(T17)使用CommonTableExpressions(CTE)  
CourseGUID: e48417fc-9db5-4e99-822c-706c5ccef6cc  
=======================================================================  
(T17)使用CommonTableExpressions(CTE)  
=======================================================================  
0. Summary

0.1. Summary

0.2. Q&A

-----------

1. DerivedTables\_CommonTableExpressions(CTE)

1.1. Create Sample Data

1.2. Drop View if it exists

1.3. Temp table

1.4. Table Variable

1.5. Derived Tables

1.6. CommonTableExpressions(CTE)

-----------

2. CommonTableExpressions(CTE)

2.1. CommonTableExpressions(CTE)

2.2. Many CommonTableExpressions(CTE)s in ONE With

-----------

3. UpdatableCommonTableExpressions(CTE)

3.1. If CTE has only one based table

3.2. If CTE has many based tables

3.3. If CTE has many based tables

3.4. Incorrectly update

-----------

4. RecursiveCommonTableExpressions(CTE)

4.1. Recursive CommonTableExpressions(CTE)

-----------

5. GetOrganizationHierarchy

6. Clean up  
=======================================================================

0. Summary

0.1. Summary

1. Common Table Expressions(CTE) and alternatives

1.1.

VIEW

VIEW can be saved in the database and be re-used some where else.

If you don't want to re-use,

then you may use CTE, Derived Tables, Temp Tables, Table Variable etc.

--------

1.2.

Temp table

Databases --> System Databases --> tempdb --> Tables --> tempTables

Temporary tables are in SystemDatabases TempTB.

1.2.1.

One pund(#) symbol prefix means Local Temporary tables.

Local Temporary tables can only survive

in current connection/session/current Query file.

Local Temporary tables will be destroyed when closing current connection.

1.2.2.

Two pund(##) symbol prefix means Global Temporary tables.

Global Temporary tables can survive

in many connections/sessions/Query files.

Global Temporary tables will be destroyed when closing all connections.

--------

1.3.

Derived Tables

Derived tables are available

only in the context of the current query.

--------

1.4.

Common Table Expressions(CTE)

1.4.1.

Common Table Expressions(CTE) must be used immediately after you defined the CTE.

It can not survive in next next Query.

It is available within a single SELECT, INSERT, UPDATE, DELETE,

or CREATE VIEW statement.

You may define many CommonTableExpressions(CTE)s in ONE With

1.4.2.

Syntax:

--WITH cteName (ColumnA1, ColumnA2, ...)

--AS

--( SELECT   ColumnB1, ColumnB2, ... )

We consider CTE as a normal Table.

In this case, Table Name is cteName, we called it as CTE Name.

Table column is ColumnA1, ColumnA2, ..., We called it as CTE Columns.

We called ( SELECT   ColumnB1, ColumnB2, ... ) as CTE Query.

The ColumnB1, ColumnB2... in the cteQuery

should be able to map to the cteColumns (ColumnA1, ColumnA2, ...).

In this case,

ColumnB1 map to ColumnA1,

ColumnB2 map to ColumnA2...etc.

We normally name ColumnB1 in cteQuery and ColumnA1 in cteColumn

as the same name to avoud confusion.

but it is not necessary.

1.4.3.

Updatable CommonTableExpressions(CTE)

1.4.3.1.

If CTE has only one based table,

then we may update the CommonTableExpressions(CTE).

1.4.3.2.

If CTE has many based tables,

and if UPDATE affects multiple base tables,

then it will return ERROR and terminates the UPDATE.

1.4.3.3.

If CTE has many based tables,

and if UPDATE affects only ONE base table,

then we may update the CommonTableExpressions(CTE).

But it might not work as we expected

0.2. Q&A

學生提問

<https://www.facebook.com/groups/934567793358849/posts/2020993988049552/>

影片T017 02 CommonTableExpressionsCTE

18:27的位置

依老師影片中所講解的，我的認知，似乎是先執行Anchor Member部分的語法，查到LeaderId，在用該結果(只有一筆)去紅色(Recursive Member)的部分INNER JOIN Gamer的GamerId，再得出該資料列的欄位資訊

例如:

EXEC spGetLeaders 5;

->則在紅色的部分(Recursive Member)查詢出GamerId為4的相關資料

->再取出GamerId=4的LeaderId->LeaderId = 1，以此為條件，查詢出GamerId為1的相關資料

我的問題是

1.在查詢出GamerId為4的相關資料後，該語法為何不會直接執行完畢，並只顯示GamerId為7及4的相關資料，

而會繼續查詢GamerId為1的相關資料並顯示GamerId為7、4、1的所有資料呢?

2.請問CTE中的Recursive的運作原理是什麼，因為看上去語法和直接寫兩個查詢語法並UNION起來似乎差不多，其中導致會遞迴查詢的語法結構是哪個部份呢?

感謝!



