

ABDYKAMAT ADILET LAB 10

STORED PROCEDURES and FUNCTION.

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

```
CREATE OR REPLACE PROCEDURE insert_new_flight(
```

```
    p_flight_id VARCHAR(50),  
    p_sch_departure_time TIMESTAMP,  
    p_sch_arrival_time TIMESTAMP,  
    p_departing_airport_id INT,  
    p_arriving_airport_id INT,  
    p_departing_gate VARCHAR(10),  
    p_arriving_gate VARCHAR(10),  
    p_airline_id INT,  
    p_country VARCHAR(100),  
    p_act_departure_time TIMESTAMP,  
    p_act_arrival_time TIMESTAMP  
)
```

```
LANGUAGE plpgsql
```

```
AS $$
```

```
BEGIN
```

```
    INSERT INTO flights(  
        flight_id,  
        sch_departure_time,  
        sch_arrival_time,  
        departing_airport_id,  
        arriving_airport_id,  
        departing_gate,  
        arriving_gate,
```

```
airline_id,  
act_departure_time,  
act_arrival_time,  
country,  
created_at,  
updated_at  
)
```

```
VALUES(  
p_flight_id,  
p_sch_departure_time,  
p_sch_arrival_time,  
p_departing_airport_id,  
p_arriving_airport_id,  
p_departing_gate,  
p_arriving_gate,  
p_airline_id,  
p_act_departure_time,  
p_act_arrival_time,  
p_country,  
NOW(),  
NOW()  
);
```

END;

\$\$;

FOR THE PROCEDURE TO WORK:

```
CALL insert_new_flight('FL123', '2024-01-20 10:00:00', '2024-01-20 12:00:00', 1, 2, 'A1', 'B2', 1, 'USA')  
;
```

The screenshot shows a MySQL Workbench interface. In the 'Query' tab, the following SQL code is displayed:

```

35     p_sch_departure_time,
36     p_sch_arrival_time,
37     p_departing_airport_id,
38     p_arriving_airport_id,
39     p_departing_gate,
40     p_arriving_gate,
41     p_airline_id,
42     p_act_departure_time,
43     p_act_arrival_time,
44     p_country,
45     NOW(),
46     NOW()
47   );
48 END;
49 $$;

```

Below the code, the status bar indicates "CREATE PROCEDURE". Under the tabs at the bottom, 'Messages' is selected, showing the message "Query returned successfully in 81 msec." with a green checkmark icon.

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

CREATE OR REPLACE PROCEDURE upupdate_flights_status(

```

    p_flight_id INT,
    p_status VARCHAR(50)
)
```

LANGUAGE plpgsql

AS \$\$

BEGIN

```

    UPDATE booking
    SET status = p_status
    WHERE flight_id = p_flight_id;
```

END;

\$\$;

FOR THE PROCEDURE TO WORK:

CALL upupdate_flights_status(5, 'Delayed');

```

1  CREATE OR REPLACE PROCEDURE update_flights_status(
2      p_flight_id VARCHAR(50),
3      p_status VARCHAR(50)
4  )
5  LANGUAGE plpgsql
6  AS $$*
7  BEGIN
8      UPDATE booking
9      SET status = p_status
10     WHERE flight_id = p_flight_id;
11 END;
12 $$;
13

```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 82 msec.

✓ Query returned successfully in 82 msec. ✕

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

CREATE OR REPLACE FUNCTION GetFlightsByDeparture(

p_departure_airport VARCHAR(5)
)

RETURNS TABLE (
 flight_number VARCHAR(10),
 departure_airport VARCHAR(5),
 arrival_airport VARCHAR(5),
 departure_time TIMESTAMP,
 arrival_time TIMESTAMP,
 status VARCHAR(20)

)

LANGUAGE plpgsql

AS \$\$

BEGIN

```

RETURN QUERY

SELECT *

FROM flights

WHERE departure_airport = p_departure_airport;

END;

$$;

SELECT * FROM GetFlightsByDeparture('SVO');

```

```

Query  Query History
6  departure_airport VARCHAR(5),
7   arrival_airport VARCHAR(5),
8   departure_time TIMESTAMP,
9   arrival_time TIMESTAMP,
10  status VARCHAR(20)
11 )
12 LANGUAGE plpgsql
13 AS $$*
14 BEGIN
15   RETURN QUERY
16   SELECT *
17   FROM flights
18   WHERE departure_airport = p_departure_airport;
19 END;
20 $$;

Data Output  Messages  Notifications
CREATE FUNCTION

Query returned successfully in 114 msec.

✓ Query returned successfully in 114 msec. ✕

