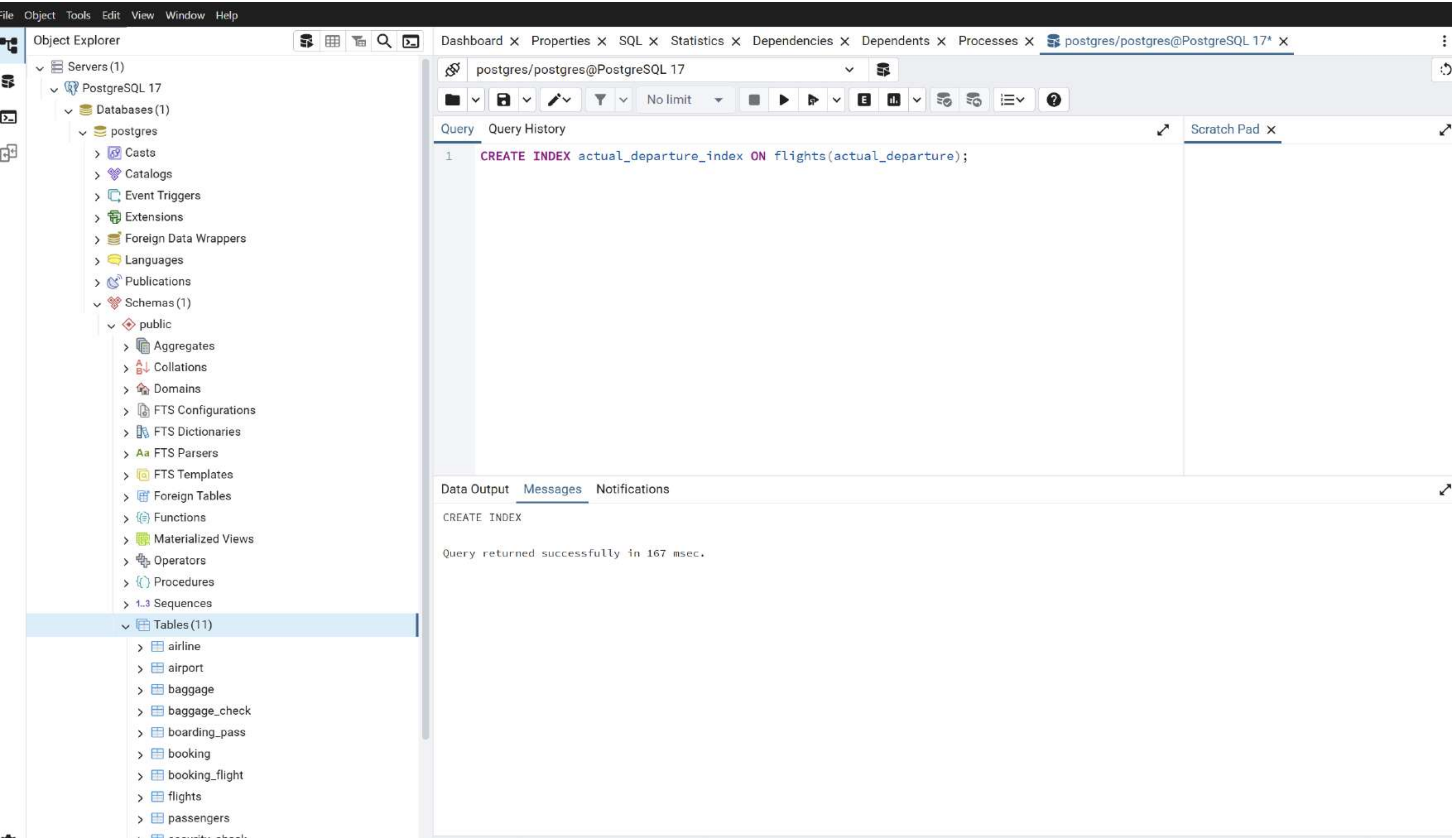
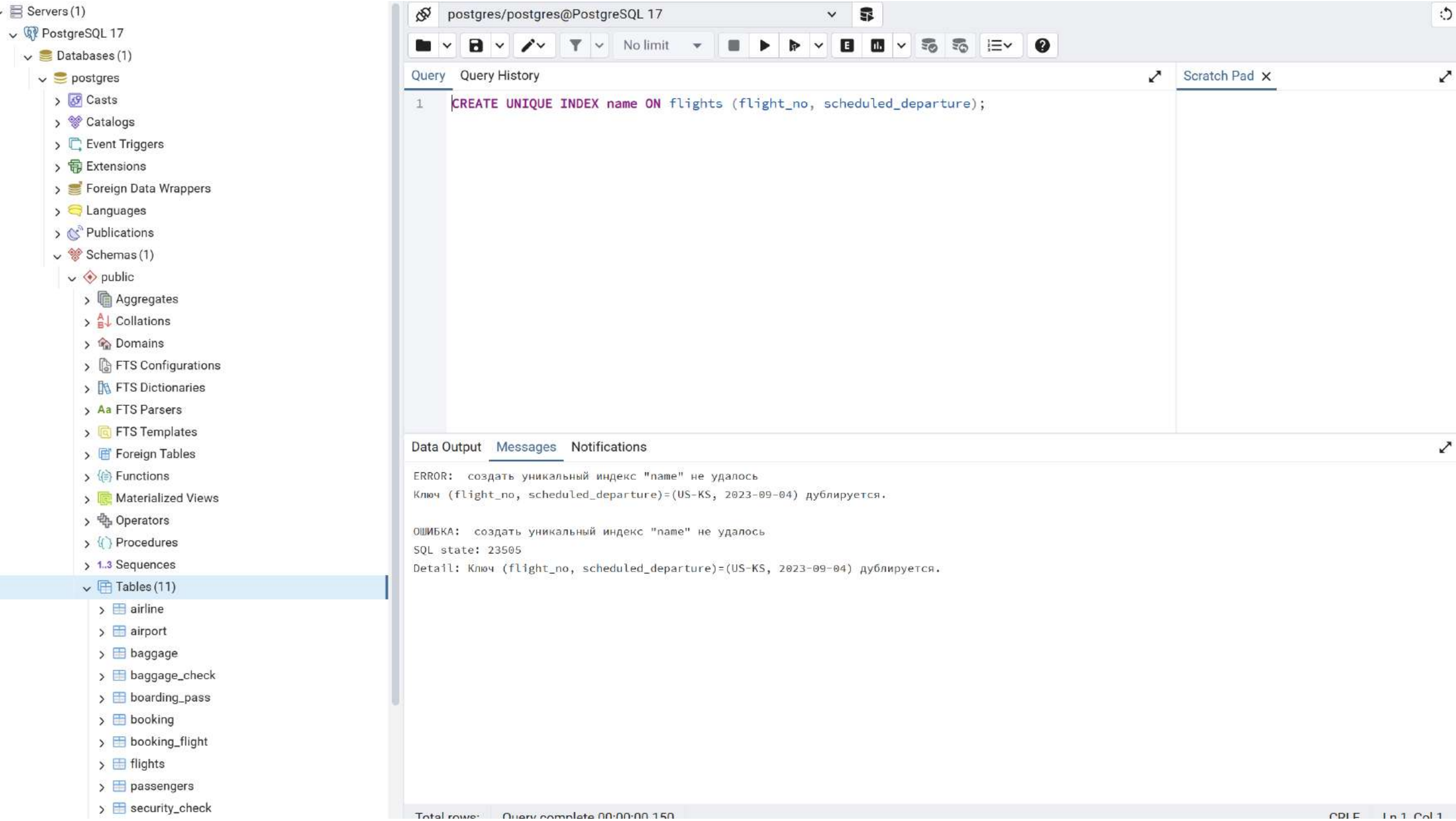