---------------------------------------------------------------------

答案:

問題2，紅框是 CTE 的本體 cteGamer，在裡面又 JOIN 自己，所以形成遞迴

A picture containing graphical user interface

Description automatically generated

問題1，紅框是資料的起點，接著綠框是繼續找下一筆的遞迴，一直找到最後 LeaderId = NULL，而 GamerId 並沒有 NULL 所以沒有吻合的資料，停止遞迴

Graphical user interface, text, application

Description automatically generated with medium confidence

結論:

Id等於1的就是所有Gamer的大領導

所以不管你輸入哪一個Id，最後一個一定是GamerId==1

它的原理就是先利用Anchor找出Result 0

然後透過Member去遞迴，找出剩下的所有Result

CTE搭配Union就是TSQL遞迴的固定寫法。

如果你只是兩個Table單純的Union

就沒辦法有遞迴唷。

1. DerivedTables\_CommonTableExpressions(CTE)

--=======================================================================================================

--T017\_01\_DerivedTables\_CommonTableExpressions(CTE)

--=======================================================================================================

1.1. Create Sample Data

--=======================================================================================================

--T017\_01\_01

--Create Sample Data

--Drop Table if it exists

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Gamer' ) )

    BEGIN

        TRUNCATE TABLE Gamer;

        DROP TABLE Gamer;

    END;

GO -- Run the previous command and begins new batch

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Team' ) )

    BEGIN

        TRUNCATE TABLE Team;

        DROP TABLE Team;

    END;

GO -- Run the previous command and begins new batch

CREATE TABLE Team

(

  TeamId INT IDENTITY(1, 1)

             PRIMARY KEY ,

  TeamName [NVARCHAR](100) NULL

);

GO -- Run the prvious command and begins new batch

CREATE TABLE Gamer

(

  GamerId INT IDENTITY(1, 1)

              PRIMARY KEY ,

  [Name] NVARCHAR(100) NULL ,

  Gender NVARCHAR(100) NULL ,

  LeaderId INT FOREIGN KEY REFERENCES Gamer ( GamerId )

               NULL ,

  TeamId INT FOREIGN KEY REFERENCES Team ( TeamId )

             NULL

);

GO -- Run the prvious command and begins new batch

INSERT  Team

VALUES  ( N'Team01' );

INSERT  Team

VALUES  ( N'Team02' );

INSERT  Team

VALUES  ( N'Team03' );

INSERT  Team

VALUES  ( N'Team04' );

GO -- Run the prvious command and begins new batch

INSERT  Gamer

VALUES  ( N'AName01', 'Male', NULL, 2 );

INSERT  Gamer

VALUES  ( N'AName02', 'Female', 1, 2 );

INSERT  Gamer

VALUES  ( N'AName03', 'Female', 2, 1 );

INSERT  Gamer

VALUES  ( N'CName04', 'Male', 1, 4 );

INSERT  Gamer

VALUES  ( N'CName05', 'Female', 3, 2 );

INSERT  Gamer

VALUES  ( N'SName06', 'Male', 1, 1 );

INSERT  Gamer

VALUES  ( N'SName07', 'Female', 4, 1 );

INSERT  Gamer

VALUES  ( N'SName08', 'Female', 4, 1 );

GO -- Run the prvious command and begins new batch

SELECT  \*

FROM    dbo.Gamer;

SELECT  \*

FROM    dbo.Team;

GO -- Run the prvious command and begins new batch

/\*

AName01

  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  |          |           |

AName02    CName04     SName06

  |          |\_\_\_\_\_\_\_\_\_

  |          |        |

AName03    SName07   SName08

  |

CName05

\*/

Graphical user interface, application, table

Description automatically generated

1.2. Drop View if it exists

--=======================================================================================================

--T017\_01\_02

--Drop View if it exists

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'vwGamerCount' ) )

    BEGIN

        DROP VIEW vwGamerCount;

    END;

GO -- Run the previous command and begins new batch

CREATE VIEW vwGamerCount

AS

    SELECT  t.TeamName ,

            g.TeamId ,

            COUNT(\*) AS TotalGamer

    FROM    dbo.Gamer g

            JOIN Team t ON g.TeamId = t.TeamId

    GROUP BY t.TeamName ,

            g.TeamId;

GO -- Run the prvious command and begins new batch

--Get TeamName and its TotalGamer.

--Select only when TotalGamer>= 2

SELECT  TeamName ,

        TotalGamer

FROM    vwGamerCount

WHERE   TotalGamer >= 2;

GO -- Run the prvious command and begins new batch

/\*

VIEW can be saved in the database and be re-used some where else.

If you don't want to re-use,

then you may use CTE, Derived Tables, Temp Tables, Table Variable etc.

\*/

Table

Description automatically generated

1.3. Temp table

--=======================================================================================================

--T017\_01\_03

--Temp table

IF OBJECT\_ID('tempdb..#TempGamerCount') IS NOT NULL

    BEGIN

        TRUNCATE TABLE #TempGamerCount;

        DROP TABLE #TempGamerCount;

    END;

GO -- Run the previous command and begins new batch

SELECT  t.TeamName ,

        g.TeamId ,

        COUNT(\*) AS TotalGamer

--Slect into tamp table

INTO    #TempGamerCount

FROM    dbo.Gamer g

        JOIN Team t ON g.TeamId = t.TeamId

GROUP BY t.TeamName ,

        g.TeamId;

GO -- Run the previous command and begins new batch

--Get TeamName and its TotalGamer.

--Select only when TotalGamer>= 2

SELECT  TeamName ,

        TotalGamer

FROM    #TempGamerCount

WHERE   TotalGamer >= 2;

GO -- Run the previous command and begins new batch

/\*

1.

Databases --> System Databases --> tempdb --> Tables --> tempTables

Temporary tables are in SystemDatabases TempTB.