```

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

```

CREATE OR REPLACE FUNCTION AvgDelayByArrivalAirport(
  p_airport_id INT
)
RETURNS INTERVAL
LANGUAGE plpgsql
AS $$

DECLARE
  avg_delay INTERVAL;

```

```

BEGIN

    SELECT AVG(act_arrival_time - sch_arrival_time)
        INTO avg_delay
        FROM flights
        WHERE arriving_airport_id = p_airport_id
        AND act_arrival_time IS NOT NULL
        AND sch_arrival_time IS NOT NULL;

    RETURN avg_delay;

END;
$$;
```

SELECT AvgDelayByArrivalAirport(55)

The screenshot shows a PostgreSQL query editor window. The title bar says "Query History". The main area contains the SQL code for creating a function:

```

CREATE OR REPLACE FUNCTION AvgDelayByArrivalAirport(
    p_airport_id VARCHAR
)
RETURNS INTERVAL
LANGUAGE plpgsql
AS $$
DECLARE
    avg_delay INTERVAL;
BEGIN
    SELECT AVG(act_arrival_time - sch_arrival_time)
    INTO avg_delay
    FROM flights
    WHERE arriving_airport_id = p_airport_id
    AND act_arrival_time IS NOT NULL
    AND sch_arrival_time IS NOT NULL;

```

Below the code, there are tabs for "Data Output", "Messages", and "Notifications". The "Messages" tab is selected, showing the message "CREATE FUNCTION". At the bottom of the editor, it says "Query returned successfully in 74 msec.".

A green success message box at the bottom right corner of the screen also says "Query returned successfully in 74 msec." with a checkmark icon.

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

```

CREATE OR REPLACE FUNCTION GetPassengersByFlight(
    p_flight_number VARCHAR(10)
```

```

)
RETURNS TABLE (
    passenger_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
)
LANGUAGE plpgsql
AS $$

BEGIN
    RETURN QUERY
        SELECT p.passenger_id, p.first_name, p.last_name
        FROM passengers p
        JOIN tickets t ON t.passenger_id = p.passenger_id
        WHERE t.flight_number = p_flight_number;
END;
$$;

SELECT * FROM GetPassengersByFlight('SU1005');

```

```

Query  Query History
1 CREATE OR REPLACE FUNCTION GetPassengersByFlight(
2     p_flight_number VARCHAR(10)
3 )
4 RETURNS TABLE (
5     passenger_id INT,
6     first_name TEXT,
7     last_name TEXT
8 )
9 LANGUAGE plpgsql
10 AS $$$
11 BEGIN
12     RETURN QUERY
13     SELECT p.passenger_id, p.first_name, p.last_name
14     FROM passengers p
15     JOIN tickets t ON t.passenger_id = p.passenger_id
Data Output  Messages  Notifications
CREATE FUNCTION
Query returned successfully in 98 msec.

```

✓ Query returned successfully in 98 msec. ✖

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

```
CREATE OR REPLACE FUNCTION GetTopPassenger()
```

```
RETURNS TABLE (
```

```
    passenger_id INT,
```

```
    first_name VARCHAR(50),
```

```
    last_name VARCHAR(50),
```

```
    flights_taken BIGINT
```

```
)
```

```
LANGUAGE plpgsql
```

```
AS $$
```

```
BEGIN
```

```
    RETURN QUERY
```

```
        SELECT p.passenger_id,
```

```
            p.first_name,
```

```
            p.last_name,
```

```
            COUNT(*) AS flights_taken
```

```
        FROM tickets t
```

```
        JOIN passengers p ON p.passenger_id = t.passenger_id
```

```
        GROUP BY p.passenger_id, p.first_name, p.last_name
```

```
        ORDER BY flights_taken DESC
```

```
        LIMIT 1;
```

```
    END;
```

```
$$;
```

```
SELECT GetTopPassenger()
```