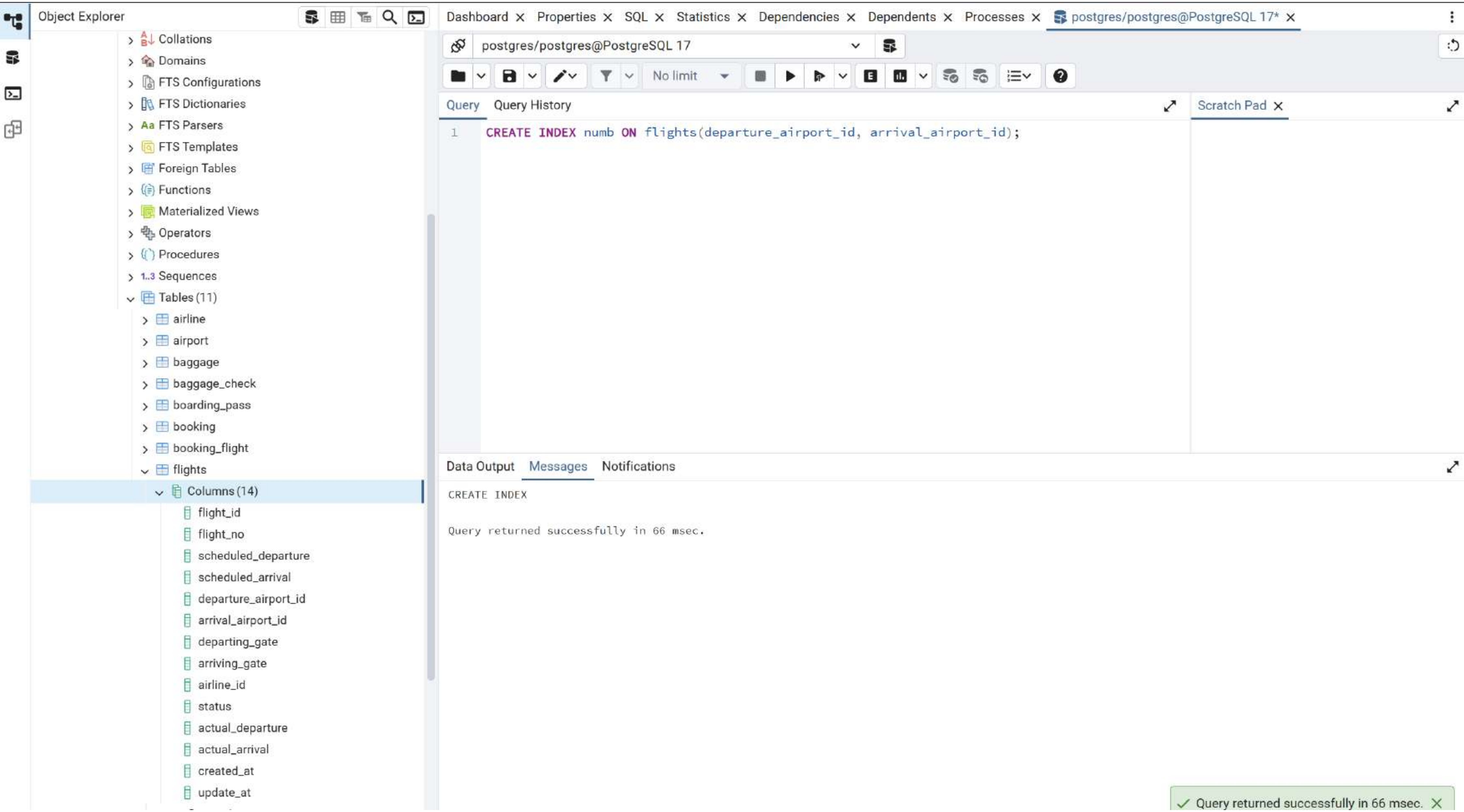
```sql
CREATE INDEX actual_departure_index ON flights(actual_departure);
```

```
CREATE INDEX

Query returned successfully in 167 msec.
```



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

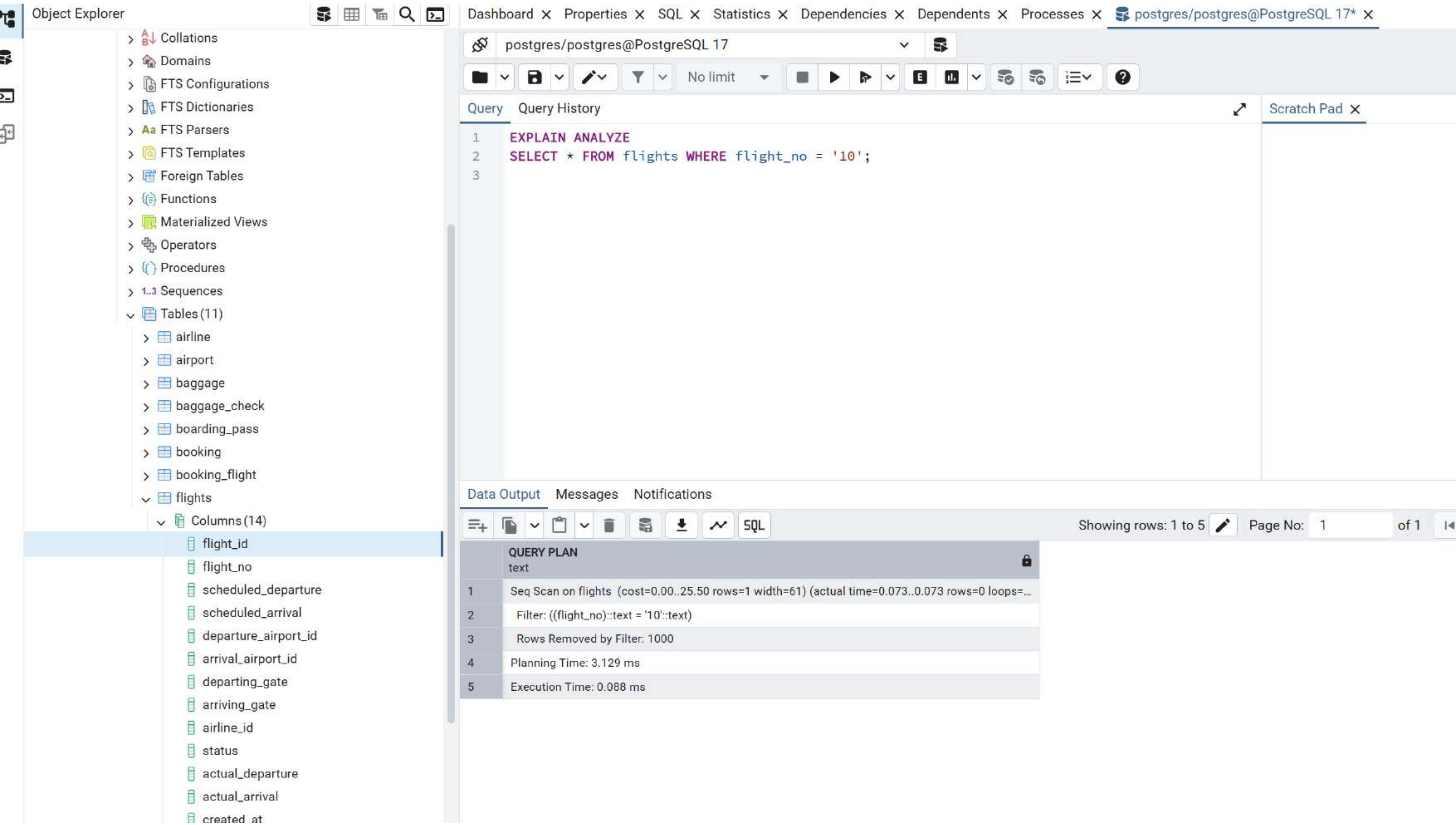
```
ERROR:  создать уникальный индекс "name" не удалось
Ключ (flight_no, scheduled_departure)=(US-KS, 2023-09-04) дублируется.

ОШИБКА:  создать уникальный индекс "name" не удалось
SQL state: 23505
Detail: Ключ (flight_no, scheduled_departure)=(US-KS, 2023-09-04) дублируется.
```



