

Subqueries and Table Joins

Introduction to SQL

Przemysław Rola

May 19, 2025

What is a Subquery?

- A **subquery** is an inner query nested within another query (outer or main query).
- The subquery's results serve as parameters for the outer query.
- Example:

```
1 SELECT * FROM address WHERE address_id =  
2 (SELECT address_id FROM customer WHERE  
3 first_name = 'Theresa' AND last_name = 'Watson');
```

- When using comparison operators (e.g., =), the subquery must return a **scalar** (single value) result.

Single-Row Subqueries

- Return **one row**.
- Used with comparison operators (e.g., =).
- Example:

```
1 SELECT * FROM address WHERE address_id =  
2 (SELECT address_id FROM customer WHERE  
3 first_name = 'Theresa' AND last_name = 'Watson');
```

- An error occurs if the subquery returns more than one row.

Multi-Row Subqueries

- Return **multiple rows**.
- Used with the IN operator.
- Example:

```
1 SELECT name FROM language WHERE language_id IN
2 (SELECT DISTINCT language_id FROM film);
```

- Safer than comparison operators when multiple results are possible.

Examples of Multi-Row Subqueries

- Example 1:

```
1 SELECT country FROM country WHERE country_id IN
2 (SELECT country_id FROM city WHERE city LIKE 'K%');
```

- Example 2 (Nested Subqueries):

```
1 SELECT CONCAT(first_name, ' ', last_name) AS customer
2 FROM customer WHERE customer_id IN
3 (SELECT customer_id FROM rental WHERE inventory_id =
4 (SELECT inventory_id FROM film WHERE title = 'Iron Moon'
   ));
```

Advanced Subqueries

- Complex Subqueries:

```
1 SELECT CONCAT(first_name, ' ', last_name) AS customer
2 FROM customer WHERE customer_id IN
3 (SELECT customer_id FROM payment GROUP BY customer_id
4  HAVING SUM(amount) > 0.75 * (SELECT SUM(amount) AS total
5  FROM payment GROUP BY customer_id
6  ORDER BY total DESC LIMIT 1));
```

- Subqueries with Multiple Columns:

```
1 SELECT title, rental_duration, rental_rate FROM film
2 WHERE (rental_duration, rental_rate) =
3 (SELECT MAX(rental_duration), MIN(rental_rate) FROM film
4  );
```

Subqueries Referencing Outer Tables

- Can reference tables from the outer query:

```
1 SELECT rental_id, staff_id FROM rental R
2 WHERE inventory_id IN
3 (SELECT inventory_id FROM inventory WHERE R.staff_id =
   store_id);
```

Advantages and Disadvantages of Subqueries

Advantages:

- Simple syntax, easy to construct and read.
- Can be used for data manipulation (insert, update, delete).

Disadvantages:

- Less readable when joining multiple tables.
- Limited to columns from a single table.
- Slower than table joins.

Use Case: Ideal for exceptional cases where joins are complex or when one query's result is needed for another.

Exercise Set 1: Subqueries

- 1 List the country where the city Bellevue is located.
- 2 List all phone numbers of customers from Poland.
- 3 List actors (first and last name in a column labeled actor) who starred in films in the `Children` category.
- 4 List titles of films with the most copies in store 1.
- 5 List customers (first and last name in a column labeled customer) whose total payments are at most half the average payment total.

Exercise 1.1: Solution

List the country where the city Bellevue is located.

```
1 SELECT country FROM country WHERE country_id =  
2 (SELECT country_id FROM city WHERE city = 'Bellevue');
```

Exercise 1.2: Solution

List all phone numbers of customers from Poland.

```
1 SELECT phone FROM address WHERE city_id IN  
2 (SELECT city_id FROM city WHERE country_id =  
3 (SELECT country_id FROM country WHERE country = 'Poland'));
```

