

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 SQL IDE interface with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query with line numbers 35 through 49. The query is a PL/SQL block that defines a procedure with several parameters and uses the NOW() function. Below the query editor, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the text 'CREATE PROCEDURE' and 'Query returned successfully in 81 msec.'. At the bottom right of the IDE, there is a green notification box with a checkmark and the text 'Query returned successfully in 81 msec. X'.

```
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  $$;
```

CREATE PROCEDURE

Query returned successfully in 81 msec.

✓ Query returned successfully in 81 msec. X

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

```
CREATE OR REPLACE PROCEDURE update_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 update_flights_status(5, 'Delayed');
```

QueryQuery History

1

2

3

4

5

6

7

8

9

10

11

12

13

```
CREATE OR REPLACE PROCEDURE update_flights_status(  
    p_flight_id VARCHAR(50),  
    p_status VARCHAR(50)  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    UPDATE booking  
    SET status = p_status  
    WHERE flight_id = p_flight_id;  
END;  
$$;
```

Data OutputMessagesNotifications

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

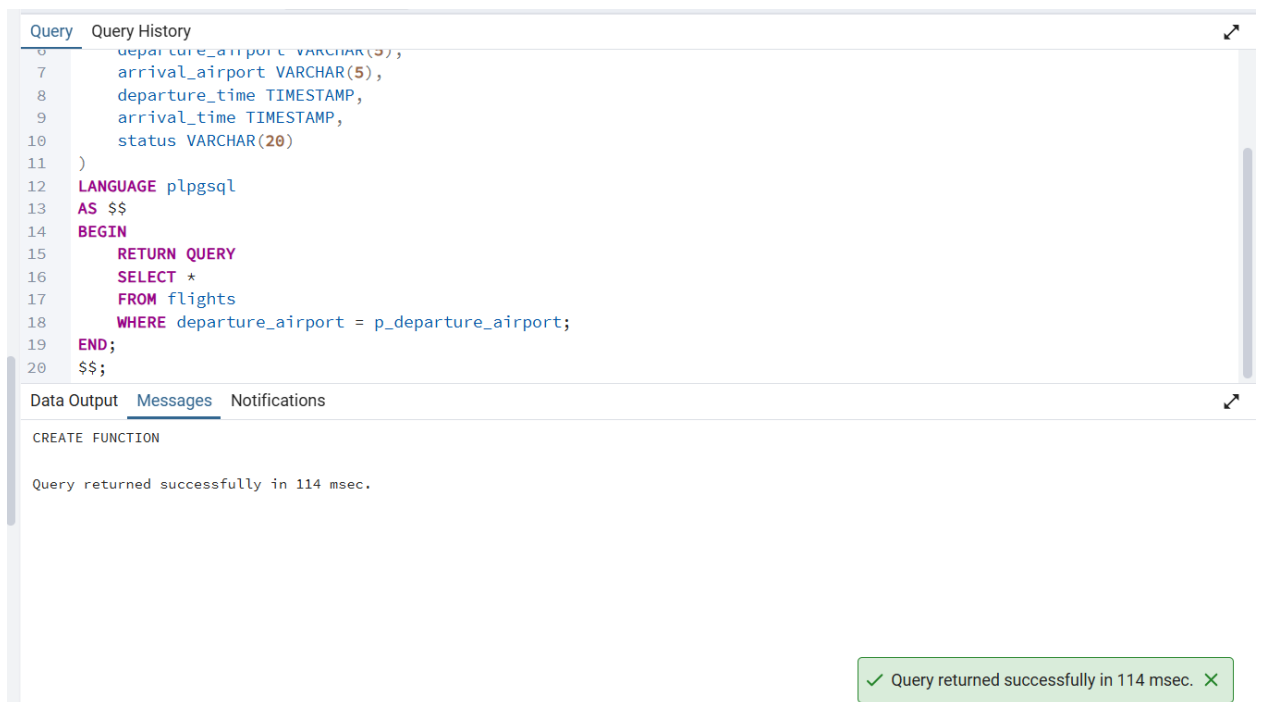
WHERE departure_airport = p_departure_airport;

END;

$$;

SELECT * FROM GetFlightsByDeparture('SVO');

```



The screenshot shows a SQL IDE interface. The top pane displays a function definition for `GetFlightsByDeparture` in PL/SQL. The function takes a departure airport as input and returns all flights from that airport. The bottom pane shows the execution results, indicating that the query was successful and returned data in 114 milliseconds.

