## ＳＱＬの世界へようこそ！

## ウィンドウ関数、ビュー、ＣＴＥをマスターしよう

この教材では、SQLの強力な機能である「ウィンドウ関数」「ビュー」「CTE（共通テーブル式）」について学びます。これらの機能を使いこなせば、データ分析やレポート作成の効率が格段に向上します！

### 1. はじめに：それぞれの役割を理解しよう

まずは、それぞれの機能がどのような役割を持っているのか、ざっくりと理解しましょう。

* **ウィンドウ関数:**
  + **例え:** テストの成績表で、クラス全体の平均点だけでなく、「自分の順位」や「平均点との差」も各生徒の行に表示する。
  + **役割:** 各行のデータはそのままに、関連する行のグループ（ウィンドウ）に基づいた計算結果を新しい列として追加する。
  + **得意なこと:** ランキング、移動平均、累積合計など、行ごとの詳細情報を保持したまま計算を行う。
* **ビュー:**
  + **例え:** よく使うWebサイトのブックマーク（ショートカット）。
  + **役割:** よく使うSELECT文（クエリ）に名前を付けてデータベースに保存しておく。
  + **得意なこと:** 複雑なクエリを隠蔽し、簡単な名前でアクセスできるようにする。データの再利用、セキュリティ向上。
* **CTE（共通テーブル式）:**
  + **例え:** 料理のレシピで、下ごしらえやソース作りなどの各ステップを「小さなレシピ」として定義し、最後に組み合わせる。
  + **役割:** SELECT文の前に、一時的な名前付きの結果セット（テーブルのようなもの）を定義する。
  + **得意なこと:** 複雑なクエリを論理的な単位に分割し、可読性・保守性を向上させる。クエリ内での再利用、再帰処理。

### 2. ウィンドウ関数：データを見ながら計算！

ウィンドウ関数は、各行に対して、関連する行のグループ（ウィンドウ）を使った計算を行います。

#### 2.1 基本の形と構成要素

SELECT

列1,

列2,

ウィンドウ関数(列3) OVER (PARTITION BY 列4 ORDER BY 列5 ROWS/RANGE ...) AS 新しい列名

FROM

テーブル名;

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

* **ウィンドウ関数(列3)**: 実行するウィンドウ関数を指定します。RANK(), AVG(), SUM(), ROW\_NUMBER() など、様々な種類があります（後述）。
* **OVER()句**: ウィンドウを定義します。
  + **PARTITION BY 列4** (オプション): どの列でデータをグループ分けするかを指定します。例えば、PARTITION BY 部署ID とすると、部署ごとにウィンドウが作成されます。
  + **ORDER BY 列5** (オプション): ウィンドウ内の行をどの列で並べ替えるかを指定します。例えば、ORDER BY 給与 DESC とすると、給与の高い順に並べ替えられます。
  + **フレーム指定** (オプション): ROWS または RANGE キーワードを使って、ウィンドウの範囲をさらに細かく指定します。

#### 2.2 ウィンドウ関数の種類

ウィンドウ関数は、大きく分けて3つのカテゴリに分類できます。

1. **順位付け関数:** 行に順位を付ける
   * ROW\_NUMBER(): 重複なしの連番を振ります。
   * RANK(): 同じ値には同じ順位を付け、次の順位は飛ばします（1, 2, 2, 4, ...）。
   * DENSE\_RANK(): 同じ値には同じ順位を付け、次の順位は飛ばしません（1, 2, 2, 3, ...）。
   * NTILE(n): 行をn個のグループに分け、各行が属するグループ番号を返します。
2. **集計関数:** ウィンドウ内の値を集計する
   * SUM(): 合計
   * AVG(): 平均
   * MIN(): 最小値
   * MAX(): 最大値
   * COUNT(): 件数
3. **値取得関数:** ウィンドウ内の特定の位置の行の値を取得する
   * LAG(列, n, default): 現在の行からn行前の値を取得します。nを省略すると1、defaultは値がない場合のデフォルト値です。
   * LEAD(列, n, default): 現在の行からn行後の値を取得します。
   * FIRST\_VALUE(列): ウィンドウ内の最初の行の値を取得します。
   * LAST\_VALUE(列): ウィンドウ内の最後の行の値を取得します。（注意点あり、後述）
   * NTH\_VALUE(列, n): ウィンドウ内のn番目の行の値を取得します。