Exercise 1.3: Solution

List actors who starred in films in the Children category.

```
1 SELECT DISTINCT CONCAT(first_name, ' ', last_name) AS actor
2 FROM actor WHERE actor_id IN
3 (SELECT actor_id FROM film_actor WHERE film_id IN
4 (SELECT film_id FROM film_category WHERE category_id =
5 (SELECT category_id FROM category WHERE name = 'Children'))))
;
```

Exercise 1.4: Solution

List titles of films with the most copies in store 1.

```
1 SELECT title
2 FROM film
3 WHERE film_id IN (
4     SELECT film_id
5     FROM inventory
6     WHERE store_id = 1
7     GROUP BY film_id
8     HAVING COUNT(*) = (
9         SELECT MAX(copy_count)
10        FROM (
11            SELECT COUNT(*) AS copy_count
12            FROM inventory
13            WHERE store_id = 1
14            GROUP BY film_id
15        ) AS counts
16    )
17 );
```

Exercise 1.5: Solution

List customers with total payments at most half the average.

```
1 SELECT CONCAT(first_name, ' ', last_name) AS customer
2 FROM customer WHERE customer_id IN
3 (SELECT customer_id FROM payment GROUP BY customer_id
4  HAVING SUM(amount) <= 0.5 * (SELECT AVG(total)
5   FROM (SELECT SUM(amount) AS total FROM payment
6    GROUP BY customer_id) AS sub));
```

What are Table Joins?

- Allow retrieval of data from **multiple tables**.
- Tables are linked using **primary and foreign keys**.
- Types of Joins:
 - Cross (CROSS JOIN)
 - Inner (INNER JOIN)
 - Outer:
 - Left (LEFT JOIN)
 - Right (RIGHT JOIN)
 - Full (FULL JOIN, not supported in MySQL)

Summary with SQL JOINS Visualization

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



Figure: Different types of the JOINS in SQL

Cross Join (CROSS JOIN)

- A Cartesian product – returns **all possible combinations** of rows.
- No key relationship required.
- Example:

```
1 SELECT * FROM category CROSS JOIN language;
```

- Caution: Large tables (e.g., 1000×1000 rows = 1,000,000 rows).

Inner Join (INNER JOIN)

- Returns rows with **matching key values**.
- Most commonly used join type.
- Example:

```
1 SELECT * FROM rental INNER JOIN inventory
2 ON rental.inventory_id = inventory.inventory_id;
```

- Alternative with WHERE:

```
1 SELECT * FROM rental, inventory
2 WHERE rental.inventory_id = inventory.inventory_id;
```

Inner Join – Multiple Tables

- Example with multiple tables:

```
1 SELECT CONCAT(C.first_name, ' ', C.last_name) AS  
   customer,  
2 F.title, R.rental_date, R.return_date  
3 FROM customer C INNER JOIN rental R USING(customer_id)  
4 INNER JOIN inventory I USING(inventory_id)  
5 INNER JOIN film F USING(film_id);
```

- Alternative with WHERE and sorting:

```
1 SELECT CONCAT(C.first_name, ' ', C.last_name) AS  
   customer,  
2 F.title, R.rental_date, R.return_date  
3 FROM customer C, rental R, inventory I, film F  
4 WHERE C.customer_id = R.customer_id AND  
5 R.inventory_id = I.inventory_id AND  
6 I.film_id = F.film_id  
7 ORDER BY R.rental_date, R.return_date;
```

Inner Join vs. Subqueries

- Subquery:

```
1 SELECT CONCAT(first_name, ' ', last_name) AS customer
2 FROM customer WHERE customer_id IN
3 (SELECT customer_id FROM rental WHERE inventory_id =
4 (SELECT inventory_id FROM film WHERE title = 'Iron Moon'
   ));
```

- Equivalent Join:

```
1 SELECT CONCAT(first_name, ' ', last_name) AS customer
2 FROM customer C, rental R, inventory I, film F
3 WHERE C.customer_id = R.customer_id AND
4 R.inventory_id = I.inventory_id AND
5 I.film_id = F.film_id AND F.title = 'Iron Moon';
```

