
SQL - Subqueries

Contents

- Subqueries
- Nested Subqueries
- Derived Subqueries
- `UNION`
- Common Table Expressions (CTE)

Subqueries

- Notice that, when making a `SELECT` query, it produce a table.
- We can use a `SELECT` query on that output
- We can use `WHERE IN`

Outer query

```
SELECT {column}
FROM {table}
WHERE {column} IN (
```

Inner query

```
{SELECT query}
```

```
)

{Additional statements}
```

Subqueries

- We can also use `JOIN`

```
SELECT {column}                                          Outer query
FROM {table}
JOIN                (
                        {SELECT query}    Inner query

                    ) {Inner_query_name}

ON {Inner_query_name}.{common_key} = {table}.{common_key}
{Additional statements}
```

Subqueries

- For example: Find the actors that played a role in film 2

```
SELECT *
FROM actor
WHERE actor_id IN
        (
         SELECT actor_id
         FROM film_actor
         WHERE film_id = 2
         )
```

This will run first.
What do we get from here?

Subqueries

- For example: Find the actors that played a role in film 2

```
SELECT *
FROM actor
JOIN(
        SELECT * FROM film_actor
        WHERE film_id = 2
        ) sub
ON actor.actor_id = sub.actor_id
```

This will run first.
What do we get from here?

Can you see the difference?

Subqueries

- 'Find the actors that played a role in film 2' is not very informative.
- We might want to see the actors that played a role in a film whose name we know
- We can do a subquery inside a subquery (inside a subquery…)

```
SELECT * FROM actor
WHERE actor_id IN
      (SELECT actor_id FROM film_actor
       WHERE film_id =
            (SELECT film_id FROM film
             WHERE title = 'DRAGON SQUAD')
      );
```

What do these queries return?

Derived Subqueries

- A derived table is a subquery nested within a `FROM` statement
- The `FROM` takes info from the output as if it was a regular table
- Subqueries have to get an alias
- Example: Average spending per customer

```
SELECT AVG(a) FROM
    (
    SELECT customer_id, SUM(amount) AS a
    FROM payment
    GROUP BY customer_id
    ) AS totals;
```

Subqueries - Practicals (Part I)

Go to the portal and complete the first practical:

Using Subqueries

Unions

- JOINs add data horizontally
- We might be interested in putting queries together vertically

```
   (
SELECT actor_id, first_name, last_name
FROM actor
WHERE first_name = 'CHRISTIAN'
   )
UNION
   (
SELECT actor_id, first_name, last_name
FROM actor
WHERE last_name = 'AKROYD'
   );
```

Can you find another way to do this?

Common Table Expressions

- Common Table Expressions (CTE) are (in a certain sense) a different version of subqueries.
- They establish temporary tables using `WITH`
- The syntax is:

```
WITH {new_table} AS ({SELECT query})
SELECT {column or aggregation}
FROM {new_table}
```

Common Table Expressions

- Average spending per customer

```
WITH total_amounts AS (
(SELECT customer_id, SUM(amount) AS a
    FROM payment
    GROUP BY customer_id)
)

SELECT AVG(a)
FROM total_amounts;
```

Common Table Expressions

- We can use as many WITH statements as we want

```
WITH
   table1 AS (SELECT * FROM rental),
   table2 AS (SELECT * FROM customer)

SELECT *
FROM table1
JOIN table2 ON table1.customer_id =
table2.customer_id;
```

Subqueries - Practicals (Part II)

Go to the portal and complete the second practical:
Difference between Subqueries and CTEs