

Fundamentals of Computer Programming

Lecture 3: Advanced SELECT queries

Zied Bouyahya

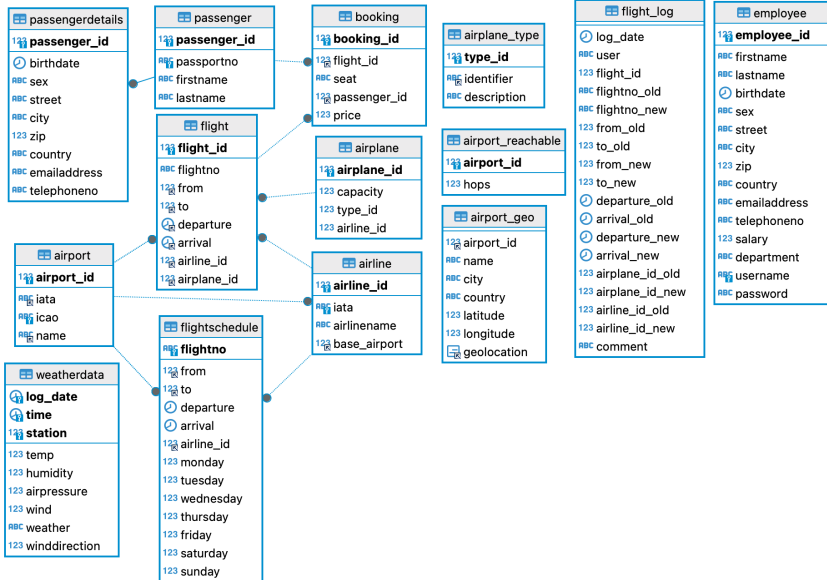
Ecole Centrale de Lyon, Bachelor of Science in data science for responsible business



Outline

- SQL nested queries
- SQL EXIST
- The CASE statement
- Window functions
- Common Table Expressions
- REGEXP_LIKE

The airportdb (ffdb) database



SQL nested queries

Definition

In SQL, a sub-query (or nested queries) consists in executing a query within another one. It is generally used within a WHERE or HAVING clauses.

Syntax

```
SELECT *  
FROM table_name  
WHERE column_name = (  
    SELECT value  
    FROM another_table  
    LIMIT 1  
)
```

This query returns a single result (row).

Remark

It is possible to use other binary (relational) operators such as <, >, =, etc.

SQL nested queries

Nested SELECT returning a column

- A nested SQL query may also be used to return an entire column.
- In this case, the outer query uses the command IN to filter the rows containing the values returned by the inner query.

Syntax

```
SELECT *  
FROM table  
WHERE a_certain_column IN (  
    SELECT one_column  
    FROM another_table  
    WHERE another_key = another_value  
)
```

SQL nested queries

Examples

Find all passengers who have made a booking on a flight that departs from a specific airport.

```
SELECT p.passenger_id, p.firstname, p.lastname, p.passportno
FROM passenger p
WHERE p.passenger_id IN (
    SELECT DISTINCT b.passenger_id
    FROM booking b
    JOIN flight f ON b.flight_id = f.flight_id
    WHERE f.from = 1
);
```

Explain?

1. The nested SELECT retrieves distinct passenger_id values from the booking table for flights departing from the specified airport (airport_id 1).
2. The outer SELECT then retrieves details (id, firstname, lastname, passport number) of passengers whose passenger_id is present in the result of the nested SELECT.

SQL nested queries

Examples (Ctd.)

Find airlines that have more than 10 flights.

```
SELECT airline_id, airlinename, total_flights
FROM (
    SELECT a.airline_id, a.airlinename, COUNT(f.flight_id) AS total_flights
    FROM airline a
    JOIN flight f ON a.airline_id = f.airline_id
    GROUP BY a.airline_id, a.airlinename
    HAVING total_flights > 10
) AS airlines_with_more_than_10_flights;
```

Explain?

1. The inner SELECT statement counts the number of flights for each airline.
2. The HAVING clause in the inner SELECT filters out the results, retaining only those airlines with more than 10 flights.
3. The outer SELECT statement then retrieves the airline information and the total number of flights from the filtered results of the inner SELECT. SELECT.

SQL nested queries

Examples (Ctd.)

Passengers who flew from Alessandria airport.

```
SELECT passenger_id, firstname, lastname
FROM passenger
WHERE passenger_id IN (
    SELECT passenger_id
    FROM booking
    WHERE flight_id IN (
        SELECT flight_id
        FROM flight
        WHERE `from` = (
            SELECT airport_id
            FROM airport
            WHERE name = 'ALESSANDRIA'
        )
    )
);
```


SQL EXIST

Definition

The EXISTS keyword in SQL is used to check for the existence of rows in a subquery.

Example

```
SELECT *  
FROM passenger p  
WHERE EXISTS (  
    SELECT 1  
    FROM booking b  
    WHERE b.passenger_id = p.passenger_id  
);
```

SQL EXIST

Using NOT EXISTS

Find passengers who have not made any bookings

Example

```
SELECT *  
FROM passenger p  
WHERE NOT EXISTS (  
    SELECT 1  
    FROM booking b  
    WHERE b.passenger_id = p.passenger_id  
);
```

SQL EXIST

Correlated Subquery

Find flights that have bookings:

Example

```
SELECT flight_id, flightno
FROM flight f
WHERE EXISTS (
    SELECT 1
    FROM booking b
    WHERE b.flight_id = f.flight_id
);
```

This subquery checks if there exist any bookings for each flight (1 if true)

SQL EXIST

Using EXIST with aggregation

Find flights that have more than 20 bookings.

