

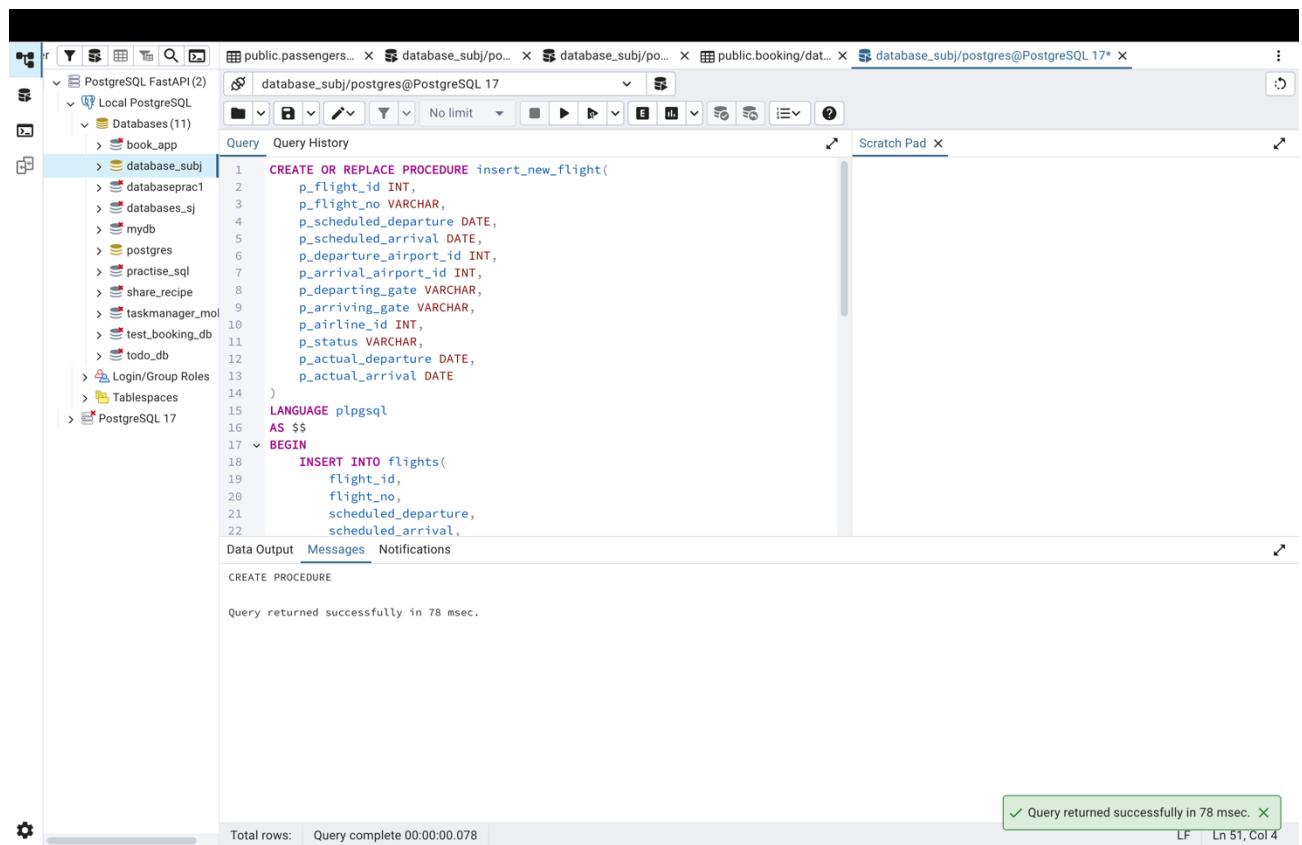
Laboratory work 10

We continue to work with the database from the previous laboratory works.

Take a full-page screenshot that covers the code and results of each task.

STORED PROCEDURES and FUNCTION.

1. Create a stored procedure to insert a new flight into the flights table.



The screenshot shows the pgAdmin 4 interface with a query editor window. The left sidebar displays a tree view of databases, including 'PostgreSQL FastAPI (2)', 'Local PostgreSQL', and several user databases like 'book_app', 'database_subj', 'practise_sql', and 'taskmanager_mol'. The main window has a 'Query' tab selected, containing the following SQL code:

```
CREATE OR REPLACE PROCEDURE insert_new_flight(
    p_flight_id INT,
    p_flight_no VARCHAR,
    p_scheduled_departure DATE,
    p_scheduled_arrival DATE,
    p_departure_airport_id INT,
    p_arrival_airport_id INT,
    p_departing_gate VARCHAR,
    p_arriving_gate VARCHAR,
    p_airline_id INT,
    p_status VARCHAR,
    p_actual_departure DATE,
    p_actual_arrival DATE
)
LANGUAGE plpgsql
AS $$

BEGIN
    INSERT INTO flights(
        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
    )
    VALUES (
        p_flight_id,
        p_flight_no,
        p_scheduled_departure,
        p_scheduled_arrival,
        p_departure_airport_id,
        p_arrival_airport_id,
        p_departing_gate,
        p_arriving_gate,
        p_airline_id,
        p_status,
        p_actual_departure,
        p_actual_arrival
    );
END;
```

