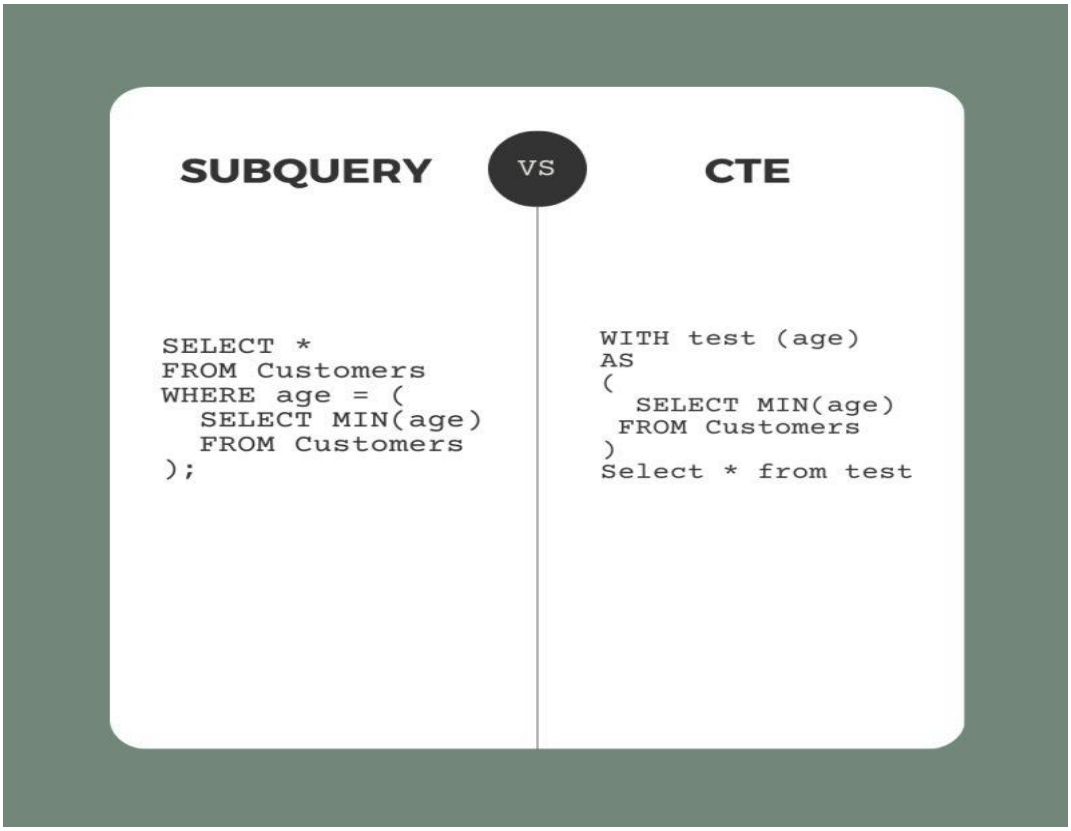
Subqueries and CTEs for Complex Data Retrieval

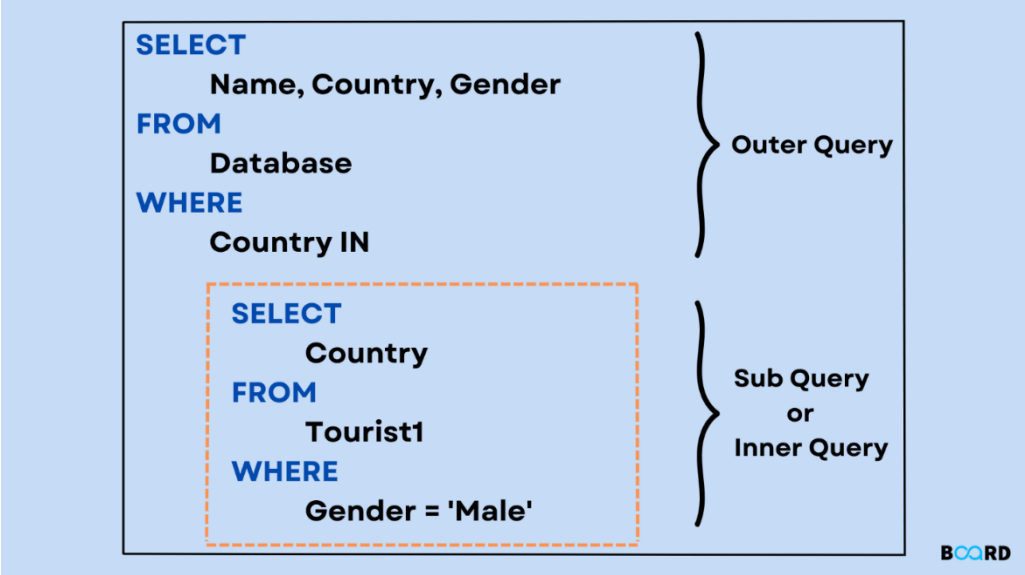
Objective:

- **Subquery**
  - Learn to construct and utilize subqueries in different parts of SQL queries (SELECT, WHERE, FROM).

- **Common Table Expressions**
  - Gain proficiency in creating and using Common Table Expressions (CTEs) to organize and simplify complex queries, including recursive CTEs..