#### 2.3 フレーム指定：ウィンドウの範囲を操る

フレーム指定を使うと、ウィンドウ内の計算対象となる行の範囲を詳細に制御できます。

* **ROWS:** 行数で範囲を指定します。

ROWS BETWEEN start AND end

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

* + start と end には以下を指定できます。
    - UNBOUNDED PRECEDING: ウィンドウの先頭
    - UNBOUNDED FOLLOWING: ウィンドウの末尾
    - CURRENT ROW: 現在の行
    - n PRECEDING: 現在の行からn行前
    - n FOLLOWING: 現在の行からn行後
* **RANGE:** 値の範囲で指定します（ORDER BY が必須）。

RANGE BETWEEN start AND end

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

* + ORDER BY で指定した列の値に基づいて範囲を決定します。
  + start と end には以下を指定できます。
    - UNBOUNDED PRECEDING: ウィンドウの先頭の値
    - UNBOUNDED FOLLOWING: ウィンドウの末尾の値
    - CURRENT ROW: 現在の行の値
    - n PRECEDING: 現在の行の値からnを引いた値（数値型の場合）
    - n FOLLOWING: 現在の行の値にnを足した値（数値型の場合）
    - INTERVAL 'n' DAY/MONTH/YEAR PRECEDING/FOLLOWING：日付/時刻型の場合

**ROWS と RANGE の違い:**

* ROWS は物理的な行の位置に基づいて範囲を決定します。
* RANGE は ORDER BY で指定した列の値に基づいて範囲を決定します。
* ORDER BY 列に重複する値がない場合は、ROWS と RANGE の結果は同じになることが多いですが、重複がある場合は結果が異なる可能性があります。

#### 2.4 具体例で理解を深める

社員テーブルと部署テーブルを使って、さまざまなウィンドウ関数の使用例を見てみましょう。

**[社員テーブル (Shain)]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **社員ID** | **氏名** | **部署ID** | **給与** | **入社日** |
| 1 | 山田 | 10 | 600 | 2022-04-01 |
| 2 | 田中 | 10 | 700 | 2022-05-01 |
| 3 | 鈴木 | 20 | 500 | 2022-06-01 |
| 4 | 佐藤 | 20 | 550 | 2022-07-01 |
| 5 | 伊藤 | 10 | 800 | 2022-08-01 |
| 6 | 加藤 | 20 | 650 | 2022-09-01 |
| 7 | 渡辺 | 10 | 750 | 2022-10-01 |

**[部署テーブル (Busho)]**

|  |  |
| --- | --- |
| **部署ID** | **部署名** |
| 10 | 営業部 |
| 20 | 開発部 |

**例1：部署ごとの平均給与と各社員の給与の比較**

SELECT

s.氏名,

b.部署名,

s.給与,

AVG(s.給与) OVER (PARTITION BY b.部署ID) AS 部署平均給与,

s.給与 - AVG(s.給与) OVER (PARTITION BY b.部署ID) AS 給与差

FROM

社員 s

JOIN

部署 b ON s.部署ID = b.部署ID;

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

**例2：部署ごとの給与ランキング**

SELECT

s.氏名,

b.部署名,

s.給与,

RANK() OVER (PARTITION BY b.部署ID ORDER BY s.給与 DESC) AS 部署内順位

FROM

社員 s

JOIN

部署 b ON s.部署ID = b.部署ID;

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

**例3：社員ID順の連番**

SELECT

ROW\_NUMBER() OVER (ORDER BY 社員ID) AS 連番,

社員ID,

氏名

FROM

社員;