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

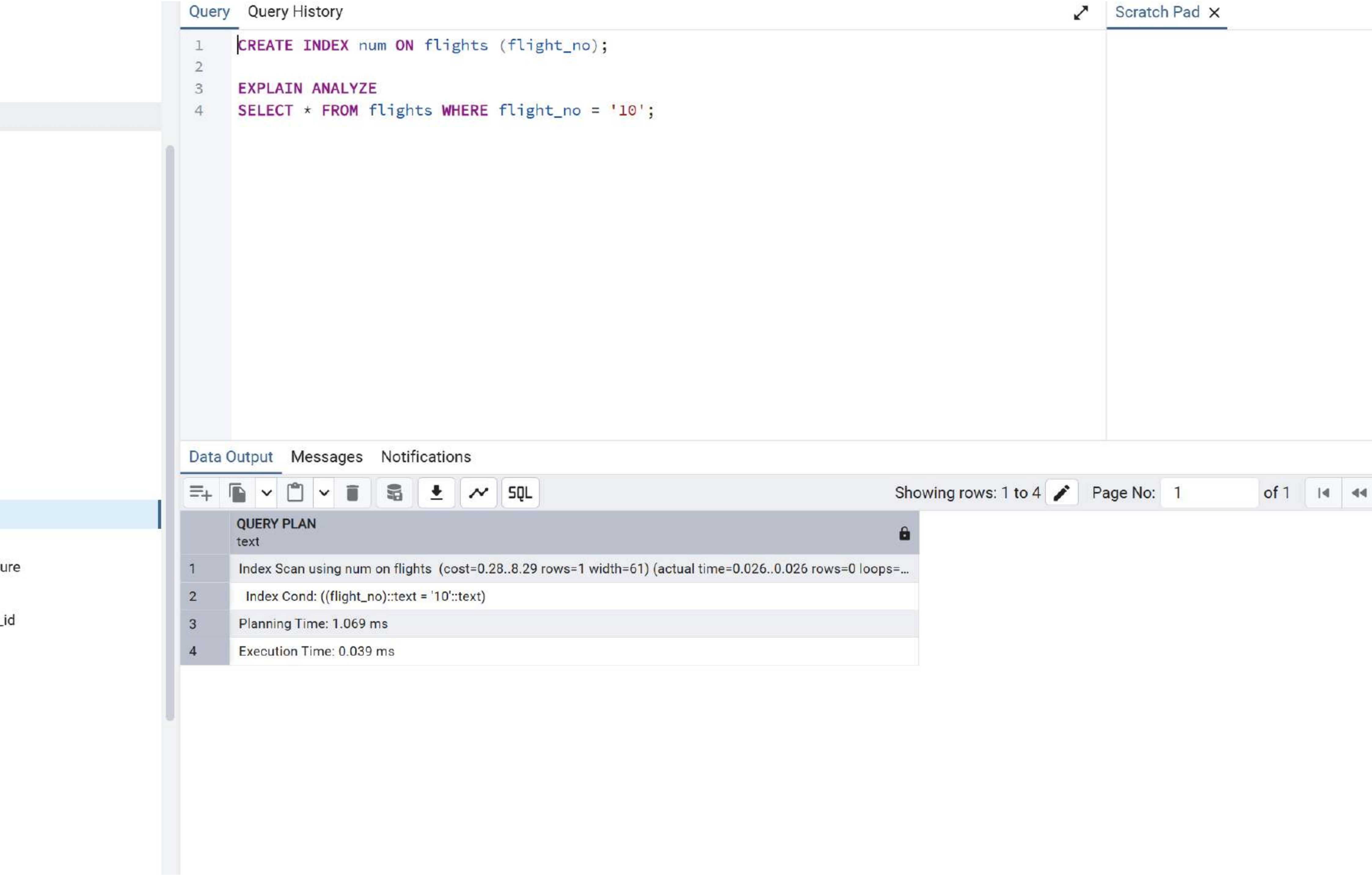
```
CREATE INDEX

Query returned successfully in 66 msec.
```

✓ Query returned successfully in 66 msec.