1.1.

One pund(#) symbol prefix means Local Temporary tables.

Local Temporary tables can only survive

in current connection/session/current Query file.

Local Temporary tables will be destroyed when closing current connection.

1.2.

Two pund(##) symbol prefix means Global Temporary tables.

Global Temporary tables can survive

in many connections/sessions/Query files.

Global Temporary tables will be destroyed when closing all connections.

\*/

Table

Description automatically generated with medium confidence

1.4. Table Variable

--=======================================================================================================

--T017\_01\_04

--Table Variable

DECLARE @GamerCount TABLE

(

  TeamName NVARCHAR(50) ,

  TeamID INT ,

  TotalGamer INT

);

--Insert into Table Variable

INSERT  @GamerCount

        SELECT  t.TeamName ,

                g.TeamId ,

                COUNT(\*) AS TotalGamer

        FROM    dbo.Gamer g

                JOIN Team t ON g.TeamId = t.TeamId

        GROUP BY t.TeamName ,

                g.TeamId;

--Get TeamName and its TotalGamer.

--Select only when TotalGamer>= 2

SELECT  TeamName ,

        TotalGamer

FROM    @GamerCount

WHERE   TotalGamer >= 2;

GO -- Run the previous command and begins new batch

/\*

Table Variable is stored in TempDB and can only survive

in the batch, statement block, or stored procedure.

Table Variable be passed as parameters between procedures.

\*/

Table

Description automatically generated

1.5. Derived Tables

--=======================================================================================================

--T017\_01\_05

--Derived Tables

SELECT  TeamName ,

        TotalGamer

FROM    ( SELECT    t.TeamName ,

                    g.TeamId ,

                    COUNT(\*) AS TotalGamer

          FROM      dbo.Gamer g

                    JOIN Team t ON g.TeamId = t.TeamId

          GROUP BY  t.TeamName ,

                    g.TeamId

        ) AS GamerCount

WHERE   TotalGamer >= 2;

GO -- Run the prvious command and begins new batch

/\*

Derived tables are available

only in the context of the current query.

\*/

Table

Description automatically generated

1.6. CommonTableExpressions(CTE)

--=======================================================================================================

--T017\_01\_06

--CommonTableExpressions(CTE)

WITH    GamerCount ( TName, TId, TotalPeople )

          AS ( SELECT   t.TeamName ,

                        g.TeamId ,

                        COUNT(\*) AS TotalGamer

               FROM     dbo.Gamer g

                        JOIN Team t ON g.TeamId = t.TeamId

               GROUP BY t.TeamName ,

                        g.TeamId

             )

    SELECT  TName ,

            TotalPeople

    FROM    GamerCount

    WHERE   TotalPeople >= 2;

GO -- Run the prvious command and begins new batch

/\*

1.

Common Table Expressions(CTE)

1.1.

Common Table Expressions(CTE) must be used immediately after you defined the CTE.

It can not survive in next next Query.

It is available within a single SELECT, INSERT, UPDATE, DELETE,

or CREATE VIEW statement.

You may define many CommonTableExpressions(CTE)s in ONE With

1.2.

Syntax:

--WITH cteName (ColumnA1, ColumnA2, ...)

--AS

--( SELECT   ColumnB1, ColumnB2, ... )

We consider CTE as a normal Table.

In this case, Table Name is cteName, we called it as CTE Name.

Table column is ColumnA1, ColumnA2, ..., We called it as CTE Columns.

We called ( SELECT   ColumnB1, ColumnB2, ... ) as CTE Query.

The ColumnB1, ColumnB2... in the cteQuery

should be able to map to the cteColumns (ColumnA1, ColumnA2, ...).

In this case,

ColumnB1 map to ColumnA1,

ColumnB2 map to ColumnA2...etc.

We normally name ColumnB1 in cteQuery and ColumnA1 in cteColumn

as the same name to avoud confusion.

but it is not necessary.

\*/

Table

Description automatically generated

==================================================================

2. CommonTableExpressions(CTE)

--=======================================================================================================

--T017\_02\_CommonTableExpressions(CTE)

--=======================================================================================================

/\*

1.

Common Table Expressions(CTE)

1.1.

Common Table Expressions(CTE) must be used immediately after you defined the CTE.

It can not survive in next next Query.

It is available within a single SELECT, INSERT, UPDATE, DELETE,

or CREATE VIEW statement.

You may define many CommonTableExpressions(CTE)s in ONE With

1.2.

Syntax:

--WITH cteName (ColumnA1, ColumnA2, ...)

--AS

--( SELECT   ColumnB1, ColumnB2, ... )

We consider CTE as a normal Table.

In this case, Table Name is cteName, we called it as CTE Name.

Table column is ColumnA1, ColumnA2, ..., We called it as CTE Columns.

We called ( SELECT   ColumnB1, ColumnB2, ... ) as CTE Query.

The ColumnB1, ColumnB2... in the cteQuery

should be able to map to the cteColumns (ColumnA1, ColumnA2, ...).

In this case,

ColumnB1 map to ColumnA1,

ColumnB2 map to ColumnA2...etc.

We normally name ColumnB1 in cteQuery and ColumnA1 in cteColumn

as the same name to avoud confusion.

but it is not necessary.