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

**例4：過去3ヶ月の給与合計（自分自身を含む）**

SELECT

氏名,

入社日,

給与,

SUM(給与) OVER (ORDER BY 入社日 ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS 過去3ヶ月給与合計

FROM

社員;

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

**例5：1つ前と1つ後の社員の給与**

SELECT

氏名,

入社日,

給与,

LAG(給与, 1, 0) OVER (ORDER BY 入社日) AS 1つ前の給与,

LEAD(給与, 1, 0) OVER (ORDER BY 入社日) AS 1つ後の給与

FROM

社員;

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

**例6：部署内で最も給与が高い/低い社員の給与**

SELECT

氏名,

部署名,

給与,

FIRST\_VALUE(給与) OVER (PARTITION BY 部署名 ORDER BY 給与 DESC) AS 部署内最高給与,

LAST\_VALUE(給与) OVER (PARTITION BY 部署名 ORDER BY 給与 DESC

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS 部署内最低給与

FROM

社員 s

JOIN

部署 b ON s.部署ID = b.部署ID;

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

**例7: 各社員を給与額によって3つのグループに分類**

SELECT

氏名,

給与,

NTILE(3) OVER (ORDER BY 給与 DESC) AS 給与グループ

FROM 社員

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

#### 2.5 ウィンドウ関数の注意点

* ウィンドウ関数はSELECT句とORDER BY句でのみ使用できます。
* OVER()句の指定によっては、パフォーマンスに影響が出る可能性があります。特に、大きなテーブルで複雑なフレーム指定（特にRANGE）を行う場合は注意が必要です。
* LAST\_VALUE() を使う場合は、ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING のようなフレーム指定を明示的に行うのが安全です。
* データベースシステムによっては、サポートされていないウィンドウ関数や機能があります。

### 3. ビュー：クエリに名前を付けて保存！

ビューは、よく使うクエリに名前を付けてデータベースに保存する機能です。

#### 3.1 ビューの作成 (CREATE VIEW)

CREATE [OR REPLACE] VIEW ビュー名 [ (列名1, 列名2, ...) ]

AS

SELECT ... -- ビューの定義となるSELECT文

[WITH CHECK OPTION];

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

* **CREATE VIEW ビュー名**: 新しいビューを作成します。
* **OR REPLACE** (オプション): 同じ名前のビューが既に存在する場合、既存のビューを置き換えます。
* **(列名1, 列名2, ...)** (オプション): ビューの列名を指定します。
* **AS SELECT ...**: ビューの定義となるSELECT文を記述します。
* **WITH CHECK OPTION** (オプション): ビューを通じて行われる更新が、ビューの定義のWHERE句の条件を満たすように強制します。

#### 3.2 ビューの使用 (SELECT)

ビューは、テーブルと同じように SELECT 文で参照できます。

SELECT 列名1, 列名2, ... FROM ビュー名 WHERE 条件 ...;

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

#### 3.3 ビューの更新 (INSERT, UPDATE, DELETE)

単純なビュー（単一のテーブルに対するSELECTで、集計関数やウィンドウ関数などを含まない）は、更新可能な場合があります。ビューに対する更新は、元のテーブルに反映されます。

**注意点:**

* ビューが更新可能かどうかは、ビューの定義によります。
* WITH CHECK OPTION を指定したビューでは、条件を満たさない更新は拒否されます。

#### 3.4 ビューの管理

* **変更:** ALTER VIEW (データベースシステムによっては未サポート)
* **削除:** DROP VIEW
* **情報確認:** データベースシステム固有のコマンドやシステムテーブルを使用

#### 3.5 マテリアライズドビュー

クエリの結果を物理的に保存するビューです。

* **メリット:** 高速なアクセス
* **デメリット:** データ更新は手動または自動で行う必要がある
* **注意:** すべてのデータベースシステムでサポートされているわけではない

