

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 PostgreSQL IDE interface. On the left, a tree view displays the database structure, including 'Local PostgreSQL', 'Databases (11)', and 'PostgreSQL 17'. The main editor window shows a SQL query to create a stored procedure named 'insert_new_flight'. The procedure has several parameters: 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), and p_actual_arrival (DATE). The procedure body contains an 'INSERT INTO flights' statement with columns flight_id, flight_no, scheduled_departure, and scheduled_arrival. The status bar at the bottom indicates 'Query complete 00:00:00.078' and 'Total rows:'. A green notification box in the bottom right corner states 'Query returned successfully in 78 msec.'.

```
1 CREATE OR REPLACE PROCEDURE insert_new_flight(  
2     p_flight_id INT,  
3     p_flight_no VARCHAR,  
4     p_scheduled_departure DATE,  
5     p_scheduled_arrival DATE,  
6     p_departure_airport_id INT,  
7     p_arrival_airport_id INT,  
8     p_departing_gate VARCHAR,  
9     p_arriving_gate VARCHAR,  
10    p_airline_id INT,  
11    p_status VARCHAR,  
12    p_actual_departure DATE,  
13    p_actual_arrival DATE  
14 )  
15 LANGUAGE plpgsql  
16 AS $$  
17 BEGIN  
18     INSERT INTO flights(  
19         flight_id,  
20         flight_no,  
21         scheduled_departure,  
22         scheduled_arrival,  
23     ) VALUES (  
24         p_flight_id,  
25         p_flight_no,  
26         p_scheduled_departure,  
27         p_scheduled_arrival,  
28     );  
29 END;  
30
```

CREATE PROCEDURE

Query returned successfully in 78 msec.

Total rows: Query complete 00:00:00.078

✓ Query returned successfully in 78 msec. ✕

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

The screenshot shows the PostgreSQL IDE interface. The left sidebar displays a tree view of databases, with 'database_subj' selected. The main editor window contains the following SQL code:

```
1 CREATE OR REPLACE PROCEDURE update_flight_status(  
2     p_flight_id INT,  
3     p_new_status VARCHAR  
4 )  
5 LANGUAGE plpgsql  
6 AS $$  
7 BEGIN  
8     IF NOT EXISTS (SELECT 1 FROM flights WHERE flight_id = p_flight_id) THEN  
9         RAISE EXCEPTION 'Flight with id % does not exist', p_flight_id;  
10    END IF;  
11  
12    UPDATE flights  
13    SET  
14        status = p_new_status,  
15        update_at = CURRENT_DATE  
16    WHERE flight_id = p_flight_id;  
17  
18 END;  
19 $$;
```

The 'Data Output' tab at the bottom shows the message: 'CREATE PROCEDURE' and 'Query returned successfully in 78 msec.' A green notification box at the bottom right confirms: '✓ Query returned successfully in 78 msec.'

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

The screenshot shows the PostgreSQL IDE interface. The left sidebar displays a tree view of databases, with 'database_subj' selected. The main editor window contains the following SQL code:

```
1 CREATE OR REPLACE PROCEDURE list_flights_from_airport(p_airport_id INT)  
2 LANGUAGE plpgsql  
3 AS $$  
4 BEGIN  
5     SELECT *  
6     FROM flights  
7     WHERE departure_airport_id = p_airport_id;  
8 END;  
9 $$;
```

The 'Data Output' tab at the bottom shows the message: 'CREATE PROCEDURE' and 'Query returned successfully in 42 msec.' A green notification box at the bottom right confirms: '✓ Query returned successfully in 42 msec.'

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

The screenshot shows the PostgreSQL IDE interface. On the left, the 'Databases (11)' list includes 'database_subj'. The main editor displays the following SQL code:

```

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

```

The 'Data Output' tab at the bottom shows the message: 'CREATE FUNCTION' and 'Query returned successfully in 41 msec.' A green status bar at the bottom right confirms: '✓ Query returned successfully in 41 msec. X'.

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

The screenshot shows the PostgreSQL IDE interface. On the left, the 'Schemas (1)' list includes 'public'. The main editor displays the following SQL code:

```

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

```

The 'Data Output' tab at the bottom shows the message: 'CREATE PROCEDURE' and 'Query returned successfully in 52 msec.' A green status bar at the bottom right confirms: '✓ Query returned successfully in 52 msec. X'.

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

The screenshot shows a PostgreSQL IDE interface. On the left, a schema browser displays the 'public' schema with various database objects. The main query editor contains the following SQL code:

```

1 CREATE OR REPLACE PROCEDURE passenger_flights()
2 LANGUAGE plpgsql
3 AS $$
4 BEGIN
5     SELECT p.passenger_id,
6           p.first_name,
7           p.last_name,
8           COUNT(bf.flight_id) AS flights_taken
9     FROM passengers p
10    JOIN booking b ON p.passenger_id = b.passenger_id
11    JOIN booking_flight bf ON b.booking_id = bf.booking_id
12   GROUP BY p.passenger_id, p.first_name, p.last_name
13  ORDER BY flights_taken DESC
14  LIMIT 1;
15 END;
16 $$;

```

Below the query editor, the 'Data Output' tab shows the message: 'CREATE PROCEDURE' and 'Query returned successfully in 44 msec.' A green status bar at the bottom right confirms: '✓ Query returned successfully in 44 msec.'