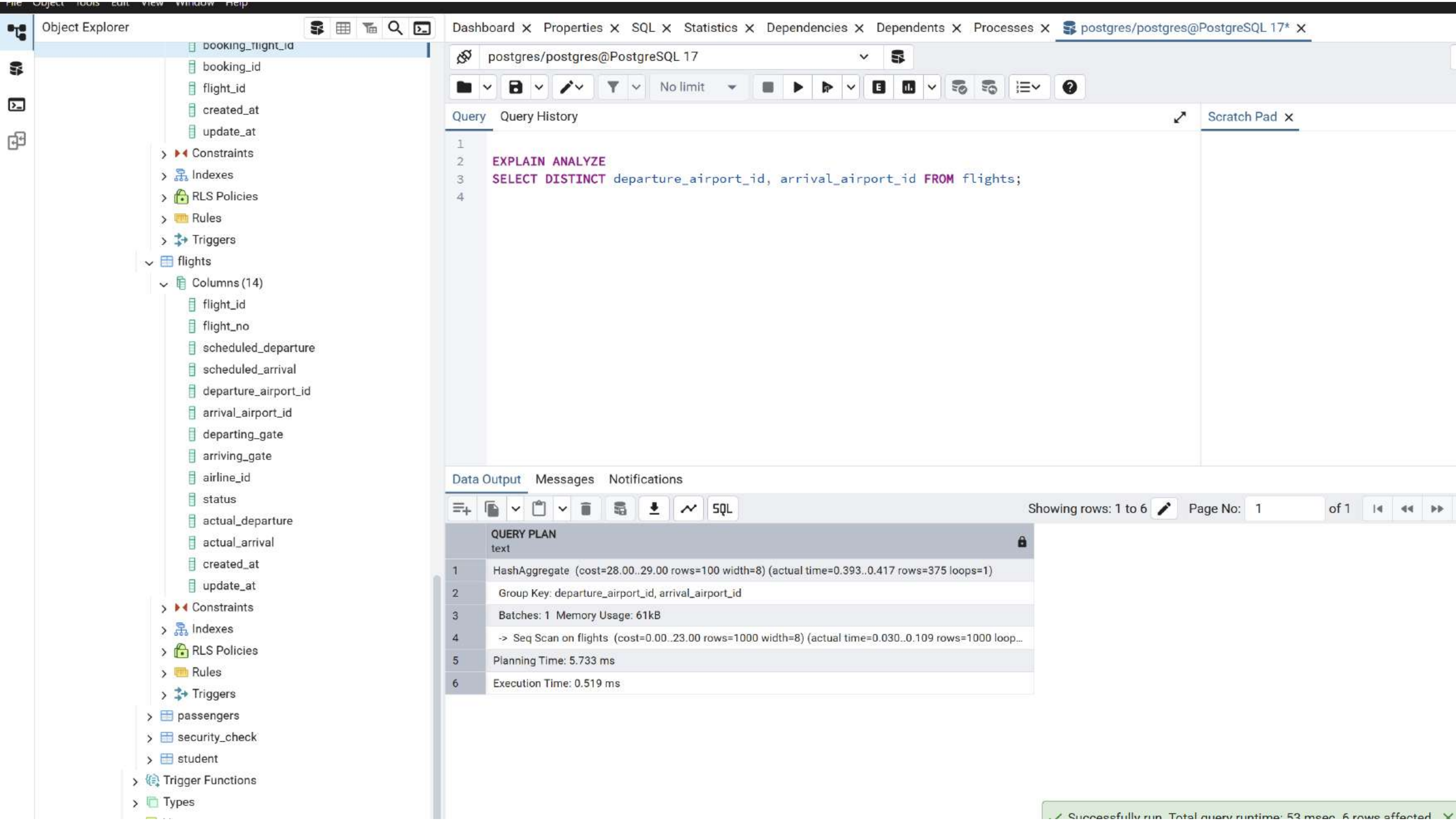
```
EXPLAIN ANALYZE
SELECT * FROM flights WHERE flight_no = '10';
```

	QUERY PLAN text
1	Seq Scan on flights (cost=0.00..25.50 rows=1 width=61) (actual time=0.073..0.073 rows=0 loops=...
2	Filter: ((flight_no)::text = '10'::text)
3	Rows Removed by Filter: 1000
4	Planning Time: 3.129 ms
5	Execution Time: 0.088 ms



```sql
CREATE INDEX num ON flights (flight_no);

EXPLAIN ANALYZE
SELECT * FROM flights WHERE flight_no = '10';
```

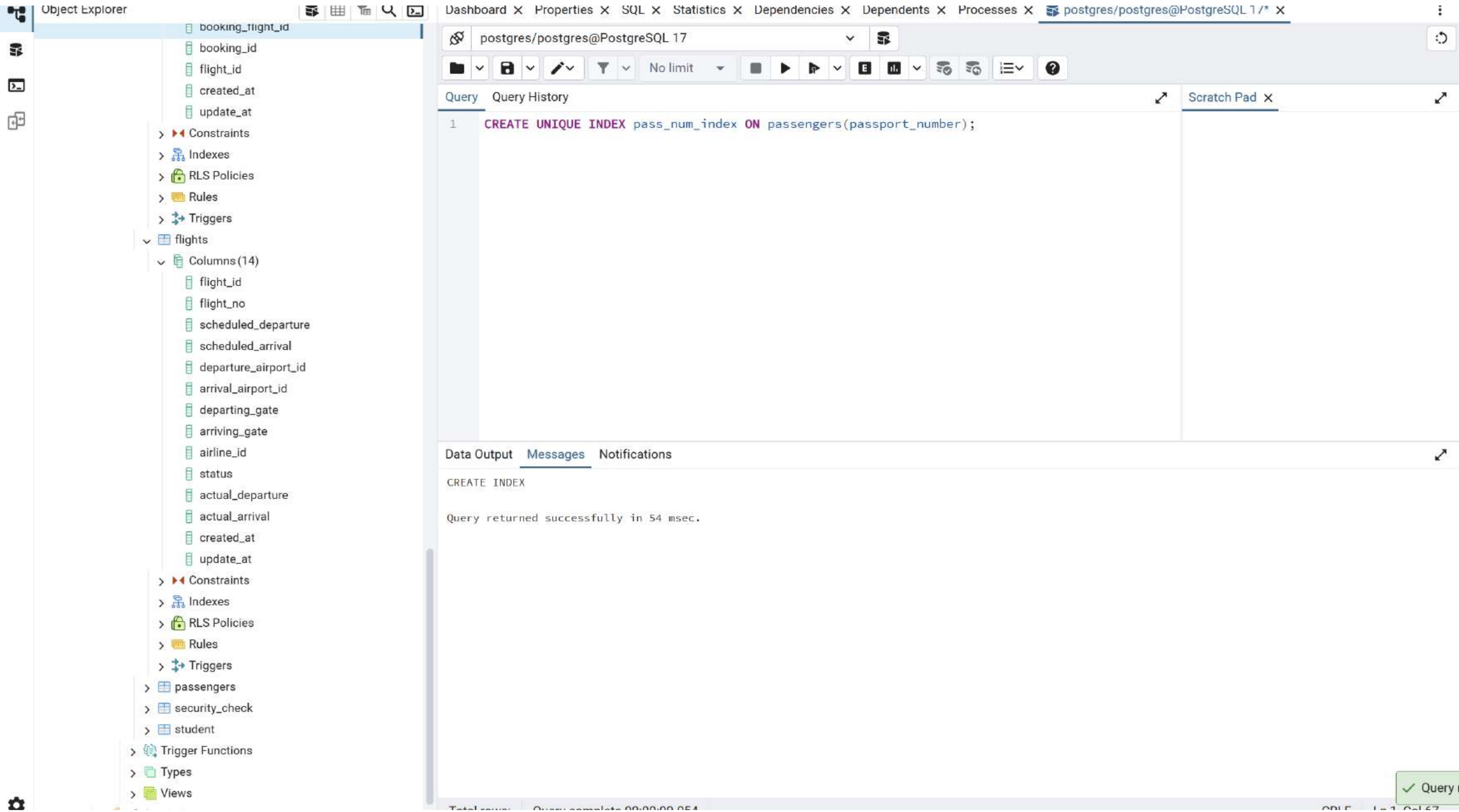
	QUERY PLAN text
1	Index Scan using num on flights (cost=0.28..8.29 rows=1 width=61) (actual time=0.026..0.026 rows=0 loops=...
2	Index Cond: ((flight_no)::text = '10'::text)
3	Planning Time: 1.069 ms
4	Execution Time: 0.039 ms



```sql

EXPLAIN ANALYZE
SELECT DISTINCT departure_airport_id, arrival_airport_id FROM flights;