#### 3.6 ビューのメリット

* **再利用性:** よく使うクエリを毎回書く必要がなくなる。
* **簡潔性:** 複雑なクエリを隠蔽し、シンプルな名前でアクセスできる。
* **セキュリティ:** 特定のユーザーに、特定の列や行のみを公開できる。
* **保守性:** 元のテーブルの構造が変更されても、ビューの定義を修正するだけで済む場合がある。
* **データの抽象化**
* **データの仮想分割**

#### 3.7 具体例

**例1：営業部社員ビュー**

CREATE VIEW 営業部社員 AS

SELECT 社員ID, 氏名, 給与

FROM 社員

WHERE 部署ID = (SELECT 部署ID FROM 部署 WHERE 部署名 = '営業部');

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

**例2：高給与社員ビュー（WITH CHECK OPTION付き）**

CREATE OR REPLACE VIEW 高給与社員 (社員番号, 氏名, 部署名, 給与)

AS

SELECT

s.社員ID,

s.氏名,

b.部署名,

s.給与

FROM

社員 s

JOIN

部署 b ON s.部署ID = b.部署ID

WHERE

s.給与 >= 8000000

WITH CHECK OPTION;

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

### 4. CTE（共通テーブル式）：クエリを分割して整理！

CTEは、SELECT文の前に、一時的な名前付きの結果セット（テーブルのようなもの）を定義する機能です。

#### 4.1 CTEの構文

WITH CTE名 [(列名1, 列名2, ...)] AS (

SELECT ... -- CTEの定義となるSELECT文

)

SELECT ... -- メインクエリ（CTEを参照できる）

;

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

* **WITH CTE名**: CTEを定義します。
* **(列名1, 列名2, ...)** (オプション): CTEの列名を指定します。
* **AS ( SELECT ... )**: CTEの定義となるSELECT文を記述します。
* **SELECT ...**: メインクエリ。CTEをテーブルのように参照できます。

#### 4.2 再帰CTE (Recursive CTE)

CTE自身を参照するCTEです。階層構造を持つデータを処理する際に便利です。

WITH RECURSIVE CTE名 (列名リスト) AS (

-- ベースケース（再帰の初期値）

SELECT ...

UNION ALL

-- 再帰部分（CTE自身を参照）

SELECT ...

FROM CTE名

WHERE ...

)

SELECT ... -- メインクエリ

FROM CTE名;

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

* **WITH RECURSIVE**: 再帰CTEを定義することを宣言します。
* **ベースケース:** 再帰の初期値となるSELECT文。
* **UNION ALL**: ベースケースと再帰部分の結果を結合します。
* **再帰部分:** CTE自身を参照するSELECT文。
* **WHERE句 (再帰部分):** 再帰を停止する条件を指定します。

#### 4.3 複数のCTEの組み合わせ

複数のCTEを定義し、それらを組み合わせて使うことができます。

WITH CTE1 AS (

SELECT ...

), CTE2 AS (

SELECT ...

), CTE3 AS (

SELECT ...

)

SELECT ...

FROM CTE1

JOIN CTE2 ON ...

WHERE ...

;

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

#### 4.4 CTEのメリット

* **可読性:** 複雑なクエリを論理的な単位に分割し、理解しやすくします。
* **再利用性（クエリ内）:** 同じCTEを、定義したクエリの中で複数回参照できます。
* **再帰処理:** WITH RECURSIVE を使って、再帰的なクエリを記述できます。

#### 4.5 CTEの注意点

* CTEは、それを定義したクエリの実行中にのみ存在します。
* CTEは、定義したクエリ内でのみ参照可能です。
* CTE自体はデータを保存しません。
* 再帰CTEは、無限ループに注意が必要です。

#### 4.6 具体例

**例1：部署ごとの平均給与より高い給与の社員を抽出**

WITH 部署別平均給与 AS (

SELECT

b.部署ID,

b.部署名,

AVG(s.給与) AS 平均給与

FROM

部署 b

JOIN

社員 s ON b.部署ID = s.部署ID

GROUP BY

b.部署ID, b.部署名

)

