

Object Explorer Database database_subj/postgres@PostgreSQL 17* Tools

Tables (10)

- airline
- airport
- baggage
- baggage_cf
- boarding_passenger
- booking
- booking_file
- flights
- flights_i
- flight_log
- scheduled
- scheduled_i
- departure
- arrival
- departure_i
- arriving
- airline_i
- status
- actual
- actual_i
- created
- update

CREATE INDEX idx_actual_departure
ON flights(actual_departure);

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 43 msec.

Total rows: Query complete 00:00:00.043

✓ Query returned successfully in 43 msec. ✎

Update available

You are currently running version 9.7 of pgAdmin 4, however the current version is 9.9.

Download Update

LF Ln 2, Col 30

This screenshot shows the pgAdmin 4 interface for PostgreSQL 17. The left sidebar displays the Object Explorer with a tree view of database objects. The 'flights' table is selected, and its 'Columns' node is expanded, showing various columns like flight_id, flight_log, scheduled, departure, arrival, and airline_id. The main pane contains a query editor window with the following content:

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

The 'Messages' tab is active, showing the message "Query returned successfully in 43 msec." A green status bar at the bottom right also displays this message. A yellow 'Update available' notification is present, stating "You are currently running version 9.7 of pgAdmin 4, however the current version is 9.9." and includes a "Download Update" button. The bottom status bar shows "Total rows: Query complete 00:00:00.043".

Object database_subj/postgres@PostgreSQL 17* X

Indexes RLS Policies Rules Triggers airport baggage baggage_check boarding_pass booking booking_flight flights Columns (14) flight_id flight_no scheduled_departure scheduled_arrival departure_airport_id arrival_airport_id departing_gate arriving_gate airline_id status actual_departure actual_arrival created_at update_at Constraints Indexes (1) idx_actual_departure RLS Policies Rules Triggers passengers security_check Trigger Functions Types

database_subj/postgres@PostgreSQL 17

No limit ▾ E ▾

Query History Scratch Pad X

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

Data Output Messages Notifications

ERROR: relation "unique_flight_schedule" already exists
SQL state: 42P07

Total rows: Query complete 00:00:00.039 LF Ln 2, Col 44

Object E T S W M F X database_subj/postgres@PostgreSQL 17* X

Indexes RLS Policies Rules Triggers

airport baggage baggage_cl boarding_p... booking booking_fli... flights

Columns flight_l flight_r schedl schedr depart... arrival... depart... arriving... airline... status actual_... actual_... created update

Constraints Indexes (idx_ac)

RLS Policies Rules Triggers

passengers security_ch

Trigger Functions Types

Query History

Query Scratch Pad

CREATE INDEX idx_departure_arrival
ON flights(departure_airport_id, arrival_airport_id);

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 38 msec.

Total rows: Query complete 00:00:00.038 ✓ Query returned successfully in 38 msec. LF Ln 2, Col 54

Object Explorer Database database_subj/postgres@PostgreSQL 17*

File Edit View Insert Tools Window Help

Indexes RLS Policies Rules Triggers

airport baggage baggage_cl boarding_pass booking booking_flight flights

Columns flight_id flight_number scheduled_time departure_time arrival_time departure_time arriving_time airline status actual_status created updated

Constraints

Indexes (idx_ac)

RLS Policies Rules Triggers

passenger security_change Trigger Functions Types

Query History

EXPLAIN ANALYZE

```
1  SELECT *
2   FROM flights
3   WHERE departure_airport_id = 1
4     AND arrival_airport_id = 2;
```

Scratch Pad

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..27.88 rows=2 width=61) (actual time=0.048..0.266 rows=2 loops=1)
2	Filter: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
3	Rows Removed by Filter: 990
4	Planning Time: 0.486 ms
5	Execution Time: 0.295 ms

Total rows: 5 Query complete 00:00:00.041

✓ Successfully run. Total query runtime: 41 msec. 5 rows affected.

LF Ln 5, Col 30

The screenshot shows the pgAdmin 4 interface with a query editor window. The left sidebar displays the database schema with tables like flights, passengers, and security_change. The main window shows a query in the 'Query' tab and its execution plan in the 'Data Output' tab. The query is an EXPLAIN ANALYZE of a SELECT statement filtering flights between two specific airports. The execution plan shows a sequential scan on the flights table, applying a filter for the specified airports, and removing 990 rows. The total planning and execution time is around 41 milliseconds, resulting in 5 rows affected. A success message at the bottom indicates the query was run successfully.

Object Explorer database_subj/postgres@PostgreSQL 17* X

Indexes RLS Policies Rules Triggers

airport baggage baggage_cl boarding_passengers booking booking_flights flights

Columns flight_id flight_number scheduled_time departure_time arrival_time departure_time arrival_time airline status actual_departure_time actual_arrival_time created update

Constraints

Indexes (idx_airport_id)

RLS Policies

Rules

Triggers

passengers security_channels Trigger Functions Types

database_subj/postgres@PostgreSQL 17

No limit

Query History

Execute script F5

Scratch Pad

EXPLAIN ANALYZE

SELECT *
FROM flights
WHERE departure_airport_id = 1
AND arrival_airport_id = 2;

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=0.064..0.067 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_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.056..0.056 rows=2 loops=1)
5 Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
6 Planning Time: 0.775 ms
7 Execution Time: 0.115 ms

Total rows: 7 Query complete 00:00:00.039

Successfully run. Total query runtime: 39 msec. 7 rows affected.