Example

```
SELECT flight_id, flightno
FROM flight f
WHERE EXISTS (
    SELECT 1
    FROM booking b
    WHERE b.flight_id = f.flight_id
    HAVING COUNT(*) > 20
);
```

The CASE statement

- The CASE statement in SQL is a conditional expression that allows you to perform conditional logic within a query.
- It is often used to create new columns based on certain conditions or to categorize data.

Syntax

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

...

ELSE default_result

END

- CASE: The keyword to start the CASE statement.
- WHEN condition THEN result: This is the first condition. If condition is true, the corresponding result1 will be returned.
- ELSE default_result: If none of the conditions is true, the ELSE clause provides a default result.
- END: The keyword to end the CASE statement

The CASE statement

Example

If a flight is less than or equal to 2 hours (120 minutes), it's labeled as a 'Short Flight.' If it's between 2 and 4 hours (240 minutes), it's a 'Medium Flight.' Anything longer falls into the 'Long Flight' category.

```
SELECT flight_id, flightno, departure, arrival,
       CASE
         WHEN TIMESTAMPDIFF(MINUTE, departure, arrival) <= 120 THEN 'Short Flight'
         WHEN TIMESTAMPDIFF(MINUTE, departure, arrival) > 120
              AND TIMESTAMPDIFF(MINUTE, departure, arrival) <= 240
              THEN 'Medium Flight'
         ELSE 'Long Flight'
       END AS flight_category
FROM flight;
```

Window functions

Definition

- Window functions allows to perform calculations across a set of rows that are related to the current row.
- Operate on a "window" of rows and are often used in conjunction with the OVER clause

Example

Find the average price of flights for each airline.

```
SELECT flight_id, airline_id, price,  
       AVG(price) OVER (PARTITION BY airline_id) AS avg_price_per_airline,  
       AVG(price) OVER () AS overall_avg_price  
FROM flight;
```

The PARTITION BY clause in the OVER clause is used to define the window within which the AVG function operates. In this case, it calculates the average price for each airline separately. The OVER () clause without any partitioning calculates the overall average price for all rows.

Common Table Expressions (CTEs)

Definition

- Common Table Expressions (CTEs) are temporary result sets that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement.
- CTEs are defined using the WITH keyword. CTEs make complex queries more readable and allow you to break down a query into smaller, more manageable parts.

Syntax

```
WITH cte_name (c1, c2, ..., cN) AS (  
    -- CTE query definition  
    SELECT ...  
    FROM ...  
    WHERE ...  
)  
-- Main query using the CTE  
SELECT ...  
FROM cte_name;
```

- cte_name: The name assigned to the CTE.
- (c1, c2, ..., cN): Optional. Specifies the column names for the CTE.
- AS: Keyword used to define the CTE.
- – CTE query definition: The actual SQL query that defines the CTE.
- – Main query using the CTE: The main query that references the CTE.

CTE

Example

```
WITH HeathrowFlights AS (  
    SELECT  
        f.flight_id,  
        f.flightno,  
        f.departure,  
        f.arrival,  
        a.name AS departure_airport  
    FROM  
        flight f  
    JOIN  
        airport a ON f.from = a.airport_id  
    WHERE  
        a.name = 'Heathrow'  
)  
SELECT * FROM HeathrowFlights;
```

- The CTE (HeathrowFlights) selects relevant columns from the flight table and joins it with the airport table to get the departure airport's name.
- The WHERE clause filters the results to include only flights departing from Heathrow. The main query then selects all columns from the CTE.

REGEXP_LIKE

Example

```
SELECT *  
FROM airport  
WHERE REGEXP_LIKE(lower(name), '^ales', 'i');
```

Advantages

- Pattern Flexibility.
- Advanced Pattern Matching: Examples include matching patterns with variable lengths, specific character combinations, or complex conditions.
- Case-Insensitive Matching (option 'i').
- Negation.

Exercises

Exercise 1: Retrieve the flight details (flight_id, flightno, departure, arrival) for flights departing from 'Heathrow' airport.

Exercise 2: Find all passengers who have made bookings for flights that depart from 'Heathrow' airport.

Exercise 3: List the airlines with more than 1000 flights.

Exercise 4: Retrieve the passengers who have not made any bookings.

Exercise 5: Find flights that have at least 3 bookings.

Exercise 6: Categorize flights as 'Cheap', 'Medium', or 'Expensive' based on their average price (≤ 300 , $301-400$, > 400).

Exercise 7: Identify flights with a duration higher than the average duration for their respective airlines.

Exercise 8: List the airports whose names start with 'Ales' (case-insensitive).

Exercise 9: Find passengers who have made bookings on flights departing from 'Heathrow' or 'Vienna'.

Exercise 10: Retrieve flights with the number of bookings for each flight.

Exercises (Ctd.)

Exercise 11: For each passenger, display their name and the total number of flights they have booked.

Exercise 12: Identify flights with the maximum number of bookings.

Exercise 13: Retrieve the average price per airline for flights with more than 5 bookings.

Exercise 14: Find passengers who have made bookings for flights departing from airports whose name do not start with a vowel. (use `regexp_like`)

Exercise 15: Display the flight details for flights that do not have any bookings.

Exercise 16: Categorize flights as 'Popular' if they have more than 10 bookings; otherwise, label them as 'Less Popular'.

Exercises (Ctd.)

Exercise 17: List the airlines and the total number of flights they operate, along with the overall average price for all flights.

Exercise 18: Using a CTE, retrieve the flights departing from 'Heathrow' airport.

Exercise 19: Find passengers who have booked at least one flight with a price higher than 500.

Exercise 20: Retrieve the airports with names that do not start with 'Lond'.

