EXPLORE AI
ACADEMY

Optimising queries

Common Table Expressions (CTEs)

Optimising queries

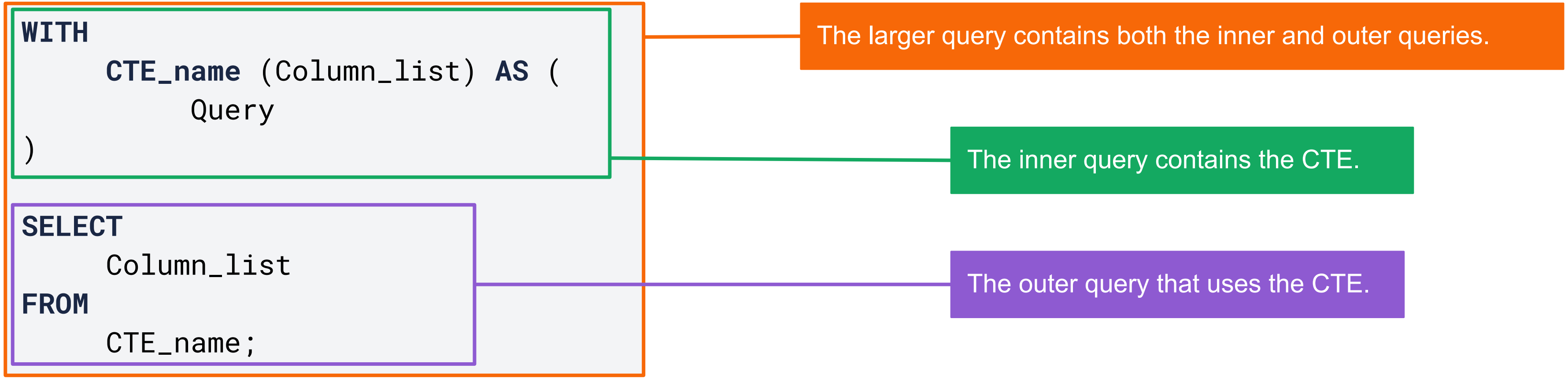
Data overview

We will use the following table called Employee that contains information about employees in a company located in both Kenya and South Africa. The salary is recorded in US dollars.

Start_date	First_name	Gender	Country	Office	Department	Salary
2015-06-01	Lily	Female	South Africa	Johannesburg	Finance	1933
2020-08-01	Gabriel	Male	Kenya	Nairobi	Marketing	1649
2022-03-01	Maryam	Female	Kenya	Nairobi	Data_analytics	1995
2015-07-15	Sophia	Female	South Africa	Cape Town	Data_analytics	2497
2019-05-01	Alex	Male	South Africa	Johannesburg	Data_analytics	1957
2012-01-01	Martha	Female	Kenya	Nairobi	Finance	2622
2014-05-01	Joshua	Male	South Africa	Cape Town	Finance	1933
2017-06-15	Emily	Female	Kenya	Kisumu	Data_analytics	2043
2016-01-01	David	Male	South Africa	Johannesburg	Marketing	2276

What are Common Table Expressions?

> A Common Table Expression, or CTE, is a **named query** that exists within the context of a **larger query**. Its results are temporarily stored such that they can be **referenced later by other queries**, other CTEs, or even itself.



```
WITH
    CTE_name (Column_list) AS (
        Query
)

SELECT
    Column_list
FROM
    CTE_name;
```

The larger query contains both the inner and outer queries.

The inner query contains the CTE.

The outer query that uses the CTE.

A CTE is only accessible within the larger query in which it is defined and will be lost as soon as the execution of this query is completed.

Optimising queries

The role of CTEs in code optimisation

> While performance is a crucial consideration in query optimisation, **readability** and **maintainability** are equally **important** in improving the development and sustainability of queries. **CTEs improve code** in the following ways:

01. Readability

- **Simplification:** CTEs allow us to break down **complex queries** into smaller, more intuitive logical parts that make the overall query **easier** to **read** and follow.
- **Comprehensibility:** By giving each CTE a **descriptive name** we make it easier to **understand** the context and **purpose** of the various parts of the query.

02. Maintainability

- **Maintenance:** CTEs help us to **modularise** our **queries** making it easier for changes or updates to be made to individual parts without affecting the entire query.
- **Debugging:** The **isolation** of different parts of a query and naming them accordingly makes the process of identifying and **fixing** issues **easier**.

03. Reusability

- **Avoid repetition:** Once a CTE has been defined, it can be **referenced multiple times** within the same query. This eliminates the need to repeat subquery calculations, but instead, reuse what we already have.

CTE syntax

Basic syntax:

```
WITH
    CTE_name (Column_1, Column_2) AS (
        SELECT
            expression
        FROM
            Table_name
)

SELECT
    Column_1, Column_2
FROM
    CTE_name;
```

NOTE: Opening and closing parenthesis are used to mark the beginning and end of a CTE definition respectively.

