Here are several **subquery** examples that illustrate their versatility in SQL:

---

1. Subquery in SELECT

Retrieve the total number of orders for each user along with their details.

```
SELECT
    u.user_id,
    u.first_name,
    u.last_name,
    (SELECT COUNT(*) FROM orders o WHERE o.user_id = u.user_id) AS total_orders
FROM
    users u;
```

---

2. Subquery in WHERE

Find all products with a price higher than the average price of all products.

```
SELECT
    product_name,
    price
FROM
    products
WHERE
    price > (SELECT AVG(price) FROM products);
```

---

3. Subquery in FROM

List the average price of products for each stock range by grouping products.

```
SELECT
    stock_range,
    AVG(price) AS avg_price
FROM
    (SELECT
         CASE
             WHEN stock < 50 THEN 'Low Stock'
```

```
            WHEN stock BETWEEN 50 AND 100 THEN 'Medium Stock'
            ELSE 'High Stock'
        END AS stock_range,
        price
    FROM
        products) AS stock_groups
GROUP BY
    stock_range;
```

---

4. Subquery in HAVING

Find categories where the total number of films exceeds 20.

```
SELECT
    category_id,
    COUNT(*) AS total_films
FROM
    film_category
GROUP BY
    category_id
HAVING
    COUNT(*) > (SELECT AVG(total) FROM (SELECT COUNT(*) AS total FROM film_category
GROUP BY category_id) sub);
```

---

5. Subquery with JOIN

List all customers who have rented films belonging to the "Comedy" category.

```
SELECT
    c.first_name,
    c.last_name
FROM
    customers c
JOIN
    rentals r ON c.customer_id = r.customer_id
WHERE
    r.film_id IN (SELECT fc.film_id
                  FROM film_category fc
                  JOIN categories cat ON fc.category_id = cat.category_id
                  WHERE cat.name = 'Comedy');
```

---

6. Correlated Subquery

Find all customers who have spent more than the average total amount paid by all customers.

```
SELECT
    c.customer_id,
    c.first_name,
    c.last_name
FROM
    customers c
WHERE
    (SELECT SUM(p.amount)
     FROM payments p
     WHERE p.customer_id = c.customer_id) >
    (SELECT AVG(total_amount)
     FROM (SELECT SUM(amount) AS total_amount FROM payments GROUP BY customer_id)
sub);
```

---

7. Subquery to Create a Derived Table

Create a list of films with their total rental counts, showing only films with rentals greater than 50.

```
SELECT
    f.film_id,
    f.title,
    rentals_count
FROM
    (SELECT
         r.film_id,
         COUNT(*) AS rentals_count
     FROM
         rentals r
     GROUP BY
         r.film_id) AS rental_counts
JOIN
    films f ON rental_counts.film_id = f.film_id
WHERE
    rentals_count > 50;
```

---

8. Subquery to Filter Rows

Get all products that have never been ordered.

```
SELECT
    p.product_id,
    p.product_name
FROM
    products p
WHERE
    p.product_id NOT IN (SELECT DISTINCT product_id FROM orders);
```

---

9. Subquery with EXISTS

Find customers who have made at least one order.

```
SELECT
    c.customer_id,
    c.first_name,
    c.last_name
FROM
    customers c
WHERE
    EXISTS (SELECT 1
            FROM orders o
            WHERE o.customer_id = c.customer_id);
```

---

10. Subquery with UPDATE

Increase the price of all products that are below the average price.

```
UPDATE
    products
SET
    price = price * 1.10
WHERE
    price < (SELECT AVG(price) FROM products);
```

Subqueries are powerful tools for breaking down complex logic into manageable parts. They are especially helpful for calculations, filtering, or transforming data before

applying the main query.

Would you like further clarification or examples for specific use cases?

Here's the example for finding **top customers based on payments** where their total payments exceed the average payment amount.

Query: Find Top Customers Based on Payments

```
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    (SELECT SUM(p.amount)
     FROM payments p
     WHERE p.customer_id = c.customer_id) AS total_payments
FROM
    customers c
WHERE
    (SELECT SUM(p.amount)
     FROM payments p
     WHERE p.customer_id = c.customer_id) >
    (SELECT AVG(total_amount)
     FROM (SELECT SUM(p.amount) AS total_amount
           FROM payments p
           GROUP BY p.customer_id) AS subquery)
ORDER BY
    total_payments DESC;
```

Explanation

1. **Inner Query for Total Payments (Per Customer):**

   ```
   SELECT SUM(p.amount)
   FROM payments p
   WHERE p.customer_id = c.customer_id
   ```

   - This calculates the total payments for each customer.

2. **Inner Query for Average Payment:**

```
SELECT AVG(total_amount)
FROM (SELECT SUM(p.amount) AS total_amount
      FROM payments p
      GROUP BY p.customer_id) AS subquery
```

- This calculates the average total payment across all customers.

3. **Filter Condition:**
   - The `WHERE` clause ensures only customers whose total payments exceed the average are selected.
4. **Sorting:**
   - The `ORDER BY total_payments DESC` lists the customers in descending order of their total payments.

Use Case 2: Customers Renting the Most Movies

List the top 5 customers who have rented the most movies.

```
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    (SELECT COUNT(*)
     FROM rentals r
     WHERE r.customer_id = c.customer_id) AS total_rentals
FROM
    customers c
ORDER BY
    total_rentals DESC
LIMIT 5;
```

Explanation

1. **Inner Query:**

```
SELECT COUNT(*)
FROM rentals r
WHERE r.customer_id = c.customer_id
```

- This counts the total number of rentals for each customer.

2. **Sorting & Limiting:**

- The `ORDER BY total_rentals DESC` ensures the top renters appear first.
- The `LIMIT 5` restricts the output to the top 5 customers.

Use Case 3: Most Popular Film in a Specific Category

Find the most rented film in the "Comedy" category.

```
SELECT
    f.film_id,
    f.title,
    (SELECT COUNT(*)
     FROM rentals r
     WHERE r.film_id = f.film_id) AS rental_count
FROM
    films f
WHERE
    f.film_id IN (SELECT fc.film_id
                  FROM film_category fc
                  JOIN categories c ON fc.category_id = c.category_id
                  WHERE c.name = 'Comedy')
ORDER BY
    rental_count DESC
LIMIT 1;
```

Explanation

1. **Category Filtering (Inner Query):**

```
SELECT fc.film_id
FROM film_category fc
```

```
JOIN categories c ON fc.category_id = c.category_id
WHERE c.name = 'Comedy'
```

- This retrieves all film IDs for the "Comedy" category.

2. **Counting Rentals:**

```
SELECT COUNT(*)
FROM rentals r
WHERE r.film_id = f.film_id
```

- This counts the number of rentals for each film.

3. **Sorting & Limiting:**

- The `ORDER BY rental_count DESC` ensures the most rented film is at the top.
- The `LIMIT 1` restricts the output to the single most rented film.

Use Case 4: Employees with Sales Above Average

Find employees whose total sales exceed the company's average sales.

```
SELECT
    s.staff_id,
    s.first_name,
    s.last_name,
    (SELECT SUM(p.amount)
     FROM payments p
     WHERE p.staff_id = s.staff_id) AS total_sales
FROM
    staff s
WHERE
    (SELECT SUM(p.amount)
     FROM payments p
     WHERE p.staff_id = s.staff_id) >
    (SELECT AVG(total_sales)
     FROM (SELECT SUM(p.amount) AS total_sales
           FROM payments p
           GROUP BY p.staff_id) AS subquery)
ORDER BY
    total_sales DESC;
```

Would you like to delve deeper into any of these use cases or explore other scenarios?