The screenshot shows a PostgreSQL query editor interface. The top section is a code editor with syntax highlighting for SQL. The code defines a function named `get_passenger_with_most_flights` that returns the passenger with the most flights taken. The function uses a temporary table `#t` to store intermediate results and a final query to select the top passenger. The bottom section shows the results of running the function, which returned successfully in 109 msec.

```

Query History
1  CREATE OR REPLACE FUNCTION get_passenger_with_most_flights()
2  (
3      passenger_name VARCHAR(50),
4      flights_taken INT
5  )
6  LANGUAGE plpgsql
7  AS $$
8  BEGIN
9      RETURN QUERY
10     SELECT p.passenger_id, p.first_name, p.last_name, COUNT(*) AS flights_taken
11     FROM tickets t
12     JOIN passengers p ON p.passenger_id = t.passenger_id
13     GROUP BY p.passenger_id, p.first_name, p.last_name
14     ORDER BY flights_taken DESC
15     LIMIT 1;
16
17 END;
18
19 $$;
Data Output Messages Notifications
CREATE FUNCTION
Query returned successfully in 109 msec.

```

✓ Query returned successfully in 109 msec. ✕

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

CREATE OR REPLACE FUNCTION GetFlightsDelayed24h()

RETURNS TABLE (

```

flight_id INT,
delay_interval INTERVAL,
delay_minutes NUMERIC
)
```

LANGUAGE plpgsql

AS \$\$

BEGIN

RETURN QUERY

SELECT

```

f.flight_id,
(f.act_arrival_time - f.sch_arrival_time) AS delay_interval,
EXTRACT(EPOCH FROM (f.act_arrival_time - f.sch_arrival_time)) / 60 AS delay_minutes
FROM flights f
```

```

WHERE f.act_arrival_time IS NOT NULL
AND f.sch_arrival_time IS NOT NULL
AND (EXTRACT(EPOCH FROM (f.act_arrival_time - f.sch_arrival_time)) / 60) > 1440;
END;
$$;
```

SELECT * FROM GetFlightsDelayed24h();

```

Query  Query History
1 v CREATE OR REPLACE FUNCTION GetFlightsDelayed24h()
2 RETURNS TABLE (
3   flight_id INT,
4   delay_interval INTERVAL,
5   delay_minutes BIGINT
6 )
7 LANGUAGE plpgsql
8 AS $$$
9 BEGIN
10   RETURN QUERY
11   SELECT
12     f.flight_id,
13     (f.act_arrival_time - f.sch_arrival_time) AS delay_interval,
14     delay_minutes
15 END
Data Output  Messages  Notifications
CREATE FUNCTION

Query returned successfully in 92 msec.
```

✓ Query returned successfully in 92 msec. X

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

```
CREATE OR REPLACE FUNCTION CountFlightsByAirline(
```

```
    p_airline_id INT
```

```
)
```

```
RETURNS BIGINT
```

```
LANGUAGE plpgsql
```

```
AS $$
```

```
DECLARE
```

```
    cnt BIGINT;
```

```
BEGIN
```

```
    SELECT COUNT(flight_id)
```

```

INTO cnt

FROM flights

WHERE airline_id = p_airline_id;

RETURN cnt;

END;

$$;

```

SELECT CountFlightsByAirline(1);

```

Open in a new tab history
11 BEGIN
12   SELECT COUNT(flight_id)
13   INTO cnt
14   FROM flights
15   WHERE airline_id = p_airline_id;
16
17   RETURN cnt;
18 END;
19 $$;
20
21 SELECT CountFlightsByAirline(1);
22
23
24
Data Output Messages Notifications
CREATE FUNCTION
Query returned successfully in 100 msec.

✓ Query returned successfully in 100 msec. ✗

```

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

```

CREATE OR REPLACE FUNCTION AvgTicketPriceByFlight(
    p_flight_number VARCHAR(50)
)
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$

DECLARE

```

```

avg_price NUMERIC;

BEGIN

    SELECT AVG(price)

        INTO avg_price

        FROM tickets

        WHERE flight_number = p_flight_number;

    RETURN avg_price;

END;
$$;
```

SELECT AvgTicketPriceByFlight('SU1005');

```

Query  Query History
1 ✓ CREATE OR REPLACE FUNCTION AvgTicketPriceByFlight(
2     p_flight_id INT
3 )
4 RETURNS NUMERIC
5 LANGUAGE plpgsql
6 AS $$$
7 DECLARE
8     avg_price NUMERIC;
9 ✓ BEGIN
10    SELECT AVG(price)
11        INTO avg_price
12        FROM tickets
13        WHERE flight_id = p_flight_id;
14
15    RETURN avg_price;
Data Output  Messages  Notifications
CREATE FUNCTION

Query returned successfully in 87 msec.
```

✓ Query returned successfully in 87 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.