```

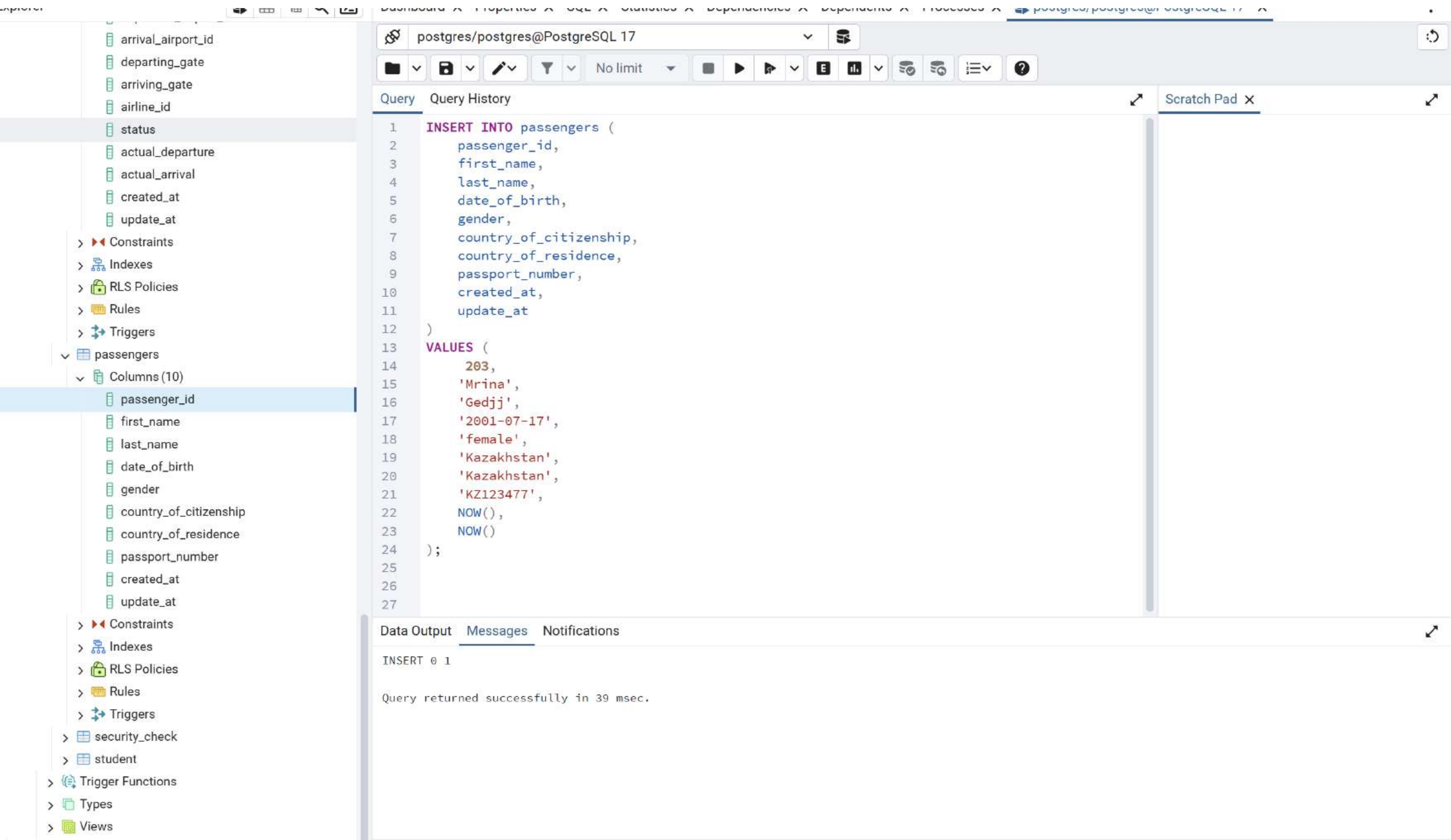
```
  QUERY PLAN
  text
1 HashAggregate  (cost=28.00..29.00 rows=100 width=8) (actual time=0.393..0.417 rows=375 loops=1)
2   Group Key: departure_airport_id, arrival_airport_id
3   Batches: 1  Memory Usage: 61kB
4   ->  Seq Scan on flights  (cost=0.00..23.00 rows=1000 width=8) (actual time=0.030..0.109 rows=1000 loop...
5 Planning Time: 5.733 ms
6 Execution Time: 0.519 ms
```

✓ Successfully run. Total query runtime: 53 msec. 6 rows affected.



