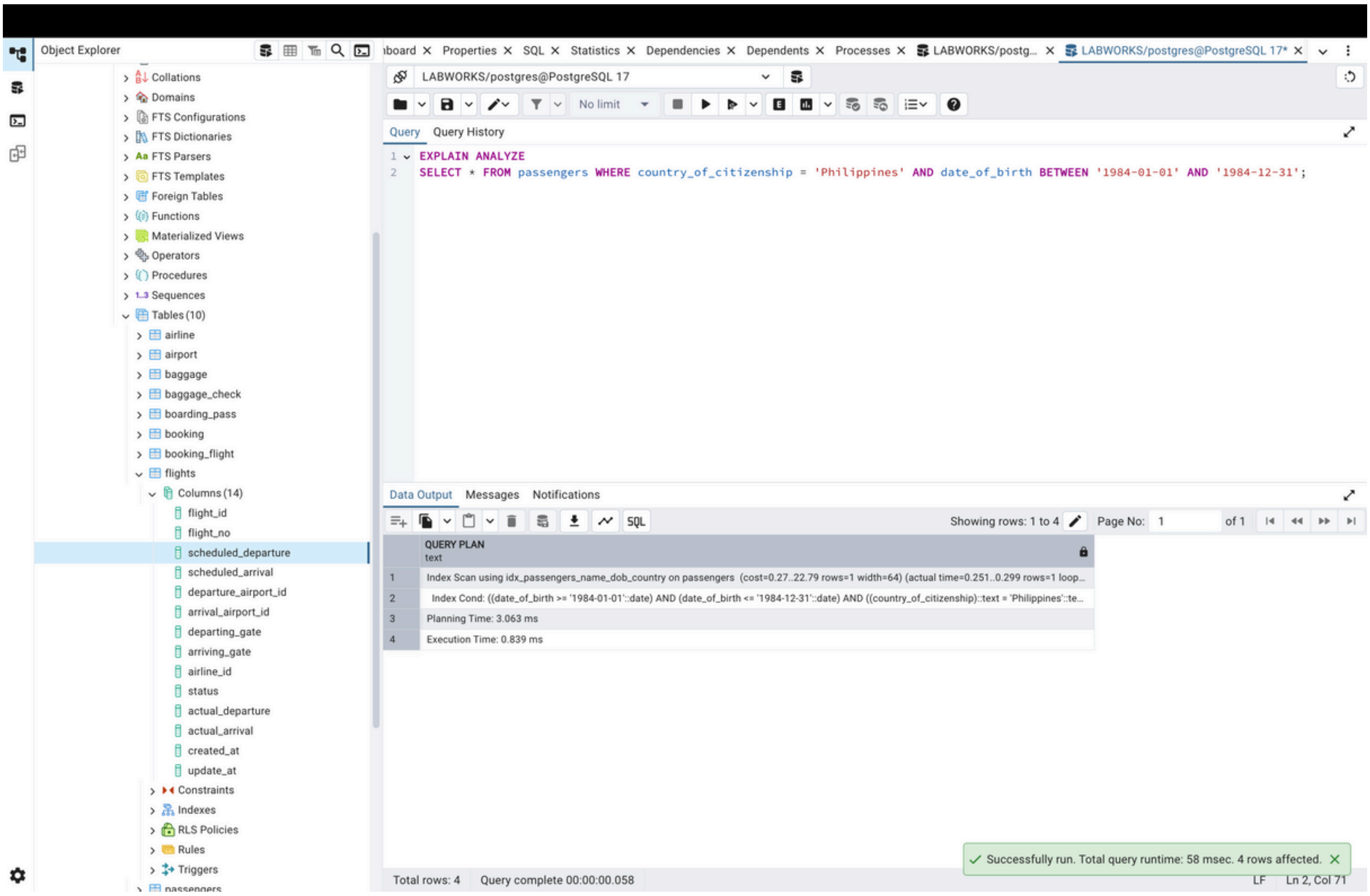
7. Create an index for the Passengers table. Use for that first name, last name, date of birth and country of citizenship. Then, write a SQL query to find a passenger who was born in Philippines and was born in 1984 and check if the query uses indexes or not. Give the explanation of the results.