CREATE OR REPLACE FUNCTION GetMostExpensiveFlight()

RETURNS TABLE (

flight_id INT,

```

departing_airport_id VARCHAR(50),
arriving_airport_id VARCHAR(50),
price NUMERIC
)
LANGUAGE plpgsql
AS $$

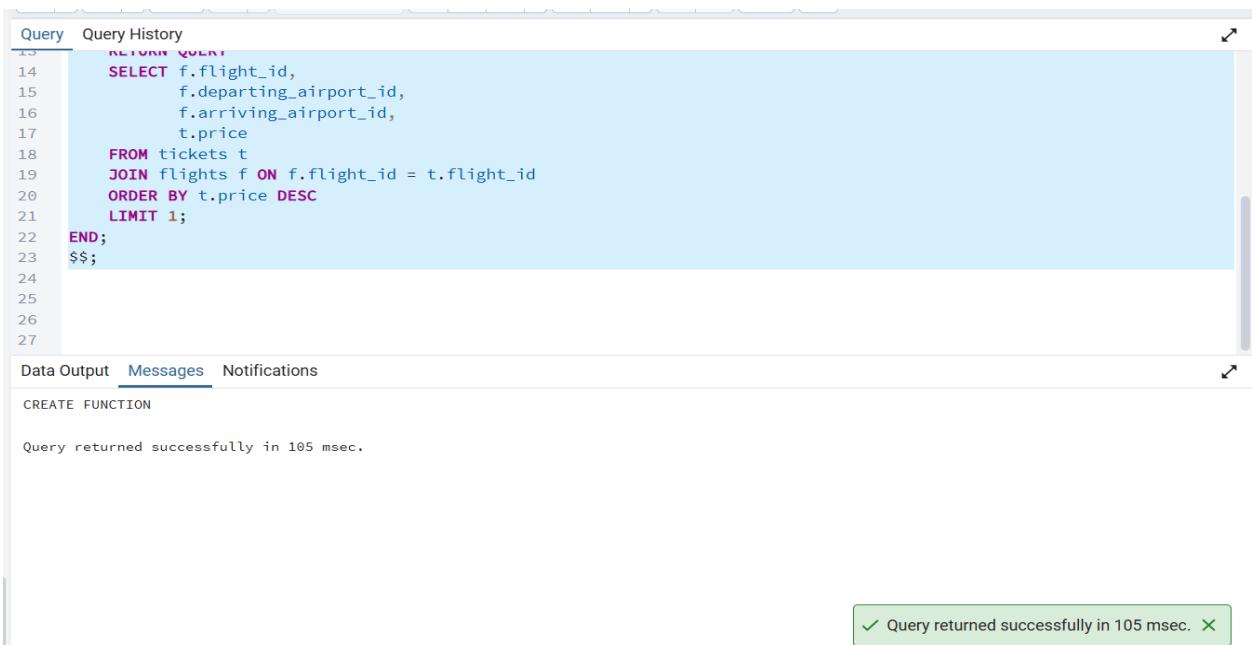
BEGIN

    RETURN QUERY

    SELECT f.flight_id,
           f.departing_airport_id,
           f.arriving_airport_id,
           t.price
      FROM tickets t
     JOIN flights f ON f.flight_id = t.flight_id
    ORDER BY t.price DESC
   LIMIT 1;

END;
$$;

```



The screenshot shows a PostgreSQL query editor window. The top section contains the SQL code for creating a function named 'expensiveFlight'. The bottom section shows the results of the query execution, indicating a successful return in 105 msec.

```

Query  Query History
13      RETURN QUERY;
14      SELECT f.flight_id,
15          f.departing_airport_id,
16          f.arriving_airport_id,
17          t.price
18      FROM tickets t
19      JOIN flights f ON f.flight_id = t.flight_id
20      ORDER BY t.price DESC
21      LIMIT 1;
22
23 END;
24
25
26
27
Data Output  Messages  Notifications
CREATE FUNCTION
Query returned successfully in 105 msec.

```

✓ Query returned successfully in 105 msec. ✘