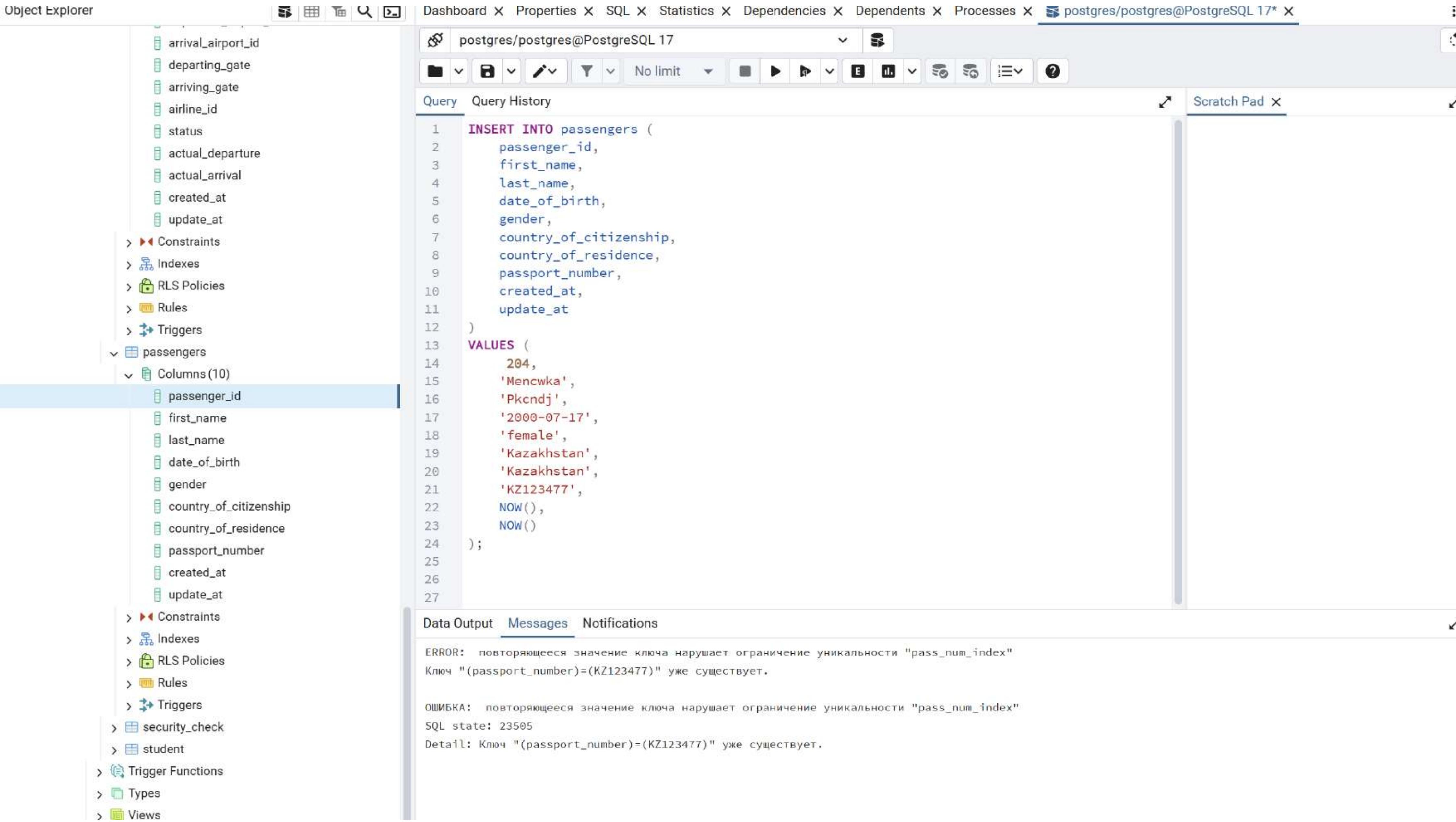
```sql
CREATE UNIQUE INDEX pass_num_index ON passengers(passport_number);
```

```
CREATE INDEX

Query returned successfully in 54 msec.
```



```sql
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 (
     203,
    'Mrina',
    'Gedjj',
    '2001-07-17',
    'female',
    'Kazakhstan',
    'Kazakhstan',
    'KZ123477',
    NOW(),
    NOW()
);
```

```
INSERT 0 1

Query returned successfully in 39 msec.
```

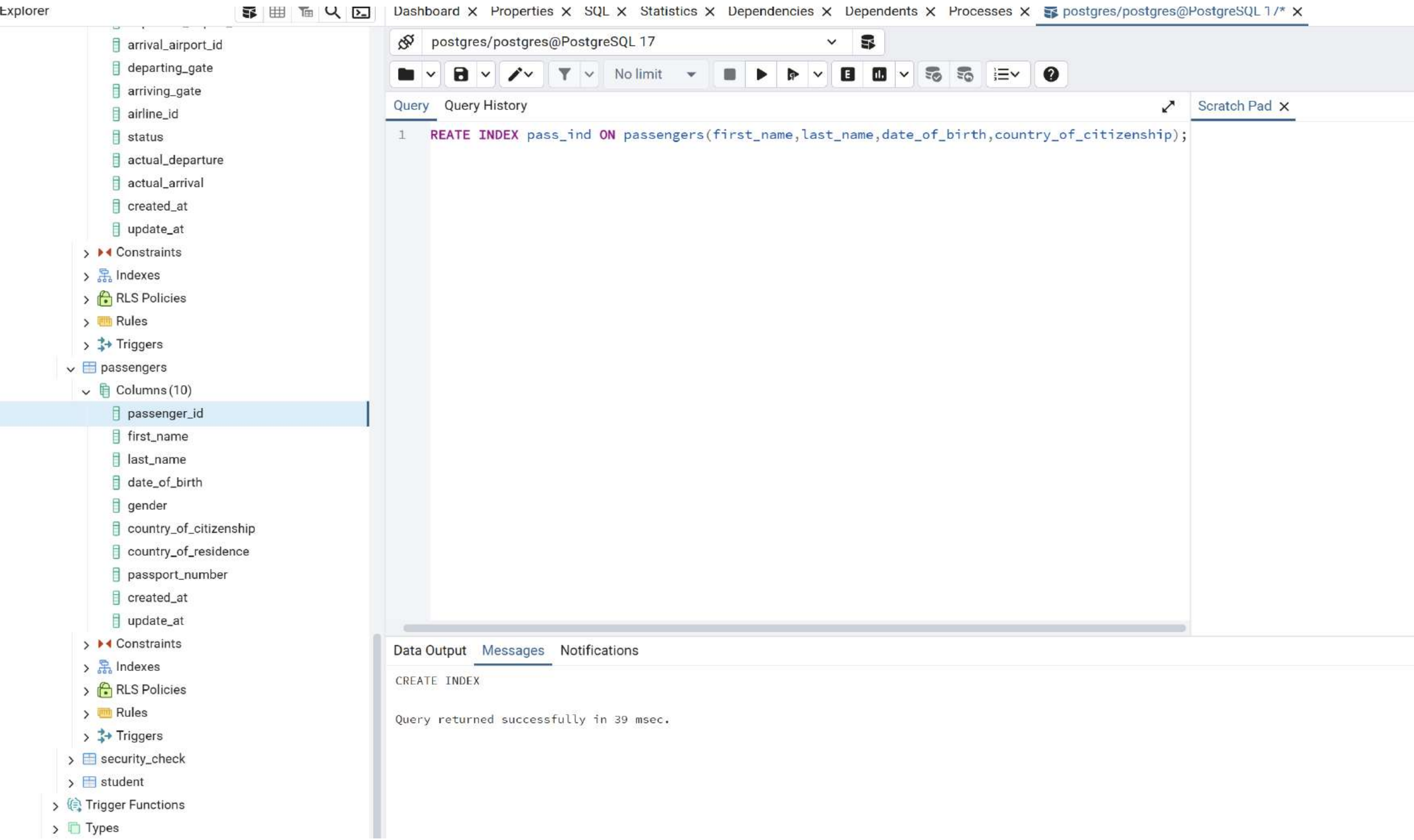


```sql
INSERT INTO passengers (
    passenger_id,
    first_name,
    last_name,
    date_of_birth,
    gender,
    country_of_citizenship,
    country_of_residence,
    passport_number,
    created_at,
    update_at
)
VALUES (
     204,
    'Mencwka',
    'Pkcndj',
    '2000-07-17',
    'female',
    'Kazakhstan',
    'Kazakhstan',
    'KZ123477',
    NOW(),
    NOW()
);
```

```
ERROR:  повторяющееся значение ключа нарушает ограничение уникальности "pass_num_index"
Ключ "(passport_number)=(KZ123477)" уже существует.

ОШИБКА:  повторяющееся значение ключа нарушает ограничение уникальности "pass_num_index"
SQL state: 23505
Detail: Ключ "(passport_number)=(KZ123477)" уже существует.
```