Below the code, the 'Messages' tab shows the output:

```
CREATE PROCEDURE
Query returned successfully in 78 msec.
```

A green success message at the bottom right of the interface states: "Query returned successfully in 78 msec. ✓".

2. Create a stored procedure to update the status of a flight.

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

- Left Panel:** Shows the database tree under "Local PostgreSQL > Databases".
- Central Panel:** A query editor window titled "database_subj/postgres@PostgreSQL 17*" containing the following SQL code:


```

CREATE OR REPLACE PROCEDURE update_flight_status(
    p_flight_id INT,
    p_new_status VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM flights WHERE flight_id = p_flight_id) THEN
        RAISE EXCEPTION 'Flight with id % does not exist', p_flight_id;
    END IF;

    UPDATE flights
    SET
        status = p_new_status,
        update_at = CURRENT_DATE
    WHERE flight_id = p_flight_id;
END;
$$;
      
```
- Bottom Panel:** Status bar showing "Query returned successfully in 78 msec." and "Total rows: 0 Query complete 00:00:00.078".

3. Create a stored procedure that returns a list of flights departing from a specific airport.

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

- Left Panel:** Shows the database tree under "Local PostgreSQL > Databases".
- Central Panel:** A query editor window titled "database_subj/postgres@PostgreSQL 17*" containing the following SQL code:


```

CREATE OR REPLACE PROCEDURE list_flights_from_airport(p_airport_id INT)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT *
    FROM flights
    WHERE departure_airport_id = p_airport_id;
END;
$$;
      
```
- Bottom Panel:** Status bar showing "Query returned successfully in 42 msec." and "Total rows: 0 Query complete 00:00:00.042".

4. Create a function to calculate the average delay time of flights arriving at a specific airport.

The screenshot shows the pgAdmin 4 interface with the 'Query' tab selected. The code entered is:

```

1 CREATE OR REPLACE FUNCTION average_delay_at_airport(p_airport_id INT)
2 RETURNS NUMERIC
3 LANGUAGE plpgsql
4 AS $$
5 DECLARE
6     avg_delay NUMERIC;
7 BEGIN
8     SELECT AVG(actual_arrival - scheduled_arrival)
9     INTO avg_delay
10    FROM flights
11   WHERE arrival_airport_id = p_airport_id
12     AND actual_arrival IS NOT NULL;
13
14    RETURN avg_delay;
15
16 $$;

```

The 'Messages' tab shows the output:

```

CREATE FUNCTION
Query returned successfully in 41 msec.

```

The status bar at the bottom indicates "Total rows: 0 Query complete 00:00:00.041". A green message box in the bottom right corner says "Query returned successfully in 41 msec.".

5. Create a stored procedure that lists all passengers for a given flight number.

The screenshot shows the pgAdmin 4 interface with the 'Query' tab selected. The code entered is:

```

1 CREATE OR REPLACE PROCEDURE list_passengers_by_flight(
2     p_flight_no VARCHAR)
3 LANGUAGE plpgsql
4 AS $$
5 BEGIN
6     SELECT p.passenger_id,
7            p.first_name,
8            p.last_name,
9            p.date_of_birth,
10           p.gender,
11           p.passport_number
12      FROM passengers p
13     JOIN booking b ON p.passenger_id = b.passenger_id
14     JOIN booking_flight bf ON b.booking_id = bf.booking_id
15     JOIN flights f ON bf.flight_id = f.flight_id
16   WHERE f.flight_no = p_flight_no
17   ORDER BY p.last_name, p.first_name;
18
19 $$;

```

The 'Messages' tab shows the output:

```

CREATE PROCEDURE
Query returned successfully in 52 msec.

```

The status bar at the bottom indicates "Total rows: 0 Query complete 00:00:00.052". A green message box in the bottom right corner says "Query returned successfully in 52 msec.".