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

The screenshot shows the same PostgreSQL IDE interface. The query editor now contains the following SQL code:

```

1 CREATE OR REPLACE PROCEDURE flights_delayed_over_24h()
2 LANGUAGE plpgsql
3 AS $$
4 BEGIN
5     SELECT flight_id,
6           flight_no,
7           scheduled_departure,
8           actual_departure,
9           scheduled_arrival,
10          actual_arrival,
11          (actual_arrival - scheduled_arrival) AS delay_days
12     FROM flights
13    WHERE actual_arrival IS NOT NULL
14          AND (actual_arrival - scheduled_arrival) > 1 -- 24 hours
15  ORDER BY delay_days DESC;
16 END;
17 $$;

```

The 'Data Output' tab shows: 'CREATE PROCEDURE' and 'Query returned successfully in 43 msec.' The green status bar at the bottom right confirms: '✓ Query returned successfully in 43 msec.'

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

The screenshot shows a PostgreSQL IDE interface. On the left, a schema browser displays a database schema with various tables and functions. The main editor window contains a SQL query to create or replace a function named `count_flights_per_airline()`. The function is written in PL/pgSQL and returns a table with columns `airline_id` (INT), `airline_name` (VARCHAR), and `flights_count` (INT). The function body uses a `SELECT` statement with a `LEFT JOIN` between the `airline` table and the `flights` table, grouped by `airline_id` and `airline_name`, and ordered by `flights_count` in descending order. The status bar at the bottom indicates the query was completed successfully in 51 msec.

```

1 CREATE OR REPLACE FUNCTION count_flights_per_airline()
2 RETURNS TABLE (
3     airline_id INT,
4     airline_name VARCHAR,
5     flights_count INT
6 )
7 LANGUAGE plpgsql
8 AS $$
9 BEGIN
10     RETURN QUERY
11     SELECT a.airline_id,
12            a.airline_name,
13            COUNT(f.flight_id) AS flights_count
14     FROM airline a
15     LEFT JOIN flights f ON a.airline_id = f.airline_id
16     GROUP BY a.airline_id, a.airline_name
17     ORDER BY flights_count DESC;
18 END;
19 $$;

```

Query returned successfully in 51 msec.

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

The screenshot shows a PostgreSQL IDE interface. On the left, a schema browser displays a database schema with various tables and functions. The main editor window contains a SQL query to create or replace a procedure named `average_ticket_price(p_flight_no VARCHAR)`. The procedure is written in PL/pgSQL and uses a `SELECT` statement to calculate the average ticket price for a given flight number. The query joins the `flights` table with the `booking_flight` and `booking` tables, filtering by the specified flight number, and groups the results by flight number. The status bar at the bottom indicates the query was completed successfully in 39 msec.

```

1 CREATE OR REPLACE PROCEDURE average_ticket_price(p_flight_no VARCHAR)
2 LANGUAGE plpgsql
3 AS $$
4 BEGIN
5     SELECT f.flight_no,
6            AVG(b.price) AS avg_ticket_price
7     FROM flights f
8     JOIN booking_flight bf ON f.flight_id = bf.flight_id
9     JOIN booking b ON bf.booking_id = b.booking_id
10    WHERE f.flight_no = p_flight_no
11    GROUP BY f.flight_no;
12 END;
13 $$;

```

Query returned successfully in 39 msec.

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 a PostgreSQL IDE interface with the following components:

- Left Panel (Schemas):** A tree view showing the database schema. The 'public' schema is expanded, showing various objects like Aggregates, Collations, Domains, Functions, Materialized Views, Operators, Procedures, Sequences, Tables, Views, and Triggers. The 'flights' table is highlighted under the 'public' schema.
- Query Editor:** A text area containing a SQL query to create a procedure that finds the most expensive flight. The query is as follows:

```
1 CREATE OR REPLACE PROCEDURE most_expensive_flight()
2 LANGUAGE plpgsql
3 AS $$
4 BEGIN
5     SELECT f.flight_no,
6           dep.airport_name AS departure_airport,
7           arr.airport_name AS arrival_airport,
8           MAX(b.price) AS max_ticket_price
9     FROM flights f
10    JOIN booking_flight bf ON f.flight_id = bf.flight_id
11    JOIN booking b ON bf.booking_id = b.booking_id
12    JOIN airport dep ON f.departure_airport_id = dep.airport_id
13    JOIN airport arr ON f.arrival_airport_id = arr.airport_id
14   GROUP BY f.flight_no, dep.airport_name, arr.airport_name
15   ORDER BY max_ticket_price DESC
16   LIMIT 1;
17 END;
18 $$;
```
- Query History:** A tab showing the executed query.
- Scratch Pad:** A tab for taking notes.
- Data Output / Messages:** A section showing the execution results. It displays 'CREATE PROCEDURE' and a message: 'Query returned successfully in 42 msec.'
- Status Bar:** At the bottom, it shows 'Total rows: Query complete 00:00:00.042' and a green notification box stating '✓ Query returned successfully in 42 msec. X'.