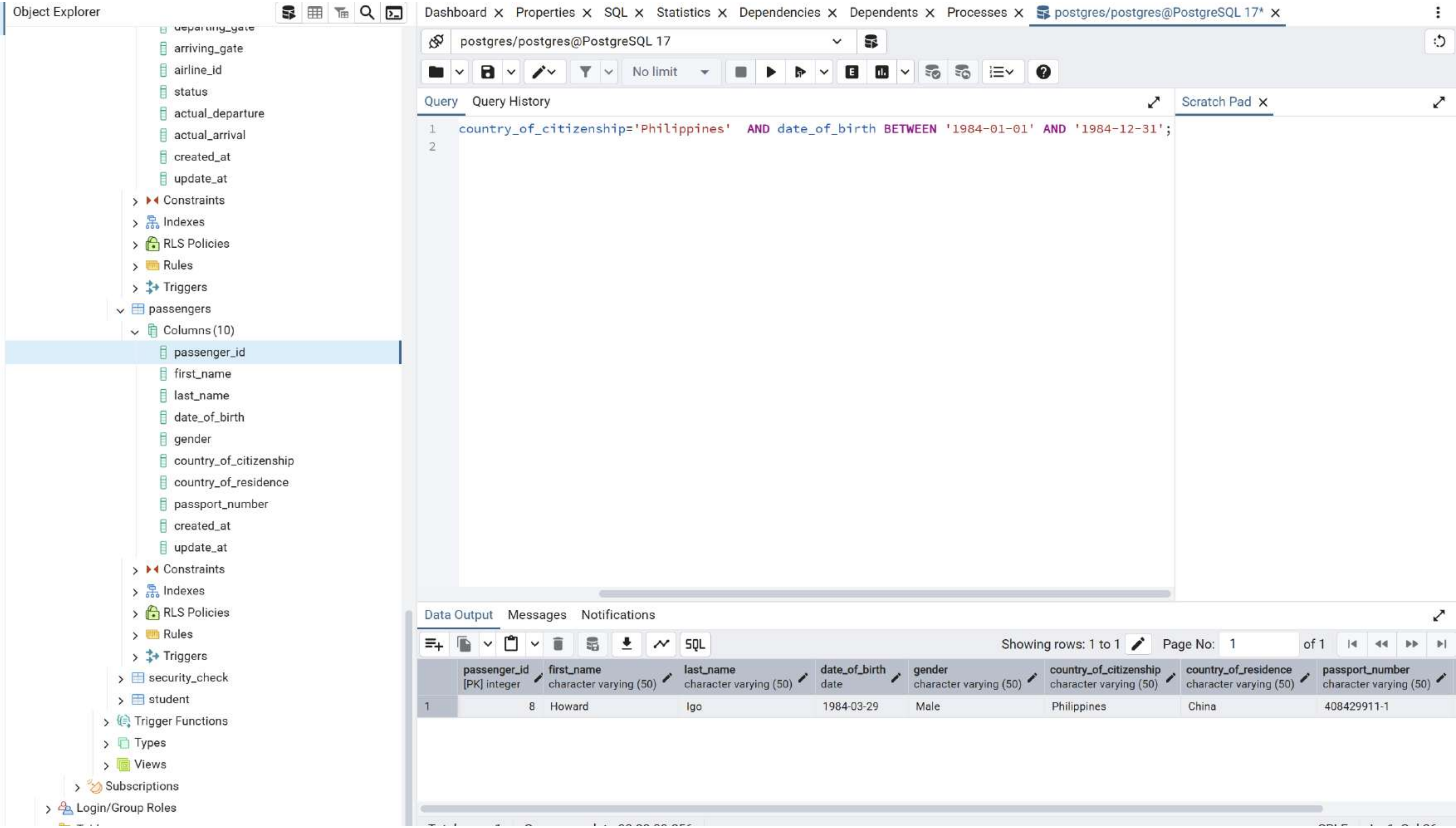
6 When you create a unique index, PostgreSQL automatically ensures that the values in that column are all unique.

If you try to insert a row with a passport number that already exists, the database rejects it to keep data integrity.



```
REATE INDEX pass_ind ON passengers(first_name,last_name,date_of_birth,country_of_citizenship);
```

```
CREATE INDEX

Query returned successfully in 39 msec.
```



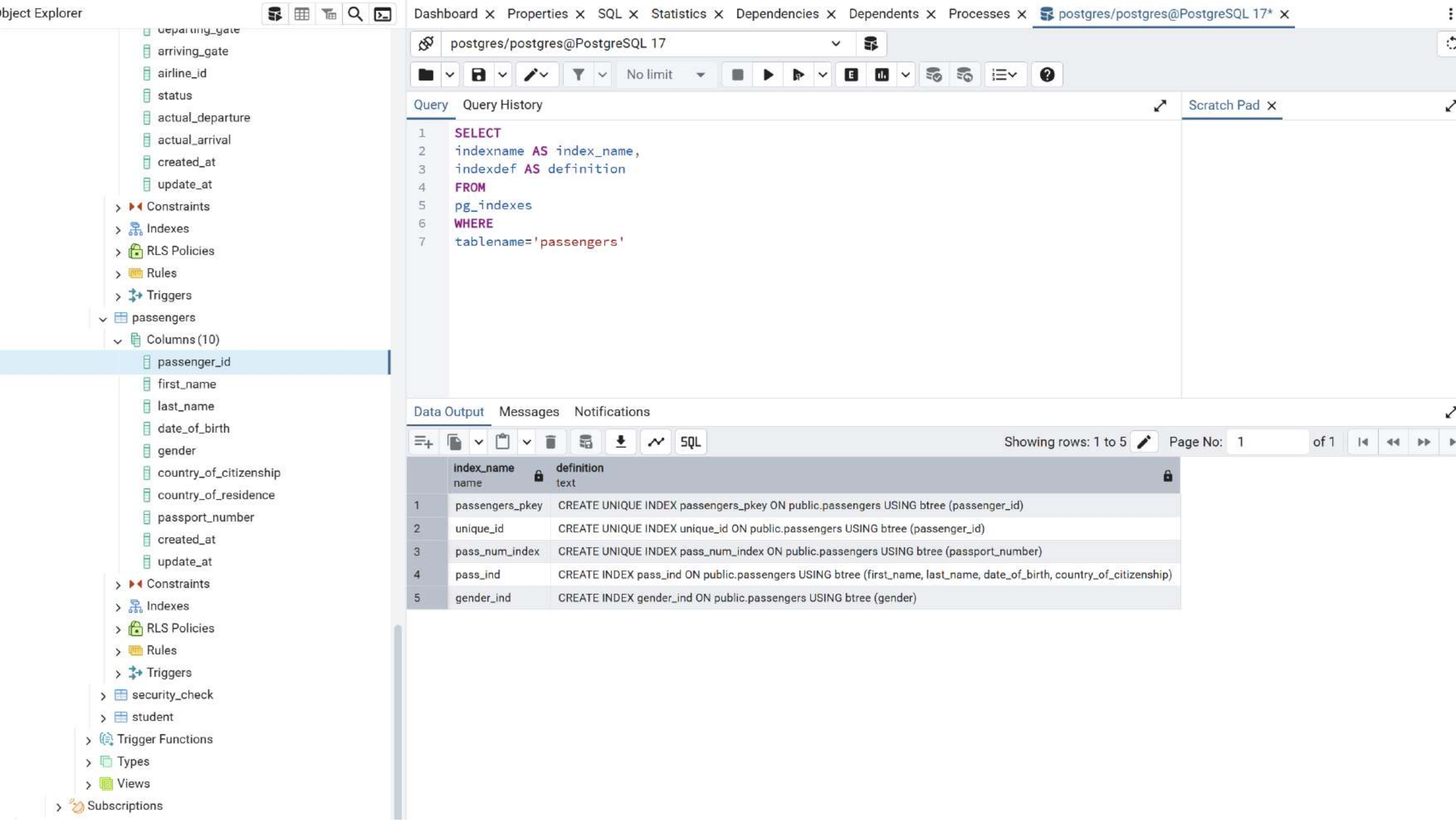
```
country_of_citizenship='Philippines'  AND date_of_birth BETWEEN '1984-01-01' AND '1984-12-31';
```

