Optimising queries

Subqueries

Optimising queries

Introduction

A subquery, also known as a **nested query**, is a query embedded within another SQL statement, such as **`SELECT`**, **`WHERE`**, **`FROM`**, **`JOIN`**, **`INSERT`**, **`UPDATE`**, or **`DELETE`**.

Subqueries enable the **retrieval** of **data** from one or more tables **based** on the results of an **inner** query, and offer **alternative options** to JOINS, functions, and window functions.

Subqueries often come at the cost of **lower readability** and **lower performance**, so they should be used with great consideration.

Example dataset

We will be using the following `Water_samples` to **describe the purity of water** as a **score from 0 to 100**. The samples have been named and the analysis type that was used to determine that score. The cost of each type of analysis is provided in the `Analysis_costs` table.

`Water_samples` table

Sample_name	Purity	Analysis_type
Alpha	68	Basic
Bravo	75	Advanced
Charlie	52	Basic
Delta	89	Advanced
Echo	92	Basic

`Analysis_costs` table

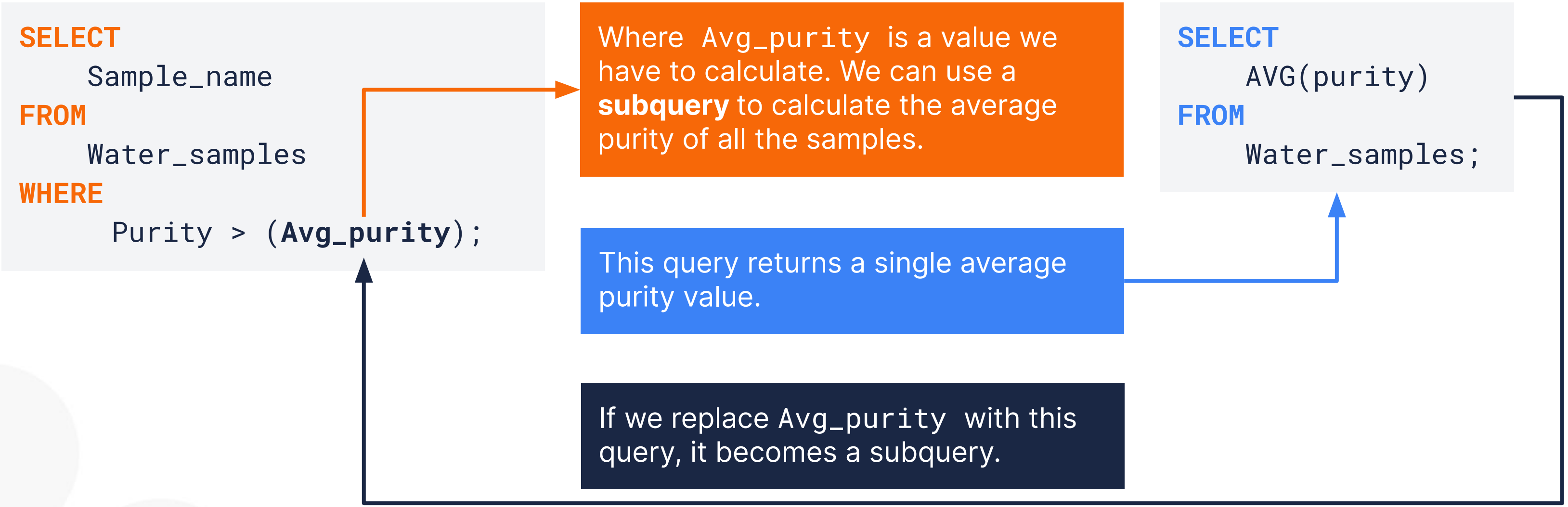
Analysis_type	Cost
Advanced	1000
Basic	50

`Sample_location` table

Sample_name	Location	Source
Alpha	Ziwa Maji	Lake
Bravo	Limpopo	River
Charlie	Mji	River
Delta	Nairobi	Tap
Echo	Jangwa	Tap

What are subqueries?

Suppose we want to retrieve the `Sample_name` in the `Water_samples` table that have **above-average purity**. To do this, we have to calculate the average purity, then use that value to filter with.



```
SELECT
    Sample_name
FROM
    Water_samples
WHERE
     Purity > (Avg_purity);
```

Where `Avg_purity` is a value we have to calculate. We can use a **subquery** to calculate the average purity of all the samples.

```
SELECT
    AVG(purity)
FROM
    Water_samples;
```

This query returns a single average purity value.

If we replace `Avg_purity` with this query, it becomes a subquery.

Optimising queries

What are subqueries?