Session: Advanced Query Techniques



General example of subquery

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE age = (
  SELECT MIN(age)
  FROM Customers
);
```

customer_id	first_name	last_name	age	country
2	Robert	Luna	22	USA
3	David	Robinson	22	UK

Working with Subqueries

Key Points:

- **Purpose**: To use subqueries for breaking down complex queries into manageable parts.

Types of Subqueries:

1. **Single-Row Subquery**:
   - Returns a single row with one or more columns.
   - **Syntax:**

```
SELECT
  column_name
FROM
  table_name
WHERE
  column_name = (SELECT MAX(column_name) FROM table_name);
```

2. **Multi-Row Subquery**:
   - Returns multiple rows, typically used with IN, ANY, ALL, etc.
   - **Syntax:**

```
SELECT
  column_name
FROM
  table_name
WHERE
  column_name IN (SELECT column_name FROM another_table);
```

3. **Multi-Column Subquery**:
   - Returns multiple columns, where the outer query compares a tuple of values.
   - **Syntax:**

```
SELECT column_list
FROM outer_table
WHERE (column1, column2) IN (
    SELECT column1, column2
    FROM inner_table
    WHERE condition
);
```

4. Corelated Subquery

- A correlated subquery is a subquery that references columns from the outer query. The subquery is executed once for each row processed by the outer query.
- **Syntax**:

```
SELECT column_list
FROM outer_table alias1
WHERE expression operator (
    SELECT column_list
    FROM inner_table alias2
    WHERE alias2.column_name = alias1.column_name
);
```

- **Example:**

```
SELECT e1.employee_id, e1.first_name, e1.salary

FROM employees e1

WHERE e1.salary > (

    SELECT AVG(e2.salary)

    FROM employees e2

    WHERE e2.department_id = e1.department_id

);
```

You can use subqueries in Select and from clause:

1. **Subqueries in SELECT**:

```
SELECT
  column_name,
  (SELECT MAX(column_name) FROM another_table) as max_value
FROM
  table_name;
```

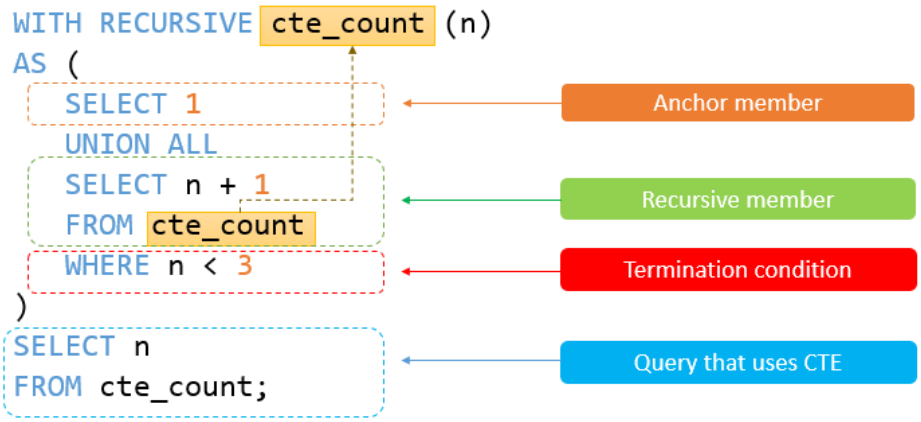
2. **Subqueries in FROM**:

```
SELECT
  sub.column_name
FROM
  (SELECT column_name FROM table_name) sub;
```

Common Table Expressions (CTEs)

A Common Table Expression (CTE) is a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs are useful for simplifying complex queries by breaking them down into simpler, reusable parts.

General Example of CTE



Key Points:

- **Purpose**: To organize and simplify complex queries using CTEs.

```
WITH test (age)
AS
(
  SELECT MIN(age)
 FROM Customers
)
Select * from test
```

Syntax:

```
WITH cte_name AS (
  SELECT column_name
  FROM table_name
```

```
)
SELECT
  column_name
FROM
  cte_name;
```

Examples:

1. **Simple CTE**:

   This example demonstrates a simple CTE that selects data from a table.

```
WITH EmployeeCTE AS (
    SELECT EmployeeID, FirstName, LastName, DepartmentID
    FROM Employees
    WHERE DepartmentID = 3
)
SELECT *
FROM EmployeeCTE;
```

2. **CTE with Aggregation:**

```
WITH SalesCTE AS (
    SELECT EmployeeID, SUM(SalesAmount) AS TotalSales
    FROM Sales
    GROUP BY EmployeeID
)
SELECT EmployeeID, TotalSales
FROM SalesCTE
WHERE TotalSales > 10000;
```

Benefits of Using CTEs

1. **Readability:** Breaks down complex queries into smaller, manageable parts.
2. **Reusability:** You can reference the same CTE multiple times in the main query.
3. **Modularity:** Makes it easier to debug and test parts of your query.

When to Use CTEs

1. Simplifying complex joins or subqueries.
2. Organizing recursive queries, such as traversing a hierarchy.
3. Creating modular and reusable query components.

Limitations

1. CTEs are temporary and only exist within the scope of the query they are defined in.
2. Performance may be an issue for large data sets.

By following this detailed manual, you will develop a comprehensive understanding of advanced query techniques, and how to effectively use these concepts to write powerful SQL queries.