\*/

2.1. CommonTableExpressions(CTE)

--=======================================================================================================

--T017\_02\_01

--CommonTableExpressions(CTE) defined and must used immediately.

---------------------------------------------------------------------------------------------------------

--T017\_02\_01\_01

--CommonTableExpressions(CTE) defined and must used immediately.

WITH    GamerCount ( TId, TotalPeople )

          AS ( SELECT   g.TeamId ,

                        COUNT(\*) AS TotalGamers

               FROM     Gamer g

               GROUP BY g.TeamId

             )

    SELECT  t.TeamName ,

            TotalPeople

    FROM    GamerCount g

            JOIN Team t ON g.TId = t.TeamId

    ORDER BY g.TotalPeople;

GO -- Run the prvious command and begins new batch

Table

Description automatically generated

---------------------------------------------------------------------------------------------------------

--T017\_02\_01\_02

--Common table expression(CTE) defined but not used immediately.

--ERROR

/\*

WITH    GamerCount ( TId, TotalPeople )

          AS ( SELECT   g.TeamId ,

                        COUNT(\*) AS TotalGamers

               FROM     Gamer g

               GROUP BY g.TeamId

             )

--Common table expression(CTE) defined but not used immediately.

SELECT  'Hello';

SELECT  t.TeamName ,

        TotalPeople

FROM    GamerCount g

        JOIN Team t ON g.TId = t.TeamId

ORDER BY g.TotalPeople;

GO -- Run the prvious command and begins new batch

\*/

/\*

Error

--Msg 422, Level 16, State 4, Line 261

--Common table expression defined but not used.

\*/

2.2. Many CommonTableExpressions(CTE)s in ONE With

--=======================================================================================================

--T017\_02\_02

--Many CommonTableExpressions(CTE)s in ONE With

SELECT  \*

FROM    Team;

WITH    cteTeam01Team03 ( TName, TotalPeople )

          AS ( SELECT   t.TeamName ,

                        COUNT(g.GamerId) AS TotalGamers

               FROM     Gamer g

                        JOIN Team t ON g.TeamId = t.TeamId

               WHERE    t.TeamName IN ( 'Team01', 'Team03' )

               GROUP BY t.TeamName

             ),

        cteTeam02Team04 ( TName, TotalPeople )

          AS ( SELECT   d.TeamName ,

                        COUNT(g.GamerId) AS TotalGamers

               FROM     Gamer g

                        JOIN Team d ON g.TeamId = d.TeamId

               WHERE    d.TeamName IN ( 'Team02', 'Team04' )

               GROUP BY d.TeamName

             )

    SELECT  \*

    FROM    cteTeam01Team03

    UNION

    SELECT  \*

    FROM    cteTeam02Team04;

GO -- Run the prvious command and begins new batch

Graphical user interface, application, table, Excel

Description automatically generated

================================================================

3. UpdatableCommonTableExpressions(CTE)

--=======================================================================================================

--T017\_03\_UpdatableCommonTableExpressions(CTE)

--=======================================================================================================

/\*

Updatable CommonTableExpressions(CTE)

1.

If CTE has only one based table,

then we may update the CommonTableExpressions(CTE).

2.

If CTE has many based tables,

and if UPDATE affects multiple base tables,

then it will return ERROR and terminates the UPDATE.

3.

If CTE has many based tables,

and if UPDATE affects only ONE base table,

then we may update the CommonTableExpressions(CTE).

But it might not work as we expected

\*/

3.1. If CTE has only one based table

--=======================================================================================================

--T017\_03\_01

--If CTE has only one based table,

--then we may update the CommonTableExpressions(CTE).

WITH    cteGamer

          AS ( SELECT   g.GamerId ,

                        g.Name ,

                        g.Gender

               FROM     Gamer g

             )

    SELECT  \*

    FROM    cteGamer

    WHERE   GamerId = 1;

GO -- Run the prvious command and begins new batch

--update CTE works as expected.

WITH    cteGamer2

          AS ( SELECT   g.GamerId ,

                        g.Name ,

                        g.Gender

               FROM     Gamer g

             )

    UPDATE  cteGamer2

    SET     cteGamer2.Gender += 'CteGamer2'

    WHERE   GamerId = 1;

GO -- Run the prvious command and begins new batch

SELECT  \*

FROM    Gamer

WHERE   GamerId = 1;

GO -- Run the prvious command and begins new batch

--Cean up

UPDATE  Gamer

SET     Gender = 'Male'

WHERE   GamerId = 1;

GO -- Run the prvious command and begins new batch

/\*

If CTE has only one based table,

then we may update the CommonTableExpressions(CTE).

\*/

Graphical user interface, application

Description automatically generated

3.2. If CTE has many based tables

--=======================================================================================================

--T017\_03\_02

--If CTE has many based tables,

--and if UPDATE affects only ONE base table,

WITH    cteGamerJoinTeam

          AS ( SELECT   g.GamerId ,

                        g.Name ,

                        g.Gender ,

                        t.TeamName

               FROM     Gamer g

                        JOIN Team t ON g.TeamId = t.TeamId

             )

    SELECT  \*

    FROM    cteGamerJoinTeam;

GO -- Run the prvious command and begins new batch

-- It works

WITH    cteGamerJoinTeam

          AS ( SELECT   g.GamerId ,

                        g.Name ,

                        g.Gender ,

                        t.TeamName

               FROM     Gamer g

                        JOIN Team t ON g.TeamId = t.TeamId

             )

    UPDATE  cteGamerJoinTeam

    SET     cteGamerJoinTeam.Gender += 'CteGamerJoinTeam'

    WHERE   GamerId = 1;

GO -- Run the prvious command and begins new batch

SELECT  \*

FROM    Gamer

WHERE   GamerId = 1;

--Clean up

UPDATE  Gamer

SET     Gender = 'Male'

WHERE   GamerId = 1;

SELECT  \*

FROM    Gamer

WHERE   GamerId = 1;

GO -- Run the prvious command and begins new batch

/\*

1.