> When there are **nested queries** present, the **inner** query is executed **first**, **then** the **outer** query is evaluated.

```
SELECT
    Sample_name,
    Purity
FROM
    Water_samples
WHERE
     Purity > (SELECT
                 AVG(Purity)
              FROM
                 Water_samples);
```

The **inner** query executes first and calculates a **single** average value of 75.2 that is used in the **outer** query as a filter.

```
SELECT
    Sample_name,
    Purity
FROM
    Water_samples
WHERE
     Purity >
```

Avg_purity
75.2

Optimising queries

Using subqueries

> **Subqueries** serve many different purposes and can be used in the SELECT, FROM, WHERE, JOIN, and HAVING clauses.

Water_samples table

Sample_name	Purity
Alpha	68
Bravo	75
Charlie	52
Delta	89
Echo	92

A subquery can be used in the WHERE clause to calculate the average purity of all the rows in the Water_samples table.

```sql
SELECT
    Sample_name,
    Purity
FROM
    Water_samples
WHERE
     Purity > (SELECT
                 AVG(Purity)
               FROM
                Water_samples);
```

Results set:

Sample_name	Purity
Delta	89
Echo	92

Subqueries in SELECT

> A subquery in the `SELECT` section of a query **always** has to return a **scalar value**.

```
SELECT
    Sample_name,
    Purity,
    (SELECT
        AVG(Purity)
     FROM
        Water_samples
     ) AS Avg_purity
FROM
    Water_samples;
```

Result set:

Sample_name	Purity	Avg_purity
Alpha	68	75.2
Bravo	75	75.2
Charlie	52	75.2
Delta	89	75.2
Echo	92	75.2

Each **row** in the `Avg_purity` column has the **same value** calculated by the subquery.

Most of the results in the following examples can be retrieved with simpler queries, so the aim is to illustrate subqueries that we may encounter in the future.

Optimising queries

Subqueries in SELECT

> A **correlated subquery** is a type of `SELECT` query that uses **values** from the **outer** query. The inner query executes referencing these value(s) and **returns** the **result** to the outer query. This happens **row by row**.

Suppose we want to calculate the total cost of each sample that was analysed. We can use a correlated subquery that looks up the cost of the analysis from the `Analysis_costs` table for each row.

```
SELECT
        ws_out.Sample_name,
        ws_out.Analysis_type,
        (SELECT
                Cost
          FROM
                Analysis_costs
          WHERE
                Analysis_type =
                ws_out.Analysis_type
          ) AS Cost
FROM
    Water_samples AS ws_out;
```

Begin with a list of samples from the `Water_samples` table.

For each sample: Look at its `Analysis_type`.

Go to the `Analysis_costs` table. Find the cost associated with that specific `Analysis_type`.

End with a list that shows each sample's name, its `Analysis_type`, and the associated cost.

Result set:

Sample_name	Analysis_type	Cost
Alpha	Basic	70.67
Bravo	Advanced	82
Charlie	Basic	70.67
Delta	Advanced	82
Echo	Basic	70.67

Optimising queries

Subqueries in SELECT

01. For row 1, Sample_name = Alpha and Analysis_type = ‘Basic’. The **subquery then references Analysis_type** which is ‘Basic’ for this row.

```
SELECT
      ws_out.Sample_name,
      ws_out.Analysis_type,
      (SELECT
            Cost
        FROM
            Analysis_costs
        WHERE
            Analysis_type = 'Basic'
        ) AS Cost
FROM
    Water_samples AS ws_out;
```

Result set:

Sample_name	Analysis_type	Cost
Alpha	Basic	
Bravo	Advanced	
Charlie	Basic	
Delta	Advanced	
Echo	Basic	

Optimising queries

Subqueries in SELECT

02. The **subquery then executes**, using `‘Basic’` as the condition to filter the `Analysis_costs table` on, and returns a single value of 50.

```
SELECT
      ws_out.Sample_name,
      ws_out.Analysis_type,
      (SELECT
            Cost
        FROM
            Analysis_costs
        WHERE
            Analysis_type = ‘Basic’
        ) AS Cost
FROM
    Water_samples AS ws_out;
```

Subquery result set:

Analysis_type	Cost
Advanced	1000
Basic	**50**

Optimising queries

Subqueries in SELECT

03. Finally, Cost is updated with the subquery result, and the next row is evaluated.

```
SELECT
      ws_out.Sample_name,
      ws_out.Analysis_type,
      (SELECT
            Cost
        FROM
            Analysis_costs
        WHERE
            Analysis_type = ws_out.Analysis_type
        ) AS Cost
FROM
    Water_samples AS ws_out;
```

Result set:

Sample_name	Analysis_type	Cost
Alpha	Basic	50
Bravo	Advanced	
Charlie	Basic	
Delta	Advanced	
Echo	Basic	