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

	passenger_id [PK] integer	first_name character varying (50)	last_name character varying (50)	date_of_birth date	gender character varying (50)	country_of_citizenship character varying (50)	country_of_residence character varying (50)	passport_number character varying (50)
1	8	Howard	Igo	1984-03-29	Male	Philippines	China	408429911-1

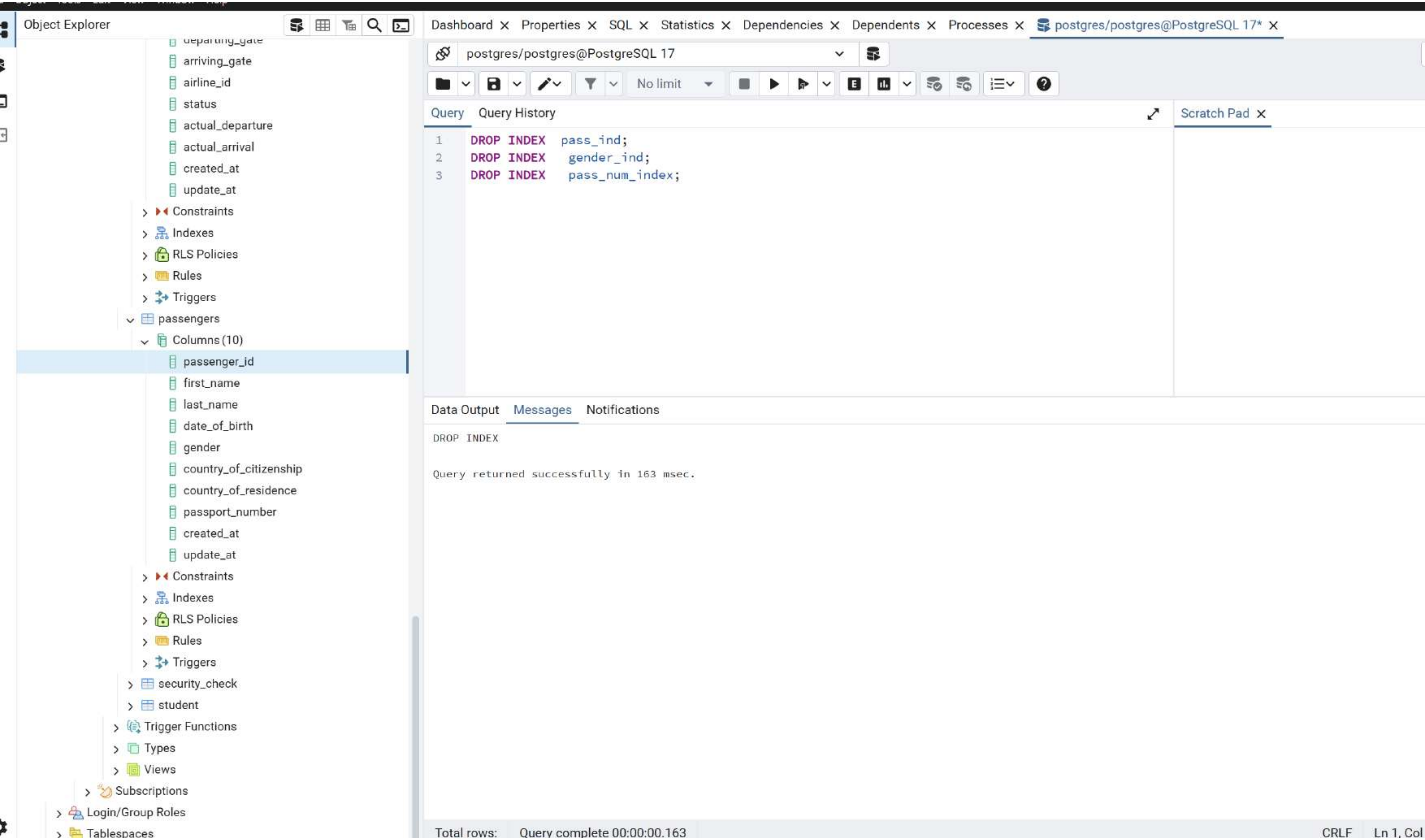
7 The query used the index because the conditions matched the indexed columns directly Using BETWEEN allowed the database to efficiently look up only the rows within the 1984 date range instead of scanning the whole table.

This makes the query faster and shows that the index is working as intended



```
SELECT
indexname AS index_name,
indexdef AS definition
FROM
pg_indexes
WHERE
tablename='passengers'
```

	index_name name	definition text
1	passengers_pkey	CREATE UNIQUE INDEX passengers_pkey ON public.passengers USING btree (passenger_id)
2	unique_id	CREATE UNIQUE INDEX unique_id ON public.passengers USING btree (passenger_id)
3	pass_num_index	CREATE UNIQUE INDEX pass_num_index ON public.passengers USING btree (passport_number)
4	pass_ind	CREATE INDEX pass_ind ON public.passengers USING btree (first_name, last_name, date_of_birth, country_of_citizenship)
5	gender_ind	CREATE INDEX gender_ind ON public.passengers USING btree (gender)



```sql
DROP INDEX  pass_ind;
DROP INDEX   gender_ind;
DROP INDEX   pass_num_index;
```

```
DROP INDEX

Query returned successfully in 163 msec.
```

Total rows: Query complete 00:00:00.163