# Subqueries

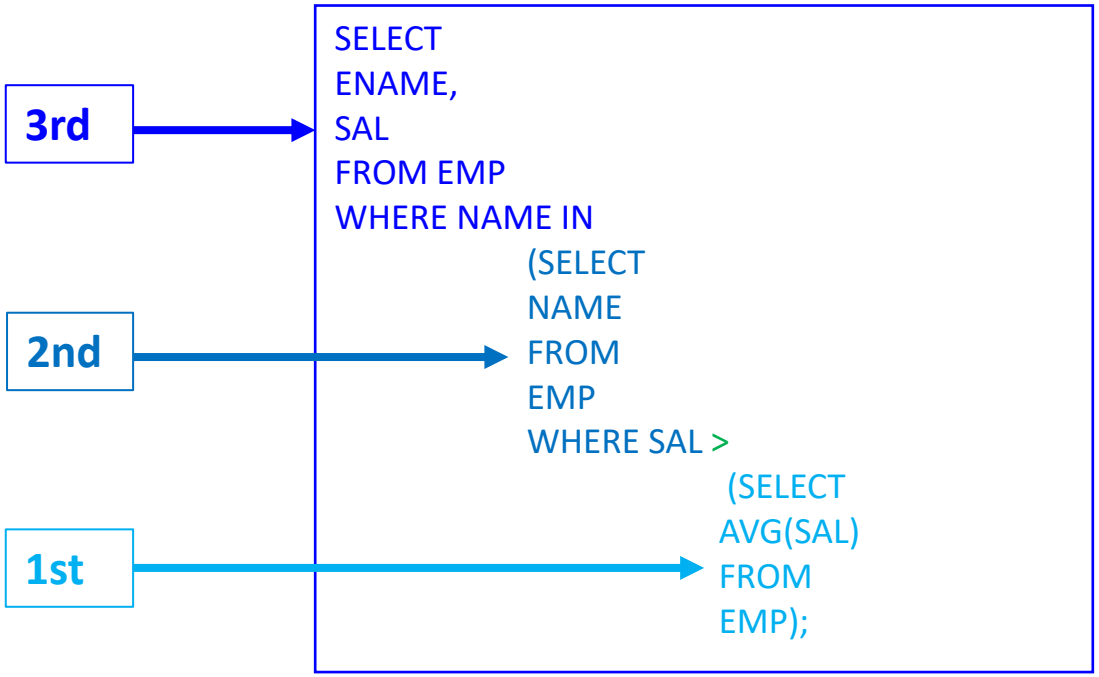
- A subquery is a query that is nested inside a SELECT, INSERT, UPADTE or DELETE statement

- A subquery may occur anywhere an expression is allowed, such as:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
  - A HAVING clause
  - Etc.

- Subqueries must be enclosed in parenthesis

# Subquery Order of Execution

- The inner-most query is executed first
- All sub-queries follow the SQL query order of execution



# Scalar Subqueries

- A scalar subquery expression is a subquery that returns exactly one column value from one row.
- You can use scalar subqueries in most places that allow an expression except:
  - As default values for columns
  - As hash expressions for clusters
  - In the RETURNING clause of DML statements
  - As the basis of a function-based index
  - In CHECK constraints
  - In WHEN conditions of CASE expressions
  - In GROUP BY and HAVING clauses
  - In START WITH and CONNECT BY clauses
  - In statements that are unrelated to queries, such as CREATE PROFILE

# Subqueries in the FROM clause

- This is referred to as an inline view

# Correlated Subqueries

- A correlated subquery contains a reference to a table in the outer query
- In a normal subquery the inner subqueries run first and execute once, they return values for the outer subquery
- In a correlated subquery the subquery is executed once for each row in the outer query
- Correlated subqueries can be used when the subquery needs to return a different result for each row in the outer query
- Correlated subqueries can be used with logical operators (<,>,=…) and IN, ANY, ALL operators

# Where Exists / Not Exists

- WHERE EXISTS is used to test the existence of any record in a subquery, and returns TRUE if the record exists
- WHERE NOT EXISTS is used to test the non-existence of any record in a subquery, returns TRUE if record doesn’t exist
- Usually used with correlated subqueries

# CTE

- A CTE (Common Table Expression) is a temporary result set that you can reference in your SELECT, INSERT, UPDATE or DELETE statement

- Because it is stored in the temporary tablespace it is returned from the temporary table rather than the base tables making it more efficient in some situations such as when the CTE is being used more than once

```
WITH CTE_NAME (COLUMN(S)) AS
(CTE SELECT_STATEMENT)
SELECT COLUMN(S) FROM CTE_NAME;
```

# Recursive CTE

- A recursive CTE has one subquery that refers to the CTE itself
- Recursive CTEs enable you to process hierarchical data and is an alternative to Hierarchical Queries

```
WITH CTE_NAME (COLUMN(S))
AS (
ANCHOR_CLAUSE
UNION ALL
RECURSIVE_CLAUSE )
SELECT ... FROM ...;
```

- The anchor clause sets an initial set of rows that are displayed at the top of the hierarchy
- The anchor clause cannot reference the CTE itself
- The anchor clause can be made up of set operators (UNION, MINUS, ADD etc.)