```

Query    Query History
6      departure_airport VARCHAR(3),
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 SQL IDE interface with a query editor and a messages pane. The query editor contains the following SQL code:

```

1 CREATE OR REPLACE FUNCTION AvgDelayByArrivalAirport(
2   p_airport_id VARCHAR
3 )
4 RETURNS INTERVAL
5 LANGUAGE plpgsql
6 AS $$
7 DECLARE
8   avg_delay INTERVAL;
9 BEGIN
10  SELECT AVG(act_arrival_time - sch_arrival_time)
11  INTO avg_delay
12  FROM flights
13  WHERE arriving_airport_id = p_airport_id
14        AND act_arrival_time IS NOT NULL
15        AND sch_arrival_time IS NOT NULL;

```

The messages pane at the bottom shows the following output:

```

CREATE FUNCTION
Query returned successfully in 74 msec.

```

A green notification box at the bottom right of the IDE states: "Query returned successfully in 74 msec."

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()
```


QueryQuery History

```

5      last_name VARCHAR(30),
6      flights_taken INT
7  )
8  LANGUAGE plpgsql
9  AS $$
10 BEGIN
11     RETURN QUERY
12     SELECT p.passenger_id, p.first_name, p.last_name, COUNT(*) AS flights_taken
13     FROM tickets t
14     JOIN passengers p ON p.passenger_id = t.passenger_id
15     GROUP BY p.passenger_id, p.first_name, p.last_name
16     ORDER BY flights_taken DESC
17     LIMIT 1;
18 END;
19 $$;

```

Data OutputMessagesNotifications

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();

```

The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following PL/SQL code:

```

1 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

```

The results pane shows the output of the function call:

```

CREATE FUNCTION

Query returned successfully in 92 msec.

```

A green notification box at the bottom right of the IDE states: "✓ Query returned successfully in 92 msec. ✕"

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);

The screenshot shows a SQL IDE interface. The top pane displays a SQL script with line numbers 11 to 24. The script defines a function `CountFlightsByAirline` and then calls it with the argument `1`. The bottom pane shows the execution results, including a message: "Query returned successfully in 100 msec." A green notification box at the bottom right also displays this message.

```
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
```

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');

```

The screenshot shows a SQL IDE interface with a 'Query' tab. The SQL code is as follows:

```

1 CREATE OR REPLACE FUNCTION AvgTicketPriceByFlight(
2   p_flight_id INT
3 )
4 RETURNS NUMERIC
5 LANGUAGE plpgsql
6 AS $$
7 DECLARE
8   avg_price NUMERIC;
9 BEGIN
10  SELECT AVG(price)
11  INTO avg_price
12  FROM tickets
13  WHERE flight_id = p_flight_id;
14
15  RETURN avg_price;
16
17 END;
18
19 $$;

```

Below the code editor, the 'Messages' tab is active, displaying the message: 'Query returned successfully in 87 msec.' A green status bar at the bottom right also indicates: '✓ 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 SQL IDE interface. The top pane, titled 'Query', displays the SQL code from the previous block. The bottom pane, titled 'Data Output', shows the message 'Query returned successfully in 105 msec.' A green notification box at the bottom right of the IDE also displays this message.

```
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  END;
23  $$;
24
25
26
27

Data Output    Messages    Notifications
CREATE FUNCTION
Query returned successfully in 105 msec.
```

✓ Query returned successfully in 105 msec. ✕