If CTE has many based tables,

and if UPDATE affects only ONE base table,

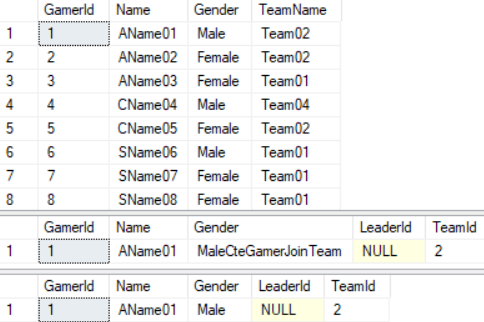
then we may update the CommonTableExpressions(CTE).

But it might not work as we expected

1.1.

In this case, it works as we expected.

\*/



3.3. If CTE has many based tables

--=======================================================================================================

--T017\_03\_03

--If CTE has many based tables,

--and if UPDATE affects multiple base tables,

--then it will return ERROR and terminates the UPDATE.

SELECT  \*

FROM    Gamer

WHERE   GamerId = 1;

SELECT  \*

FROM    Team;

GO -- Run the prvious command and begins new batch

WITH    cteGamerJoinTeam

          AS ( SELECT   g.GamerId ,

                        g.Name ,

                        g.Gender ,

                        t.TeamName

               FROM     dbo.Gamer g

                        JOIN Team t ON g.TeamId = t.TeamId

             )

    UPDATE  cteGamerJoinTeam

    SET     cteGamerJoinTeam.Gender += 'cteGamerJoinTeam' ,

            cteGamerJoinTeam.TeamName = 'Team03'

    WHERE   GamerId = 1;

GO -- Run the prvious command and begins new batch

SELECT  \*

FROM    Gamer

WHERE   GamerId = 1;

SELECT  \*

FROM    Team;

GO -- Run the prvious command and begins new batch

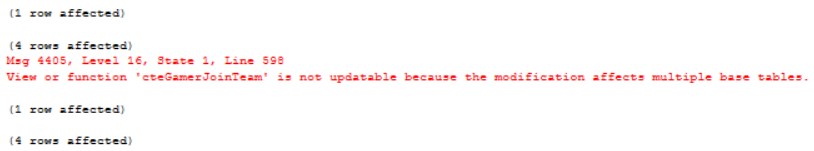
/\*

If CTE has many based tables,

and if UPDATE affects multiple base tables,

then it will return ERROR and terminates the UPDATE.

\*/



Graphical user interface, table

Description automatically generated

3.4. Incorrectly update

--=======================================================================================================

--T017\_03\_04

--\*\*Incorrectly update

--If CTE has many based tables,

--and if UPDATE affects only ONE base table,

--then we may update the CommonTableExpressions(CTE).

--But it might not work as we expected

SELECT  \*

FROM    Gamer

WHERE   GamerId = 1;

SELECT  \*

FROM    Team;

GO -- Run the prvious command and begins new batch

WITH    cteGamerJoinTeam

          AS ( SELECT   g.GamerId ,

                        g.Name ,

                        g.Gender ,

                        t.TeamName

               FROM     dbo.Gamer g

                        JOIN Team t ON g.TeamId = t.TeamId

             )

    UPDATE  cteGamerJoinTeam

    SET     cteGamerJoinTeam.TeamName = 'Team03'

    WHERE   GamerId = 1;

SELECT  \*

FROM    Gamer

WHERE   GamerId = 1;

SELECT  \*

FROM    Team;

--Clean up

UPDATE  Team

SET     TeamName = 'Team02'

WHERE   TeamId = 2;

GO -- Run the prvious command and begins new batch

/\*

1.

If CTE has many based tables,

and if UPDATE affects only ONE base table,

then we may update the CommonTableExpressions(CTE).

But it might not work as we expected

2.

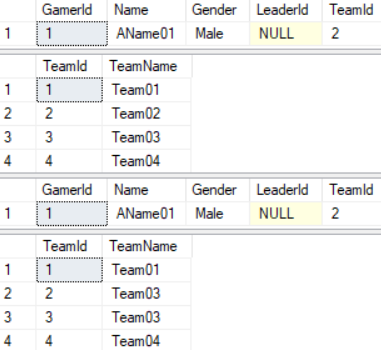
It has the same result as you run the following

UPDATE  Team

SET     TeamName = 'Team03'

WHERE   TeamId = 2;

\*/



4. RecursiveCommonTableExpressions(CTE)

--=======================================================================================================

--T017\_04\_RecursiveCommonTableExpressions(CTE)

--=======================================================================================================