6. Create a stored procedure to find the passenger who has taken the greatest number of flights.

The screenshot shows the pgAdmin 4 interface with a database connection to 'public.flights/data...'. In the left sidebar, under 'Schemas(1) / public / Procedures', a new procedure is being created. The code in the query editor is:

```

CREATE OR REPLACE PROCEDURE flights()
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT p.passenger_id,
           p.first_name,
           p.last_name,
           COUNT(bf.flight_id) AS flights_taken
    FROM passengers p
    JOIN booking b ON p.passenger_id = b.passenger_id
    JOIN booking_flight bf ON b.booking_id = bf.booking_id
    GROUP BY p.passenger_id, p.first_name, p.last_name
    ORDER BY flights_taken DESC
    LIMIT 1;
END;
$$;

```

The 'Messages' tab shows the message 'CREATE PROCEDURE' and 'Query returned successfully in 44 msec.' at the bottom.

7. Create a stored procedure to find all flights that are delayed by more than 24 hours.

The screenshot shows the pgAdmin 4 interface with a database connection to 'public.flights/data...'. In the left sidebar, under 'Schemas(1) / public / Procedures', a new procedure is being created. The code in the query editor is:

```

CREATE OR REPLACE PROCEDURE flights_delayed_over_24h()
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT flight_id,
           flight_no,
           scheduled_departure,
           actual_departure,
           scheduled_arrival,
           actual_arrival,
           (actual_arrival - scheduled_arrival) AS delay_days
    FROM flights
    WHERE actual_arrival IS NOT NULL
        AND (actual_arrival - scheduled_arrival) > 1 -- 24 hours
    ORDER BY delay_days DESC;
END;
$$;

```

The 'Messages' tab shows the message 'CREATE PROCEDURE' and 'Query returned successfully in 43 msec.' at the bottom.

8. Create a function that counts the number of flights for each airline.

```

CREATE OR REPLACE FUNCTION count_flights_per_airline()
RETURNS TABLE (
    airline_id INT,
    airline_name VARCHAR,
    flights_count INT
)
LANGUAGE plpgsql
AS $$

BEGIN
    RETURN QUERY
    SELECT a.airline_id,
           a.airline_name,
           COUNT(f.flight_id) AS flights_count
    FROM airline a
    LEFT JOIN flights f ON a.airline_id = f.airline_id
    GROUP BY a.airline_id, a.airline_name
    ORDER BY flights_count DESC;
END;
$$;

```

Total rows: Query complete 00:00:00.051

✓ Query returned successfully in 51 msec. LF Ln 19, Col 4

9. Create a stored procedure to calculate the average ticket price for a specific flight.

```

CREATE OR REPLACE PROCEDURE average_ticket_price(p_flight_no VARCHAR)
LANGUAGE plpgsql
AS $$

BEGIN
    SELECT f.flight_no,
           AVG(b.price) AS avg_ticket_price
    FROM flights f
    JOIN booking_flight bf ON f.flight_id = bf.flight_id
    JOIN booking b ON bf.booking_id = b.booking_id
    WHERE f.flight_no = p_flight_no
    GROUP BY f.flight_no;
END;
$$;

```

Total rows: Query complete 00:00:00.039

✓ Query returned successfully in 39 msec. LF Ln 13, Col 4

10. Create a stored procedure to find the flight with the highest ticket price. The procedure should return the flight number, the departure and arrival airports, and

the ticket price for the most expensive flight.

The screenshot shows the pgAdmin 4 interface with a database connection to 'public.flights/data...'. The left sidebar displays the schema tree under 'public' schema, including 'Tables' like 'flights', 'booking', 'airport', and 'passenger'. The main query editor window contains the following SQL code:

```
1 CREATE OR REPLACE PROCEDURE most_expensive_flight()
2 LANGUAGE plpgsql
3 AS $$
4 BEGIN
5     SELECT f.flight_no,
6            dep.airport_name AS departure_airport,
7            arr.airport_name AS arrival_airport,
8            MAX(b.price) AS max_ticket_price
9     FROM flights f
10    JOIN booking_flight bf ON f.flight_id = bf.flight_id
11    JOIN booking b ON bf.booking_id = b.booking_id
12    JOIN airport dep ON f.departure_airport_id = dep.airport_id
13    JOIN airport arr ON f.arrival_airport_id = arr.airport_id
14    GROUP BY f.flight_no, dep.airport_name, arr.airport_name
15    ORDER BY max_ticket_price DESC
16    LIMIT 1;
17
18 $$;
```

The 'Data Output' tab shows the message: 'CREATE PROCEDURE'. Below it, 'Messages' shows 'Query returned successfully in 42 msec.' and 'Notifications' is empty. A status bar at the bottom right indicates 'Query complete 00:00:00.042' and 'Ln 18, Col 4'.