Optimising queries

Subqueries in SELECT

04. For row 2, `Analysis type = ‘Advanced’`. The subquery filters `Analysis_costs` using `‘Advanced’`, and returns a value of 1000.

```
SELECT
      ws_out.Sample_name,
      ws_out.Analysis_type,
      (SELECT
            Cost
        FROM
            Analysis_costs
        WHERE
            Analysis_type = ‘Advanced’
        ) AS Cost
FROM
    Water_samples AS ws_out;
```

Result set:

Sample_name	Analysis_type	Cost
Alpha	Basic	50
Bravo	Advanced	1000
Charlie	Basic	
Delta	Advanced	
Echo	Basic	

Optimising queries

Subqueries in SELECT

05. The calculation is repeated for each row in the dataset.

```
SELECT
      ws_out.Sample_name,
      ws_out.Analysis_type,
      (SELECT
            Cost
        FROM
            Analysis_costs
        WHERE
            Analysis_type = ws_out.Analysis_type
        ) AS Cost
FROM
    Water_samples AS ws_out;
```

Result set:

Sample_name	Analysis_type	Cost
Alpha	Basic	50
Bravo	Advanced	1000
Charlie	Basic	50
Delta	Advanced	1000
Echo	Basic	50

Optimising queries

Subqueries in SELECT

> This query can also be done using a JOIN statement that is **simpler** to understand. Note how both the WHERE condition and the ON statement refer to the **connection** we would like to "join" on.

Subquery

```
SELECT
      ws_out.Sample_name,
      ws_out.Analysis_type,
      (SELECT
            Cost
        FROM
            Analysis_costs
        WHERE
            Analysis_type = ws_out.Analysis_type
        ) AS Cost
FROM
    Water_samples AS ws_out;
```

JOIN

```
SELECT
    ws_out.Sample_name,
    ws_out.Analysis_type,
    (an_cost.cost) AS Cost
FROM
   Water_samples AS ws_out
JOIN
   Analysis_costs AS an_cost
ON
   an_cost.Analysis_type = ws_out.Analysis_type;
```

Optimising queries

Subqueries in FROM/JOIN

> Subqueries can be used in `FROM` and `JOIN` to create intermediate or **derived tables**, which can be **queried again**. This is particularly useful when we would like to **use aggregated** data along with column data.

Suppose we want to calculate the total cost of all the samples analysed. We can use the previous query where we calculated the cost, and `SUM()` the column.

```sql
SELECT
      ws_out.Sample_name,
      ws_out.Analysis_type,
      (SELECT
            Cost
        FROM
            Analysis_costs
        WHERE
            Analysis_type = ws_out.Analysis_type) AS Cost
FROM
    Water_samples AS ws_out;
```

Previous results set:

Sample_name	Analysis_type	Cost
Alpha	Basic	50
Bravo	Advanced	1000
Charlie	Basic	50
Delta	Advanced	1000
Echo	Basic	50

Optimising queries

Subqueries in FROM/JOIN

```
SELECT
    SUM(Cost)
FROM
    (previous_query) AS Total_cost;
```

The previous query result can be used as a **derived table**, which the **outer query** uses to **sum** with. If we use a **subquery** in FROM, we **have to alias** the derived table using **AS**.

Input tables:

Sample _name	Purity	Analysis _type
Alpha	68	Basic
Bravo	75	Advanced
Charlie	52	Basic
Delta	89	Advanced
Echo	92	Basic

Analysis _type	Cost
Advanced	1000
Basic	50

Optimising queries

Subqueries in FROM/JOIN

```
SELECT
   SUM(Cost)
FROM
   (SELECT
       ws_out.Sample_name,
       ws_out.Analysis_type,
           (SELECT
               Cost
            FROM
               Analysis_costs
            WHERE
               Analysis_type = ws_out.Analysis_type
              ) AS Cost
    FROM
       Water_samples AS ws_out) AS Total_cost;
```

Input tables:

Sample_name	Purity	Analysis_type
Alpha	68	Basic
Bravo	75	Advanced
Charlie	52	Basic
Delta	89	Advanced
Echo	92	Basic

Analysis_type	Cost
Advanced	1000
Basic	50

Derived table:

Sample_name	Analysis_type	Cost
Alpha	Basic	50
Bravo	Advanced	1000
Charlie	Basic	50
Delta	Advanced	1000
Echo	Basic	50

Final result:

Total_cost
50

Subqueries can, in theory, be nested indefinitely, but become **harder to understand** with **each** added **level**.