Outer Joins (OUTER JOIN)

- Return matching rows and **non-matching rows** from one or both tables.
- Types:
 - LEFT JOIN: All rows from the left table.
 - RIGHT JOIN: All rows from the right table.
 - FULL JOIN: All rows from both tables (not supported in MySQL).
- Example (LEFT JOIN):

```
1 SELECT * FROM film F LEFT OUTER JOIN film_actor FA
2 ON F.film_id = FA.film_id;
```

Outer Joins – Example

- Example (RIGHT JOIN):

```
1 SELECT * FROM film RIGHT OUTER JOIN language
2 USING(language_id);
```

- Full Join in MySQL (using UNION):

```
1 (SELECT * FROM film LEFT JOIN language USING(language_id
   ))
2 UNION
3 (SELECT * FROM film RIGHT JOIN language USING(
   language_id));
```

Union (UNION)

- UNION [DISTINCT]: Removes duplicates.
- UNION ALL: Includes all rows.
- Example:

```
1 SELECT actor_id, first_name, last_name FROM actor
2 UNION
3 SELECT customer_id, first_name, last_name FROM customer;
```

- With sorting:

```
1 (SELECT actor_id AS id, first_name AS first, last_name
   AS last FROM actor)
2 UNION
3 (SELECT customer_id, first_name, last_name FROM customer
   )
4 ORDER BY last, first;
```

FULL OUTER JOIN

- **Definition:** Returns all records from both tables, with NULLs for non-matching rows.
- **Use Case:** Combine rows from two tables, including all rows regardless of matches.
- **Syntax:**

```
1 SELECT a.column1, b.column2
2 FROM tableA a
3 FULL OUTER JOIN tableB b ON a.common_column = b.
   common_column;
```

- **Result:** Includes matching rows, non-matching rows from the first table (NULLs for second), and non-matching rows from the second table (NULLs for first).
- **Note:** Not natively supported in MySQL; use LEFT JOIN + RIGHT JOIN with UNION.

UNION – Detailed

- **Definition:** Combines result sets of two or more SELECT statements, removing duplicates (UNION) or keeping all rows (UNION ALL).
- **Use Case:** Aggregate similar data from multiple queries with compatible structures.
- **Syntax:**

```
1 SELECT column1, column2 FROM tableA
2 UNION
3 SELECT column1, column2 FROM tableB;
```

- **Result:** Unique rows (UNION) or all rows including duplicates (UNION ALL).

FULL OUTER JOIN vs. UNION – Example

- **Table A::**

ID	Name
1	Alice
2	Bob

- **Table B::**

ID	Name
2	Bob
3	Charlie

- **FULL OUTER JOIN Result:**

ID	Name	ID	Name
1	Alice	NULL	NULL
2	Bob	2	Bob
NULL	NULL	3	Charlie

- **UNION Result:**

ID	Name
1	Alice
2	Bob
3	Charlie

Example: Joining Multiple Tables

- Query:

```
1 SELECT F.film_id, F.title, I.store_id,  
2 CONCAT(C.first_name, ' ', C.last_name) AS customer,  
3 R.rental_date, R.return_date  
4 FROM film F LEFT JOIN inventory I USING(film_id)  
5 JOIN rental R USING(inventory_id)  
6 JOIN customer C USING(customer_id)  
7 ORDER BY 2, 3, 5, 6;
```

- Notes:

- JOIN, INNER JOIN, CROSS JOIN are equivalent without conditions.
- Joins are **more efficient** than subqueries.

Summary

- **Subqueries:** Simple but slower, used in exceptional cases.
- **Joins:** More efficient, support multiple table operations.
- Key join types: Cross, Inner, Outer.
- UNION combines query results.
- Choose based on data structure and required output.

Exercise Set 2: Joins

- 1 List all possible combinations of actor names (`actor`) and category names (`category`), sorted by actor's last name, first name, and category name.
- 2 List all possible combinations of category names (`type`) and film ratings (`category`), sorted by category name and rating.
- 3 List film titles (`film`), category names (`type`), and ratings (`category`), sorted by category and title.
- 4 List all customers of store 1 served by staff member 2.

