Simplifying Queries with Common Table Expressions

Jason P. Browning

EXECUTIVE DIRECTOR, DIXIE STATE UNIVERSITY

@jason_from_ky

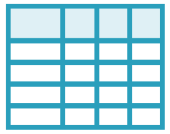
Overview

Defining common table expressions

Using common table expressions

Non-recursive and recursive expressions

Common Table Expressions



Created result sets that can be referenced in subsequent queries

Promotes readability and interpretability for complex queries

Can reference a CTE view multiple times in same query statement

```
WITH freshmen AS (
SELECT a.student_id,
       a.student_name,
       b.hours_completed
  FROM students a
 INNER JOIN credit_hours b
    ON a.student_id = b.student_id
 WHERE b.hours_completed < 30);
```

- CTEs are defined using the WITH keyword
- A name is assigned to the CTE
- Logic is specified following the AS keyword

```
WITH freshmen AS (
SELECT a.student_id,
       a.student_name,
       b.hours_completed
  FROM students a
 INNER JOIN credit_hours b
    ON a.student_id = b.student_id
 WHERE b.hours_completed < 30)
SELECT student_id,
       student_name
  FROM freshmen;
```

- CTEs are defined using the WITH keyword
- A name is assigned to the CTE
- Logic is specified following the AS keyword
- CTEs are defined before the query statement
- The CTE is referenced in the query statement by its name

```
SELECT a.trans_id,
       b.first_name, b.last_name,
       a.amount
  FROM transactions a
INNER JOIN customers b
    ON a.cust_id = b.cust_id
 WHERE a.amount >
       (SELECT AVG(amount)
          FROM transactions aa
         WHERE a.cust_id = aa.cust_id)
 ORDER BY a.trans_id;
```

- This query employs a correlated subquery
- The subquery calculates the average transaction amount by customer
- The WHERE clause of the primary query limits results to transactions that have an amount exceeding the customer's average transaction amount

```
WITH AvgAmount AS (
  SELECT cust_id,
      AVG(amount) AS avg_amount
   FROM transactions
GROUP BY cust_id)
  SELECT a.trans_id,
      b.first_name, b.last_name,
      a.amount
    FROM transactions a
INNER JOIN customers b
      ON a.cust_id = b.cust_id
INNER JOIN AvgAmount c
      ON a.cust_id = c.cust_id
   WHERE a.amount > c.avg_amount;
```

- The query can be rewritten using a CTE
- The CTE presented calculates the average transaction amount for each customer
- The query statement references the AvgAmount CTE to limit results
- As before, the query returns transactions that have an amount exceeding the customer's average transaction amount

Demo

Writing CTEs

CTEs can be used to improve readability and interpretability of code.

Recursive CTEs

CTE that calls itself until some condition is met

Can be used to create series or work with hierarchical data

Similar to a “for loop” in other programming languages

```
WITH RECURSIVE series (list_num) AS (
    SELECT 5
     UNION ALL
    SELECT list_num + 5 FROM series
     WHERE list_num + 5 <= 50)
SELECT list_num FROM series;
```

Recursive CTEs

Use the RECURSIVE keyword to create a recursive expression

When creating a CTE, optional column names can be specified

Code generates a list of multiples of five until 50 is reached

```
WITH RECURSIVE series (list_num) AS (
    SELECT 5
     UNION ALL
    SELECT list_num + 5
     FROM series
     WHERE list_num + 5 <= 50)
SELECT list_num FROM series;
```

- SELECT clause adds 5 to previous value
- FROM clause self-references the CTE
- WHERE clause stops recursive action once 50 is reached

list_num
5
10
15
20
25
30
35
40
45
50

Summary

CTEs create temporary views that can be referenced within subsequent queries

CTEs can improve readability of code

Recursive CTEs can perform iterative functions