Subqueries in FROM/JOIN

> Subqueries can add **complexity** to SQL statements, often making them harder to understand. However, **there are cases where they offer a more concise and efficient solution**.

```
SELECT
   SUM(Cost)
FROM
   (SELECT
       ws_out.Sample_name,
       ws_out.Analysis_type,
           (SELECT
               Cost
            FROM
               Analysis_costs
            WHERE
               Analysis_type = ws_out.Analysis_type
              ) AS Cost
    FROM
       Water_samples AS ws_out) AS Total_cost;
```

With the highlighting removed it becomes much **harder to understand** what the subquery does, especially compared to a solution that does not use subqueries:

```
SELECT
   SUM(ac.Cost) AS Total_cost
FROM
   Water_samples AS ws_out
INNER JOIN
   Analysis_costs AS an_cost
ON
   ws_out.Analysis_type = an_cost.Analysis_type;
```

Subqueries in WHERE/HAVING

> Subqueries can be used in `WHERE` and `HAVING` to enable **customised or advanced filtering** of results.

Suppose we want to retrieve the records in the `Water_samples` table that have **above-average purity**.

```
SELECT
   Sample_name,
   Purity,
   AVG(Purity)
FROM
    Water_samples
WHERE
     Purity > (SELECT
                 AVG(Purity)
                FROM
                 Water_samples)
```

Results set:

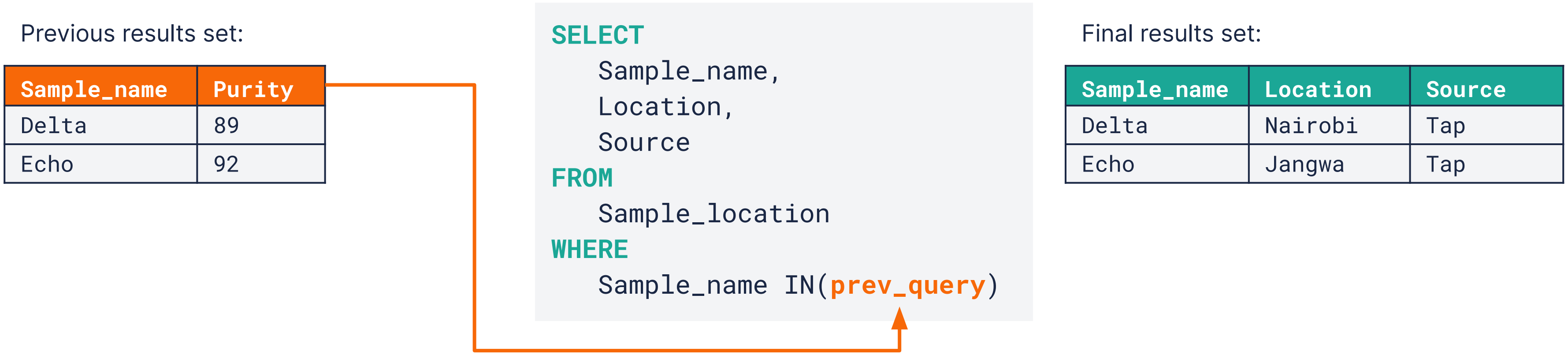
Sample_name	Purity
Delta	89
Echo	92

The subquery removes all records with `Purity < AVG(Purity)`.

Subqueries in WHERE/HAVING

> Subqueries in `WHERE` and `HAVING` can return **scalar values** if used with comparison operators, or can return a **single column** of values if used with the `IN()` operator.

Suppose we want to retrieve the **source types** of samples that have **above-average purity**. We can use results of the previous query as a list of options in a `WHERE… IN` clause to select samples that have **above-average purity**.



Previous results set:

Sample_name	Purity
Delta	89
Echo	92

```
SELECT
   Sample_name,
   Location,
   Source
FROM
   Sample_location
WHERE
   Sample_name IN(prev_query)
```

Final results set:

Sample_name	Location	Source
Delta	Nairobi	Tap
Echo	Jangwa	Tap

Subqueries in WHERE/HAVING

Here's the entire query and the results sets:

Input tables:

Sample_name	Purity	Analysis _type
Alpha	68	Basic
Bravo	75	Advanced
Charlie	52	Basic
Delta	89	Advanced
Echo	92	Basic

Sample_name	Location	Source
Alpha	Ziwa Maji	Lake
Bravo	Limpopo	River
Charlie	Mji	River
Delta	Nairobi	Tap
Echo	Jangwa	Tap

```
SELECT
    Sample_name,
    Location,
    Source
FROM
    Sample_location
WHERE
    Sample_name IN (
    SELECT
        Sample_name
    FROM
        Water_samples
    WHERE
        Purity > (
        SELECT
            AVG(Purity)
        FROM
            Water_samples)
        );
```

Intermediate results:

Avg_purity
75.2

Sample_name
Delta
Echo

Final result:

Sample_name	Location	Source
Delta	Nairobi	Tap
Echo	Jangwa	Tap