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

```
CREATE INDEX

Query returned successfully in 39 msec.
```

✓ Query returned successfully in 39 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

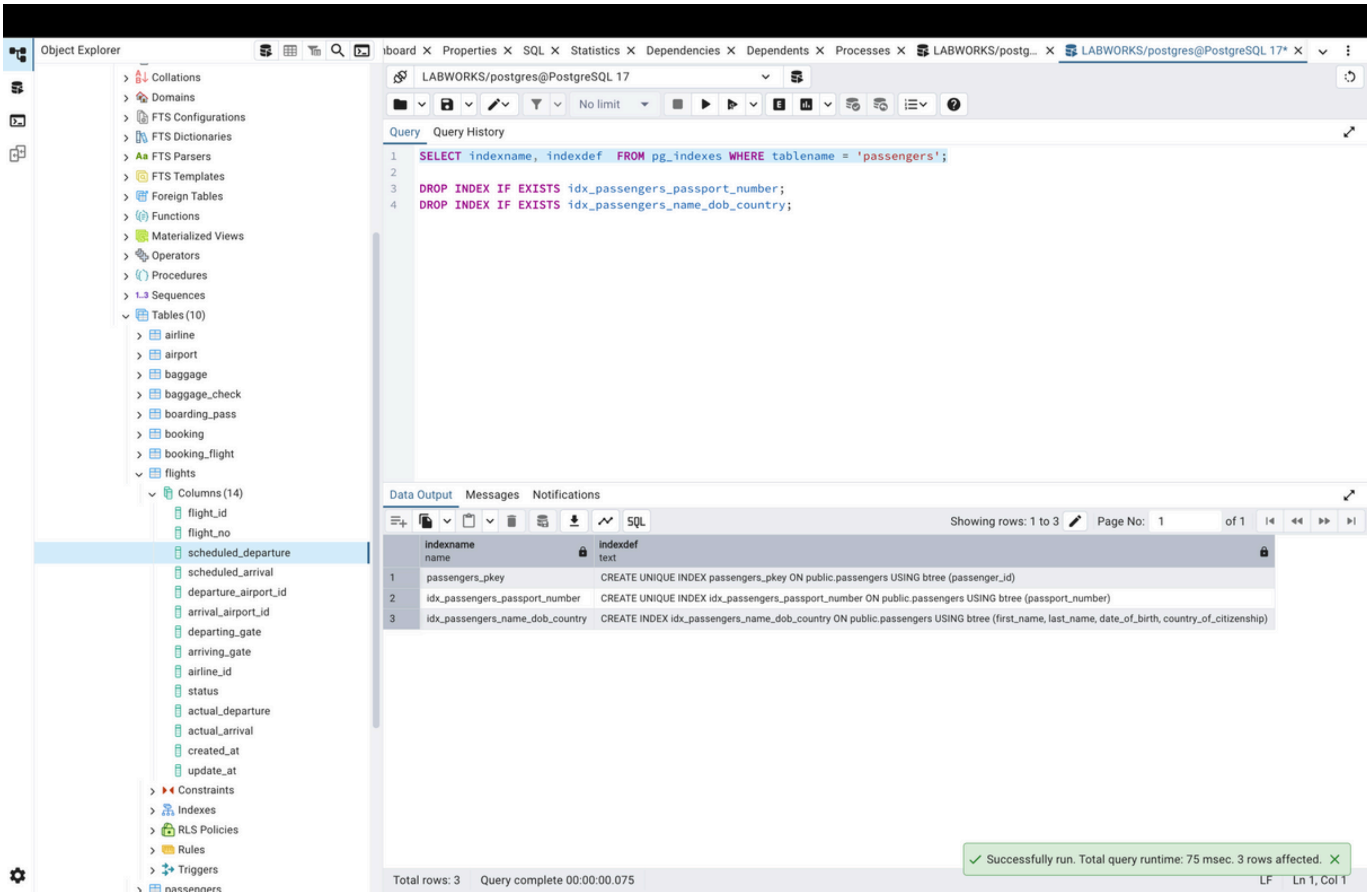
	QUERY PLAN text
1	Index Scan using idx_passengers_name_dob_country on passengers (cost=0.27..22.79 rows=1 width=64) (actual time=0.251..0.299 rows=1 loop...
2	Index Cond: ((date_of_birth >= '1984-01-01'::date) AND (date_of_birth <= '1984-12-31'::date) AND ((country_of_citizenship)::text = 'Philippines'::te...
3	Planning Time: 3.063 ms
4	Execution Time: 0.839 ms

✓ Successfully run. Total query runtime: 58 msec. 4 rows affected.

Total rows: 4 Query complete 00:00:00.058 LF Ln 2, Col 71

PostgreSQL used the index to find passengers from the Philippines born in 1984. Because of the index, it didn’t need to scan the whole table, so the query ran much faster, less than 1 millisecond.