WITH: SQL keyword which indicates the beginning of the CTE definition.

CTE_name: This is the name assigned to the CTE, to be referenced later within the main query.

(Column_1, Column_2): An optional list of column names. These are the alias names that will be given to the columns returned by the CTE.

AS: SQL keyword which separates the CTE name and the CTE definition.

CTE definition: A valid SQL query that defines the CTE.

Main query: The larger/outer query that references the CTE defined above just as if it were a regular table.

Optimising queries

CTEs – Example

Suppose we want to compare the **total salary** with the **average salary** in **each department.**

- In this example, we create a **CTE named `Salary_totals`**, which calculates the total salary and the number of employees in each department.
- Then, the **main query** uses the results of the CTE to **calculate the average salary for each department** by dividing the "Total_salary" by the "No_Employees".

- We have simplified our query by separating the total salary and number of employees calculations from that of the average salary.
- The CTE name `Salary_totals` also makes its purpose clear which improves the readability and understandability of the query.
- Also, we have avoided repeating the SUM and COUNT calculations to find the average.

Query

```
WITH
      Salary_totals AS (
            SELECT
                  Department,
                  SUM(Salary) AS Total_salary
                  COUNT(Department) AS No_Employees
            FROM
                  Employees
            GROUP BY
                  Department
)
SELECT
      Department,
      No_Employees,
      Total_salary,
      Total_salary/ No_Employees AS Avg_salary
FROM
      Salary_totals;
```

Optimising queries

CTEs – Example

Output

Department	No_Employees	Total_salary	Avg_salary
Finance	3	6488	2163
Marketing	2	3925	1963
Data_analytics	4	8492	2123

Optimising queries

CTEs – Example

Suppose we want to find and award the **longest-serving employee** in **each department.**

- In this example, we create a **CTE named Tenure_rank** where we use the RANK( ) function to assign a rank number to each employee based on their start date within each department.
- Then, the **main query** filters the results of the CTE and only selects employees with a rank of 1, i.e. the employee with the longest tenure in the department.

- We have separated logic for calculating the employee ranks based on their start dates within each department into a CTE, leaving the main query only for retrieval and filtering. This makes the query easier to understand and manage.
- We can also easily reuse the ranking calculations in other parts of the query without having to repeat ourselves, for example, the lowest rank.

Query

```
WITH
        Tenure_rank AS (
                SELECT
                        Department,
                        First_name,
                        Start_date,
                        RANK() OVER (
                              PARTITION BY Department
                              ORDER BY Start_date DESC) AS Rank

                FROM
                        Employees
)
SELECT
        Department,
        First_name,
        Start_date
FROM
        Tenure_rank
WHERE Rank = 1;
```

Optimising queries

CTEs – Example

Output

Department	First_name	Start_date
Finance	Martha	2012-01-01
Marketing	David	2016-01-01
Data_analytics	Sophia	2015-07-15

Optimising queries

Multiple CTEs

> It is possible to **specify more than one CTE** within a single query. In such a case, each CTE definition is **separated by a comma.**

Basic syntax:

```
WITH
      CTE_name1 (Column_1, Column_2) AS (
           query1),
      CTE_name2 (Column_3, Column_4) AS (
           query2)
SELECT *
FROM
CTE_name1
UNION ALL
SELECT *
FROM
CTE_name2;
```

First CTE definition.

Second CTE definition.

Optimising queries

Multiple CTEs – Example

Suppose we want to **view** the **employees posted at the main offices in Kenya and South Africa.**

In this example, we have **created two CTEs**. We have:

- **`Main_office_employee_ke`** which returns results for employees working in the "Nairobi" office in Kenya.

- **`Main_office_employee_sa`** whose results contain employees working in the “Johannesburg" office in South Africa.

The main query then combines these two result sets using the **`UNION` operator** into a single unified result containing all the employees working in the company’s main offices in Kenya and South Africa.

```
WITH
        Main_office_employee_ke AS (
                SELECT
                        Name,
                        Gender,
                        Department
                FROM    Employees
                WHERE Country = “Kenya” AND Office = “Nairobi”),

        Main_office_employee_sa AS
                SELECT
                        Name,
                        Gender,
                        Department
                FROM    Employees
                WHERE Country = “South Africa” AND Office =
        “Johannesburg”)
SELECT *
FROM    Main_office_employee_ke
UNION
SELECT *
FROM    Main_office_employee_sa;
```

Multiple CTEs – Example

Output

Name	Gender	Department
Gabriel	Male	Marketing
Maryam	Female	Data_analytics
Martha	Female	Finance
Lily	Female	Finance
Alex	Male	Data_analytics
David	Male	Marketing