/\*

AName01

  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  |          |           |

AName02    CName04     SName06

  |          |\_\_\_\_\_\_\_\_\_

  |          |        |

AName03    SName07   SName08

  |

CName05

\*/

4.1. Recursive CommonTableExpressions(CTE)

--=======================================================================================================

--T017\_04\_01

--Recursive CommonTableExpressions(CTE)

----------------------------------------------------------------------------------------

--T017\_04\_01\_01

SELECT  g.Name ,

        ISNULL(g2.Name, 'Boss') AS [Leader Name]

FROM    Gamer g

        LEFT JOIN Gamer g2 ON g.LeaderId = g2.GamerId;

GO -- Run the prvious command and begins new batch

Graphical user interface, table, Excel

Description automatically generated

----------------------------------------------------------------------------------------

--T017\_04\_01\_02

/\*

AName01

  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  |          |           |

AName02    CName04     SName06

  |          |\_\_\_\_\_\_\_\_\_

  |          |        |

AName03    SName07   SName08

  |

CName05

This query will return the Orgination Level.

E.g.

[Level]=1 contains AName01

[Level]=2 contains AName02, CName04, SName06

[Level]=3 contains AName03, SName07, SName08

[Level]=4 contains CName05

\*/

WITH    cteGamer ( GId, GName, LId, [Level] )

          AS ( --Anchor Member

               SELECT   g.GamerId ,

                        g.Name ,

                        g.LeaderId ,

                        1

               FROM     Gamer g

               WHERE    g.LeaderId IS NULL

               UNION ALL

                       --Recursive Member

               SELECT   g.GamerId ,

                        g.Name ,

                        g.LeaderId ,

                        cteG.[Level] + 1

               FROM     Gamer g

                        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

             )

       --\*\*The Changes here

    SELECT  cteG.GName AS Gamer ,

            ISNULL(cteG2.GName, 'Boss') AS Leader ,

            cteG.[Level]

    FROM    cteGamer cteG

            LEFT JOIN cteGamer cteG2 ON cteG.LId = cteG2.GId;

GO -- Run the prvious command and begins new batch

Graphical user interface, application, table

Description automatically generated with medium confidence

----------------------------------------------------------------------------------------

--T017\_04\_01\_03

WITH    cteGamer ( GId, GName, LId, [Level] )

          AS ( --Anchor Member

               SELECT   g.GamerId ,

                        g.Name ,

                        g.LeaderId ,

                        1

               FROM     Gamer g

               WHERE    g.LeaderId IS NULL

               UNION ALL

                       --Recursive Member

               SELECT   g.GamerId ,

                        g.Name ,

                        g.LeaderId ,

                        cteG.[Level] + 1

               FROM     Gamer g

                        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

             )

       --\*\*The Changes here

    SELECT  \*

    FROM    cteGamer cteG

            LEFT JOIN cteGamer cteG2 ON cteG.LId = cteG2.GId;

GO -- Run the prvious command and begins new batch

Table

Description automatically generated

/\*

0.

AName01

  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  |          |           |

AName02    CName04     SName06

  |          |\_\_\_\_\_\_\_\_\_

  |          |        |

AName03    SName07   SName08

  |

CName05

----------------------------------

0.1.

----The 1st select query is Anchor Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        1

--FROM     Gamer g

--WHERE    g.LeaderId IS NULL

--UNION ALL

--------------------------------

----The 2nd select query is Recursive Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        cteG.[Level] + 1

--FROM     Gamer g

--        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

How does the recursive CTE execute?

Step1: Execute the anchor member and get result R0

Step2: Execute the recursive member by using R0 as input and output result R1

Step3: Execute the recursive member by using R1 as input and output result R2

Step4: Recursion goes on until the recursive member output result is NULL

Step5: Finally apply UNION ALL on all the results to produce the final output

----------------------------------------------------------------------

1.

The cteGamer contains 2 queries.

1.1.

The 1st select query of cteGamer,

it gets the 'Boss' whose 'LeaderId' is null.

and Set [Level] of Boss to 1.

In this case, ID=1 is the boss.

The 1st select query will be completed in 1st round of Recursive cteGamer

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        1

--FROM     Gamer g

--WHERE    g.LeaderId IS NULL

-----------------------------------------------

1.2.

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        cteG.[Level] + 1

--FROM     Gamer g

--        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

The 2nd select query of cteGamer,

It will set [Level] of the rest of people recursively under boss

and loop throgh the hierarchy.

(cteG.[Level] + 1)   means (his Leader level + 1).

Thus, the 2nd select query will start the 2st round of Recursive cteGamer

until the end of recursive.

In this case, we know ID=1 is the boss.

2nd select query will start from id=2 then id=3 then id=4 ... .

------------------------------

1.2.1.

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        cteG.[Level] + 1

--FROM     Gamer g

--        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

--WHERE    g.LeaderId = 2   or   4   or  6

The 2nd round of Recursive cteGamer will get all sub-member of the id=1 boss.

In this case, Id=2, 4, 6 are the sub-members of the id=1 boss.

(cteG.[Level] + 1)   means (his Leader level + 1).

Thus, (the cteG.[Level] of Id=2, 4, 6)  will be  (  (their Leader id=1 Boss level which is 1) + 1).

Therefore, (the cteG.[Level] of Id=2, 4, 6) will be 2.

------------------------------

1.2.2.