Object Explorer database_subj/postgres@PostgreSQL 17* X

Indexes RLS Policies Rules Triggers

airport baggage baggage_check boarding_pass booking booking_flight flights

Columns (14)

- flight_id
- flight_no
- scheduled_departure
- scheduled_arrival
- departure_airport_id
- arrival_airport_id
- departing_gate
- arriving_gate
- airline_id
- status
- actual_departure
- actual_arrival
- created_at
- update_at

Constraints

Indexes (1)

- idx_actual_departure

RLS Policies Rules Triggers

passengers security_check

Trigger Functions

Types

database_subj/postgres@PostgreSQL 17

No limit

Query Query History Scratch Pad

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

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=0.038..0.042 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_departure_arrival (cost=0.00..4.29 rows=2 width=0) (actual time=0.029..0.029 rows=2 loops=1)
5 Index Cond: ((departure_airport_id = 1) AND (arrival_airport_id = 2))
6 Planning Time: 0.190 ms
7 Execution Time: 0.079 ms

Total rows: 7 Query complete 00:00:00.092 LF Ln 5, Col 30

database_subj/postgres@PostgreSQL 17* | public.passengers... X

database_subj/postgres@PostgreSQL 17 No limit

Query History Scratch Pad

```
1 -- In my own words: I created a unique index on passport_number to ensure each record is unique.
2 -- When inserting new passengers, PostgreSQL checks this index.
3 -- If the passport_number already exists, the insertion fails with a duplicate key error.
4 -- If the value is unique, the insertion succeeds.
5
6 -- Create a unique index on passport_number
7 CREATE UNIQUE INDEX IF NOT EXISTS unique_passport_numbers
8 ON passengers(passport_number);
9
10 -- Check that the index exists
11 SELECT indexname, indexdef
12 FROM pg_indexes
13 WHERE tablename = 'passengers';
14
15 -- Insert two new passengers
16 INSERT INTO passengers (
17     passenger_id, first_name, last_name, passport_number,
18     date_of_birth, gender, country_of_citizenship, country_of_residence,
19     created_at, update_at
20 )
21 VALUES
22     (201, 'John', 'Doe', 'P123456', '1990-01-01', 'M', 'USA', 'USA', CURRENT_DATE, CURRENT_DATE),
23     (202, 'Jane', 'Smith', 'P654321', '1992-02-02', 'F', 'USA', 'USA', CURRENT_DATE, CURRENT_DATE);
```

Data Output Messages Notifications

INSERT 0 2

Query returned successfully in 31 msec.

Total rows: Query complete 00:00:00.031 LF Ln 10, Col 31

The screenshot shows the pgAdmin 4 interface with the following details:

- Left Sidebar:** Shows the database structure for the "public" schema, including tables like "passengers", columns like "passenger_id", and various indexes and constraints.
- Query Editor:** The main window displays a SQL script for creating a composite index on the "passenger_info" column of the "passengers" table. The script includes explanatory comments and an EXPLAIN ANALYZE command to demonstrate its performance.
- Data Output:** Below the query editor, the "Data Output" tab is active, showing the execution plan for the query. The plan indicates a sequential scan on the "passengers" table followed by a filter operation.
- Status Bar:** At the bottom, it shows "Query executed with server cursor" and "Query complete 00:00:00.042".

```
-- ⚡ Create a composite index
CREATE INDEX IF NOT EXISTS idx_passenger_info
ON passengers(first_name, last_name, date_of_birth, country_of_citizenship);

-- ⚡ Query using a range to allow index usage
EXPLAIN ANALYZE
SELECT *
FROM passengers
WHERE date_of_birth BETWEEN '1984-01-01' AND '1984-12-31'
    AND country_of_citizenship = 'Philippines';

-- ⚡ Explanation:
-- The composite index on first_name, last_name, date_of_birth, and country_of_citizenship
-- helps PostgreSQL quickly locate rows matching the query.
-- We use a date range for date_of_birth instead of a function like date_part('year', ...)
-- because functions prevent the index from being used.
-- In the EXPLAIN ANALYZE output, if you see "Index Scan" or "Bitmap Index Scan",
-- it means PostgreSQL is using the index to fetch rows efficiently.
-- If you see "Seq Scan", the index is not being used, and PostgreSQL is scanning the whole table.
-- Using the index significantly reduces query time, especially on large tables.
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of the database schema, including nodes for 'public' and 'passenger'. The main window contains a query editor with the following SQL code:

```
1 SELECT indexname, indexdef
2 FROM pg_indexes
3 WHERE tablename = 'passengers';
4
5 DROP INDEX IF EXISTS idx_passenger_info;
6
7 DROP INDEX IF EXISTS unique_passport_numbers;
```

The 'Data Output' tab shows the results of the executed queries:

```
DROP INDEX
Query returned successfully in 46 msec.
```

The status bar at the bottom indicates: 'Query executed with server cursor' and 'Query complete 00:00:00.046'. A green message box in the bottom right corner states: '✓ Query returned successfully in 46 msec. LF Ln 7, Col 45'.