Exercise 2.1: Solution

List all combinations of actor names and category names.

```
1 SELECT CONCAT(A.first_name, ' ', A.last_name) AS actor,  
2 C.name AS category FROM actor A CROSS JOIN category C  
3 ORDER BY A.last_name, A.first_name, C.name;
```

Exercise 2.2: Solution

List all combinations of category names and film ratings.

```
1 SELECT C.name AS type, F.rating AS category
2 FROM category C CROSS JOIN film F
3 GROUP BY type, category ORDER BY type, category;
```

Exercise 2.3: Solution

List film titles, category names, and ratings.

```
1 SELECT F.title AS film, C.name AS type, F.rating AS category
2 FROM film F INNER JOIN film_category CF USING(film_id)
3 INNER JOIN category C USING(category_id)
4 ORDER BY type, film;
```


Exercise 2.4: Solution

List customers of store 1 served by staff member 2.

```
1 SELECT DISTINCT C.* FROM customer C
2 INNER JOIN rental R USING(customer_id)
3 WHERE C.store_id = 1 AND R.staff_id = 2;
```

Exercise Set 3: Joins and UNION

- 1 List film ID, title, and total copies across both stores (`id`, `film`, `n`). Films with no copies should have `n=0`.
- 2 List all payment dates and amounts, along with corresponding rental ID, rental date, and return date (`payment_date`, `amount`, `id`, `rental_date`, `return_date`).
- 3 List all pairs of IDs from `film_actor` and `film_category` (including duplicates) in columns `film` and `actor_category`, sorted by film ID and actor/category ID.
- 4 Assign each country the total payments made by its customers, sorted in descending order.
- 5 For each customer, list the number of rented films per category (`customer`, `category`, `n`), sorted by last name, first name, and number of rentals (descending).

Exercise 3.1: Solution

List film ID, title, and total copies across both stores.

```
1 SELECT F.film_id AS id, F.title AS film, COUNT(I.film_id) AS  
   n  
2 FROM film F LEFT JOIN inventory I USING(film_id)  
3 GROUP BY film_id;
```

Exercise 3.2: Solution

List payment dates, amounts, and corresponding rental details.

```
1 SELECT P.payment_date AS payment_date, P.amount AS amount,
2 R.rental_id AS id, R.rental_date AS rental_date,
3 R.return_date AS return_date
4 FROM payment P LEFT JOIN rental R USING(rental_id)
5 ORDER BY payment_date;
```

Exercise 3.3: Solution

List pairs of IDs from film_actor and film_category.

```
1 (SELECT FA.film_id AS film, FA.actor_id AS actor_category
2 FROM film_actor FA)
3 UNION ALL
4 (SELECT FC.film_id, FC.category_id FROM film_category FC)
5 ORDER BY film, actor_category;
```

Exercise 3.4: Solution

Assign each country the total payments by its customers.

```
1 SELECT C.country AS country, SUM(P.amount) AS total
2 FROM payment P INNER JOIN customer USING(customer_id)
3 INNER JOIN address USING(address_id) INNER JOIN city USING(
   city_id)
4 INNER JOIN country C USING(country_id)
5 GROUP BY country ORDER BY total DESC;
```

Exercise 3.5: Solution

List the number of rented films per category for each customer.

```
1 SELECT CONCAT(C.first_name, ' ', C.last_name) AS customer,
2 CA.name AS category, COUNT(R.rental_id) AS n
3 FROM customer C INNER JOIN rental R USING(customer_id)
4 INNER JOIN inventory I USING(inventory_id) INNER JOIN film F
   USING(film_id)
5 INNER JOIN film_category FC USING(film_id) INNER JOIN
   category CA USING(category_id)
6 GROUP BY customer, category
7 ORDER BY C.last_name, C.first_name, n DESC;
```