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        cteG.[Level] + 1

--FROM     Gamer g

--        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

--WHERE    g.LeaderId = 3

The 3rd round of Recursive cteGamer will get all sub-member of the id=2 leader.

In this case, Id=3 is the sub-member of the id=2 Leader.

(cteG.[Level] + 1)   means (his Leader level + 1).

Thus, (the cteG.[Level] of Id=3)  will be  (  (their Leader id=2 leader level which is 2) + 1).

Therefore, (the cteG.[Level] of Id=3) will be 3.

------------------------------

1.2.3.

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        cteG.[Level] + 1

--FROM     Gamer g

--        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

--WHERE    g.LeaderId = 7   or   8

The 3rd round of Recursive cteGamer will get all sub-member of the id=4 leader.

In this case, Id=7,8 are the sub-members of the id=4 Leader.

(cteG.[Level] + 1)   means (his Leader level + 1).

Thus, (the cteG.[Level] of Id=7,8)  will be  (  (their Leader id=4 leader level which is 2) + 1).

Therefore, (the cteG.[Level] of Id=7,8) will be 3.

------------------------------

1.2.4.

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId ,

--        cteG.[Level] + 1

--FROM     Gamer g

--        JOIN cteGamer cteG ON g.LeaderId = cteG.GId

--WHERE    g.LeaderId = 5

The 4th round of Recursive cteGamer will get all sub-member of the id=3 leader.

In this case, Id=5 is the sub-member of the id=3 Leader.

(cteG.[Level] + 1)   means (his Leader level + 1).

Thus, (the cteG.[Level] of Id=5)  will be  (  (their Leader id=3 leader level which is 3) + 1).

Therefore, (the cteG.[Level] of Id=5) will be 4.

\*/

================================================

5. GetOrganizationHierarchy

--=======================================================================================================

--T017\_05\_GetOrganizationHierarchy

--=======================================================================================================

--=======================================================================================================

--T017\_05\_01

------------------------------------------------------------------------------

--T017\_05\_01\_01

SELECT  g.Name ,

        ISNULL(g2.Name, 'Boss') AS [Leader Name]

FROM    Gamer g

        LEFT JOIN Gamer g2 ON g.LeaderId = g2.GamerId

WHERE   g.GamerId = 5;

GO -- Run the prvious command and begins new batch



------------------------------------------------------------------------------

--T017\_05\_01\_02

/\*

AName01

  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  |          |           |

AName02    CName04     SName06

  |          |\_\_\_\_\_\_\_\_\_

  |          |        |

AName03    SName07   SName08

  |

CName05

Stored procedure spGetLeaders and spGetLeaders2

will take an ID INT as input,

Then return its leaders' information.

E.g.

--EXEC spGetLeaders 5;

will return information of ID=5, ID=3, ID2, ID1.

E.g.

--EXEC spGetLeaders 7;

will return information of ID=7, ID=4, ID1.

\*/

GO -- Run the prvious command and begins new batch

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.ROUTINES

              WHERE     ROUTINE\_TYPE = 'PROCEDURE'

                        AND LEFT(ROUTINE\_NAME, 3) NOT IN ( 'sp\_', 'xp\_', 'ms\_' )

                        AND SPECIFIC\_NAME = 'spGetLeaders' ) )

    BEGIN

        DROP PROCEDURE spGetLeaders;

    END;

GO -- Run the previous command and begins new batch

CREATE PROC spGetLeaders ( @Id INT )

AS

    BEGIN

        WITH    cteGamer

                  AS ( --Anchor Member

                       SELECT   g.GamerId ,

                                g.Name ,

                                g.LeaderId

                       FROM     Gamer g

                       WHERE    GamerId = @Id

                       UNION ALL

                                    --Recursive Member

                       SELECT   g.GamerId ,

                                g.Name ,

                                g.LeaderId

                       FROM     Gamer g

                                JOIN cteGamer cteG ON g.GamerId = cteG.LeaderId

                     )

            --\*\*The Changes here

            SELECT  cteG1.Name ,

                    ISNULL(cteG2.Name, 'No Boss') AS LeaderName

            FROM    cteGamer cteG1

                    LEFT JOIN cteGamer cteG2 ON cteG1.LeaderId = cteG2.GamerId;

    END;

GO -- Run the prvious command and begins new batch

EXEC spGetLeaders 5;

EXEC spGetLeaders 7;

GO -- Run the prvious command and begins new batch

Graphical user interface, application, table, Excel

Description automatically generated

------------------------------------------------------------------------------

--T017\_05\_01\_03

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.ROUTINES

              WHERE     ROUTINE\_TYPE = 'PROCEDURE'

                        AND LEFT(ROUTINE\_NAME, 3) NOT IN ( 'sp\_', 'xp\_', 'ms\_' )

                        AND SPECIFIC\_NAME = 'spGetLeaders2' ) )

    BEGIN

        DROP PROCEDURE spGetLeaders2;

    END;

GO -- Run the previous command and begins new batch

CREATE PROC spGetLeaders2 ( @Id INT )

AS

    BEGIN

        WITH    cteGamer

                  AS ( --Anchor Member

                       SELECT   g.GamerId ,

                                g.Name ,

                                g.LeaderId

                       FROM     Gamer g

                       WHERE    GamerId = @Id

                       UNION ALL

                                    --Recursive Member

                       SELECT   g.GamerId ,

                                g.Name ,

                                g.LeaderId

                       FROM     Gamer g

                                JOIN cteGamer cteG ON g.GamerId = cteG.LeaderId

                     )

            --\*\*The Changes here

            SELECT \*

            FROM    cteGamer cteG1

                    LEFT JOIN cteGamer cteG2 ON cteG1.LeaderId = cteG2.GamerId;

    END;

GO -- Run the prvious command and begins new batch

EXEC spGetLeaders2 5;

EXEC spGetLeaders2 7;

GO -- Run the prvious command and begins new batch

Graphical user interface, application, table

Description automatically generated

/\*

0.