- The Anchor clause and recursive clause must be combined with a union all operator

- This must reference the CTE
- The recursive clause selects the next layer of the hierarchy based on the previous layer
- The anchor result set is the first layer
- Aggregations, window functions, distinct, order by, limit are not allowed

# CTE Evaluation

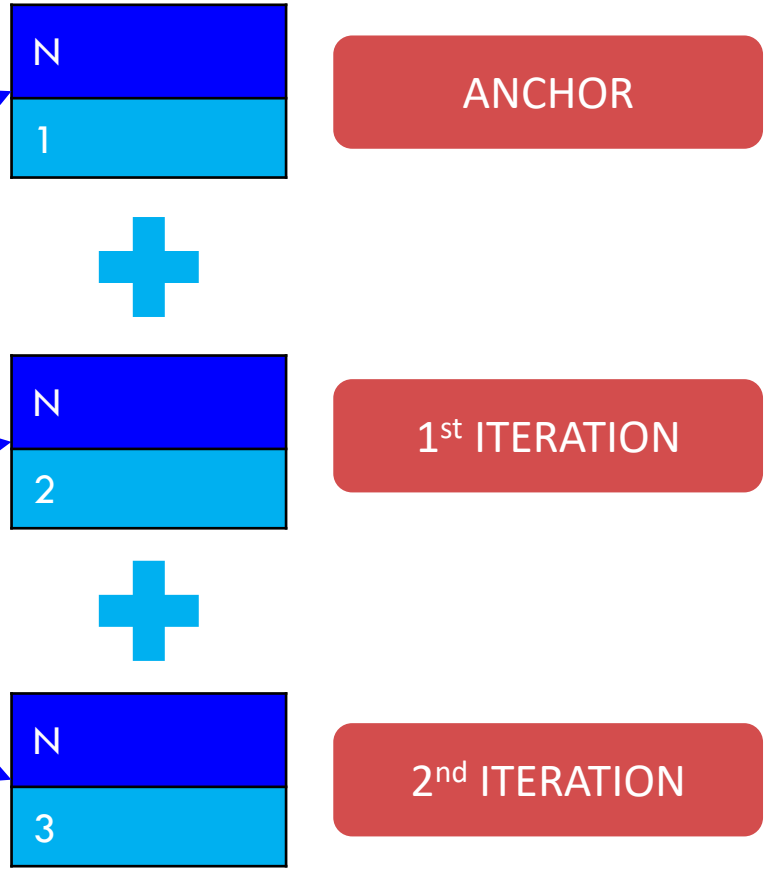
```
WITH CTE_EMP (EMPNO, ENAME, MGR)
AS (
SELECT EMPNO, ENAME, MGR FROM EMP WHERE ENAME IN
('BLAKE','CLARK')
UNION ALL
SELECT EMP.EMPNO, EMP.ENAME, EMP.MGR FROM CTE_EMP JOIN
EMP ON CTE_EMP.EMPNO = EMP.MGR
)
SELECT * FROM CTE_EMP
```

| EMPNO | ENAME | MGR |
|---|---|---|
| 7698 | BLAKE | 7839 |
| 7782 | CLARK | 7839 |

| | | |
|---|---|---|
| 7499 | ALLEN | 7698 |
| 7521 | WARD | 7698 |
| 7654 | MARTIN | 7698 |
| 7844 | TURNER | 7698 |
| 7900 | JAMES | 7698 |
| 7934 | MILLER | 7782 |

# CTE Evaluation

```
WITH CTE_EMP (N)
AS (
SELECT N FROM SCALAR_TABLE
UNION ALL
SELECT N+1 FROM CTE_EMP WHERE N<4
)
SELECT * FROM CTE_EMP
```



# CTE Considerations

- You must be careful to ensure that your recursive CTE does not end up in an infinite loop

```
WITH CTE_EMP (EMPNO, ENAME, MGR)
AS (
SELECT EMPNO, ENAME, MGR FROM EMP WHERE ENAME IN
('BLAKE','CLARK')
UNION ALL
SELECT EMP.EMPNO, EMP.ENAME, EMP.MGR FROM CTE_EMP JOIN
EMP ON CTE_EMP.EMP = EMP.EMP
)
SELECT * FROM CTE_EMP
```

```
WITH CTE_EMP (N)
AS (
SELECT N FROM SCALAR_TABLE
UNION ALL
SELECT N+1 FROM CTE_EMP
)
SELECT * FROM CTE_EMP
```