■SQLのWITH句について

WITH句は、SQLクエリの中で一時的な名前付きの結果セット、つまり **共通テーブル式 (CTE: Common Table Expression)** を定義するために使用されます。CTEは、複雑なクエリをより読みやすく、管理しやすくするために非常に役立つ機能です。

**WITH句の基本的な構文**

WITH

CTE名1 AS (

-- CTE名1 のクエリ定義

SELECT 列1, 列2, ...

FROM テーブル1

WHERE 条件

),

CTE名2 AS (

-- CTE名2 のクエリ定義 (CTE名1を参照することも可能)

SELECT 列A, 列B, ...

FROM CTE名1

WHERE 条件2

)

-- メインクエリ (CTE名1, CTE名2 を参照可能)

SELECT 列X, 列Y, ...

FROM CTE名2

WHERE 条件3;

**構文のポイント**

* **WITH** で始まり、その後に一つ以上のCTE定義が続きます。
* **CTE名**: CTEにつける名前です。これは一時的なテーブル名としてクエリ内で使用できます。
* **AS ( ... )**: AS キーワードの後に括弧 () で囲んで、CTEのクエリ定義 (SELECT文) を記述します。
* **カンマ区切り**: 複数のCTEを定義する場合は、CTE定義をカンマ , で区切ります。
* **メインクエリ**: WITH句の後に、通常のSELECT文などのメインクエリを記述します。このメインクエリの中で、定義したCTEをテーブルのように参照できます。

**WITH句を使うメリット**

1. **クエリの可読性向上**:
   * 複雑なクエリを複数の小さな論理的なステップに分割できます。
   * 各CTEに意味のある名前をつけることで、クエリ全体の意図が理解しやすくなります。
   * 長くてネストが深くなりがちなサブクエリを避けることができます。
2. **クエリの再利用性**:
   * CTEを一度定義すれば、同じWITH句の中で何度も参照できます。
   * 同じロジックを繰り返し記述する必要がなくなり、コードの重複を減らせます。
3. **複雑なクエリの構造化**:
   * 複雑なクエリを段階的に構築できます。
   * CTEを連鎖させることで、前のCTEの結果を次のCTEで利用するといった処理が容易になります。
   * 再帰的なクエリ (階層構造データの処理など) を記述することも可能です (再帰CTE)。

**具体的な使用例**

**例1: 単純な集計クエリを分かりやすくする**

-- WITH句なし (サブクエリを使用)

SELECT

顧客名,

注文数,

平均注文金額

FROM (

SELECT

顧客ID,

COUNT(\*) AS 注文数,

AVG(注文金額) AS 平均注文金額

FROM 注文テーブル

GROUP BY 顧客ID

) AS 顧客別注文集計

JOIN 顧客テーブル ON 顧客別注文集計.顧客ID = 顧客テーブル.顧客ID

WHERE 注文数 > 10;

-- WITH句あり (CTEを使用)

WITH 顧客別注文集計 AS (

SELECT

顧客ID,

COUNT(\*) AS 注文数,

AVG(注文金額) AS 平均注文金額

FROM 注文テーブル

GROUP BY 顧客ID

)

SELECT

顧客テーブル.顧客名,

顧客別注文集計.注文数,

顧客別注文集計.平均注文金額

FROM 顧客別注文集計

JOIN 顧客テーブル ON 顧客別注文集計.顧客ID = 顧客テーブル.顧客ID

WHERE 注文数 > 10;

上記の例では、WITH句を使った方が、顧客別の注文数を集計する部分 (顧客別注文集計 CTE) と、最終的な結果を取得する部分が明確に分かれており、クエリの意図が理解しやすくなっています。

**例2: 複数のCTEを組み合わせる**

WITH

売上集計 AS (

SELECT

店舗ID,

SUM(売上金額) AS 総売上

FROM 売上テーブル

GROUP BY 店舗ID

),

店舗情報 AS (

SELECT

店舗ID,

店舗名,

所在地

FROM 店舗テーブル

)

SELECT

店舗情報.店舗名,

店舗情報.所在地,

売上集計.総売上

FROM 売上集計

JOIN 店舗情報 ON 売上集計.店舗ID = 店舗情報.店舗ID

ORDER BY 総売上 DESC;

この例では、売上集計 CTE と 店舗情報 CTE の2つのCTEを定義し、それらを結合して店舗別の売上情報を取得しています。このように、複数のCTEを組み合わせることで、複雑なデータ処理を段階的に行うことができます。

**例3: 再帰CTE (階層構造データの処理)**

WITH RECURSIVE 組織階層 AS (

-- アンカーメンバ (再帰の開始点)

SELECT

従業員ID,

上司ID,

従業員名,

役職,

1 AS 階層レベル

FROM 従業員テーブル

WHERE 上司ID IS NULL -- 最上位の従業員 (社長など)

UNION ALL

-- 再帰メンバ (再帰的な処理)

SELECT

e.従業員ID,

e.上司ID,

e.従業員名,

e.役職,

oh.階層レベル + 1 AS 階層レベル

FROM 従業員テーブル e

JOIN 組織階層 oh ON e.上司ID = oh.従業員ID -- 上司IDで結合

)

SELECT

階層レベル,

従業員名,

役職

FROM 組織階層

ORDER BY 階層レベル, 従業員ID;

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).SQL

再帰CTEは少し高度な機能ですが、WITH RECURSIVE を使うことで、階層構造を持つデータを処理することができます。上記の例では、従業員テーブルから組織階層を再帰的に辿り、階層レベルと従業員情報を取得しています。

**注意点**

* **スコープ**: CTEは、WITH句が定義されたクエリ内でのみ有効な一時的なものです。他のクエリからCTEを参照することはできません。
* **パフォーマンス**: CTE自体は、必ずしもクエリのパフォーマンスを向上させるものではありません。むしろ、複雑なCTEを多用すると、パフォーマンスが低下する可能性もあります。適切なインデックスの設計やクエリの最適化が必要です。
* **データベースの種類**: WITH RECURSIVE 構文は、すべてのデータベースでサポートされているわけではありません。使用するデータベースが再帰CTEをサポートしているか確認が必要です。

**まとめ**

WITH句は、SQLクエリの可読性と保守性を向上させる強力なツールです。複雑なクエリを分割したり、同じ結果セットを再利用したり、階層構造データを処理したりする際に非常に役立ちます。積極的にWITH句を活用して、より分かりやすく効率的なSQLクエリを作成してみてください。