AName01

  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  |          |           |

AName02    CName04     SName06

  |          |\_\_\_\_\_\_\_\_\_

  |          |        |

AName03    SName07   SName08

  |

CName05

----------------------------------

0.1.

----The 1st select query is Anchor Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = @Id

--UNION ALL

-------------------------------------------------

----The 2nd select query is Recursive Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--        JOIN cteGamer cteG ON g.GamerId = cteG.LeaderId

How does the recursive CTE execute?

Step1: Execute the anchor member and get result R0

Step2: Execute the recursive member by using R0 as input and output result R1

Step3: Execute the recursive member by using R1 as input and output result R2

Step4: Recursion goes on until the recursive member output result is NULL

Step5: Finally apply UNION ALL on all the results to produce the final output

----------------------------------------------------------------------

1.

--EXEC spGetLeaders 5;

This will output as following.

--Name    LeaderName

--CName05 AName03

--AName03 AName02

--AName02 AName01

--AName01 No Boss

---------------------------------

1.1.

----Anchor Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = 5

The 1st round of Recursive cteGamer will get the parents-member of the id=5.

In this case, Id=3 is the parents-member of the id=5.

--AName01 No Boss

---------------------------------

1.2.

----Recursive Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = 3

The 2nd round of Recursive cteGamer will get the parents-member of the id=3.

In this case, Id=2 is the parents-member of the id=3.

--AName01 No Boss

---------------------------------

1.3.

----Recursive Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = 2

The 3rd round of Recursive cteGamer will get the parents-member of the id=2.

In this case, Id=1 is the parents-member of the id=2.

--AName01 No Boss

---------------------------------

1.4.

----Recursive Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = 1

The 4th round of Recursive cteGamer will get the parents-member of the id=1.

In this case, nobody is the parents-member of the id=1.

--ISNULL(cteG2.Name, 'No Boss') AS LeaderName

Thus, it will return 'No Boss'

----------------------------------------------------------------------

2.

--EXEC spGetLeaders 7;

This will output as following.

--Name    LeaderName

--SName07 CName04

--CName04 AName01

--AName01 No Boss

--AName01 No Boss

---------------------------------

2.1.

----Anchor Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = 7

The 1st round of Recursive cteGamer will get the parents-member of the id=7.

In this case, Id=4 is the parents-member of the id=7.

--AName01 No Boss

---------------------------------

2.2.

----Recursive Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = 4

The 2nd round of Recursive cteGamer will get the parents-member of the id=4.

In this case, Id=1 is the parents-member of the id=4.

--AName01 No Boss

---------------------------------

2.3.

----Recursive Member

--SELECT   g.GamerId ,

--        g.Name ,

--        g.LeaderId

--FROM     Gamer g

--WHERE    GamerId = 1

The 3rd round of Recursive cteGamer will get the parents-member of the id=1.

In this case, nobody is the parents-member of the id=1.

--ISNULL(cteG2.Name, 'No Boss') AS LeaderName

Thus, it will return 'No Boss'

\*/

================================================

6. Clean up

--=======================================================================================================

--T017\_06\_Clean up

--=======================================================================================================

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Gamer' ) )

    BEGIN

        TRUNCATE TABLE Gamer;

        DROP TABLE Gamer;

    END;

GO -- Run the previous command and begins new batch

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Team' ) )

    BEGIN

        TRUNCATE TABLE Team;

        DROP TABLE Team;

    END;

GO -- Run the previous command and begins new batch

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'vwGamerCount' ) )

    BEGIN

        DROP VIEW vwGamerCount;

    END;

GO -- Run the previous command and begins new batch

IF OBJECT\_ID('tempdb..#TempGamerCount') IS NOT NULL

    BEGIN

        TRUNCATE TABLE #TempGamerCount;

        DROP TABLE #TempGamerCount;

    END;

GO -- Run the previous command and begins new batch

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.ROUTINES

              WHERE     ROUTINE\_TYPE = 'PROCEDURE'

                        AND LEFT(ROUTINE\_NAME, 3) NOT IN ( 'sp\_', 'xp\_', 'ms\_' )

                        AND SPECIFIC\_NAME = 'spGetLeaders' ) )

    BEGIN

        DROP PROCEDURE spGetLeaders;

    END;

GO -- Run the previous command and begins new batch

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.ROUTINES

              WHERE     ROUTINE\_TYPE = 'PROCEDURE'

                        AND LEFT(ROUTINE\_NAME, 3) NOT IN ( 'sp\_', 'xp\_', 'ms\_' )

                        AND SPECIFIC\_NAME = 'spGetLeaders2' ) )

    BEGIN

        DROP PROCEDURE spGetLeaders2;

    END;

GO -- Run the previous command and begins new batch