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



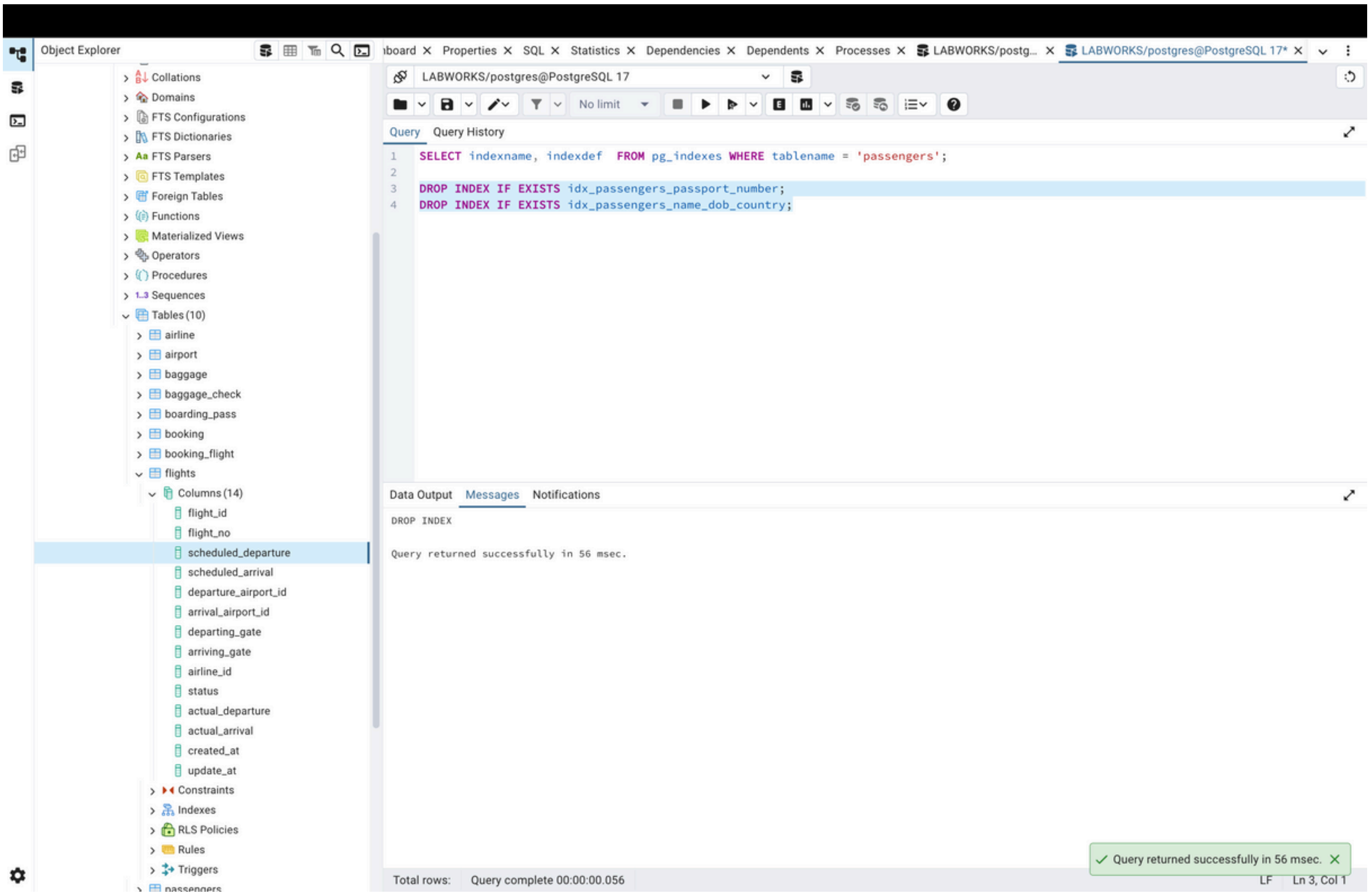
```sql
SELECT indexname, indexdef  FROM pg_indexes WHERE tablename = 'passengers';

DROP INDEX IF EXISTS idx_passengers_passport_number;
DROP INDEX IF EXISTS idx_passengers_name_dob_country;
```

Showing rows: 1 to 3

	indexname name	indexdef text
1	passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
2	idx_passengers_passport_number	CREATE UNIQUE INDEX idx_passengers_passport_number ON public.passengers USING btree (passport_number)
3	idx_passengers_name_dob_country	CREATE INDEX idx_passengers_name_dob_country ON public.passengers USING btree (first_name, last_name, date_of_birth, country_of_citizenship)

✓ Successfully run. Total query runtime: 75 msec. 3 rows affected.

Total rows: 3   Query complete 00:00:00.075   LF   Ln 1, Col 1



```sql
SELECT indexname, indexdef  FROM pg_indexes WHERE tablename = 'passengers';

DROP INDEX IF EXISTS idx_passengers_passport_number;
DROP INDEX IF EXISTS idx_passengers_name_dob_country;
```

```
DROP INDEX

Query returned successfully in 56 msec.
```

✓ Query returned successfully in 56 msec.

Total rows: Query complete 00:00:00.056   LF   Ln 3, Col 1