SELECT

s.氏名,

b.部署名,

s.給与,

d.平均給与

FROM

社員 s

JOIN

部署 b ON s.部署ID = b.部署ID

JOIN

部署別平均給与 d ON b.部署ID = d.部署ID

WHERE

s.給与 > d.平均給与;

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

**例2：社員の階層構造を表示（再帰CTE）**

WITH RECURSIVE 社員階層 AS (

SELECT

社員ID,

氏名,

上司ID,

CAST(氏名 AS VARCHAR(255)) AS 階層パス,

0 AS レベル

FROM

社員

WHERE

上司ID IS NULL

UNION ALL

SELECT

e.社員ID,

e.氏名,

e.上司ID,

CONCAT(h.階層パス, ' -> ', e.氏名),

h.レベル + 1

FROM

社員 e

INNER JOIN

社員階層 h ON e.上司ID = h.社員ID

)

SELECT

\*

FROM

社員階層

ORDER BY

階層パス;

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

### 5. 使い分けの指針：状況に応じて最適なツールを選ぼう！

ウィンドウ関数、ビュー、CTEは、それぞれ異なる特性を持っています。どの機能を使うべきかは、目的や状況によって異なります。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **機能** | **主な用途** | **メリット** | **デメリット** | **典型的な使いどころ** |
| ウィンドウ関数 | 行ごとの詳細情報を保持したまま、関連する行の集合に対して計算 | GROUP BYなしで集計、ランキング、移動平均などを計算できる。柔軟なフレーム指定。 | OVER()句の指定によってはパフォーマンスが低下する可能性がある。 | 各社員の部署内給与ランキング、過去3ヶ月の売上合計、商品の価格帯別グループ分け |
| CTE | 一時的な名前付き結果セットを定義し、クエリ内で再利用 | クエリの可読性、保守性、再利用性が向上。再帰処理が可能。 | CTE自体はデータを保存しない。再帰CTEは無限ループに注意。 | 複雑なクエリを段階的に構築、階層データの処理（組織図、部品表など） |
| ビュー | 複雑なクエリや頻繁に使用するクエリを仮想テーブルとして保存 | クエリの再利用性、簡潔性、セキュリティが向上。データの抽象化、仮想分割。 | 実データを保持しないため、ビュー自体にインデックスを作成できない（マテリアライズドビューを除く）。複雑なビューはパフォーマンスに影響。 | よく使うデータの抽出（営業部社員、高額商品など）、ユーザーへのデータ公開範囲の制限、レポート作成の基盤データ |
| マテリアライズドビュー | クエリの結果を物理的に保存 | 複雑なクエリの結果を高速に取得できる | データの更新は手動、または自動で行う必要がある。 | 集計結果、ランキング |

47.0s

edit

more\_vert

続けて

edit

more\_vert

はい、承知しました。ここからは、ウィンドウ関数、ビュー、CTEを組み合わせて使用する応用例と、SQLの学習をさらに深めるためのヒントを紹介します。

### 6. 応用編：機能を組み合わせてパワーアップ！

ウィンドウ関数、ビュー、CTEは、それぞれ単独でも強力な機能ですが、これらを組み合わせることで、さらに複雑なデータ処理や分析が可能になります。

#### 6.1 ウィンドウ関数 + CTE

ウィンドウ関数の結果をCTEに格納し、その結果に対してさらに別の処理を行うことで、クエリの見通しを良くし、複雑な計算を段階的に実行できます。

**例：部署ごとの給与ランキング上位3名の平均給与を計算する**

WITH 部署内給与ランキング AS (

SELECT

s.氏名,

b.部署名,

s.給与,

RANK() OVER (PARTITION BY b.部署ID ORDER BY s.給与 DESC) AS 部署内順位

FROM

社員 s

JOIN

部署 b ON s.部署ID = b.部署ID

), 上位3名 AS (

SELECT

部署名,

氏名,

給与

FROM

部署内給与ランキング

WHERE

部署内順位 <= 3

)

SELECT

部署名,

AVG(給与) AS 上位3名平均給与

FROM

上位3名

GROUP BY

部署名;

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

**解説:**

1. **部署内給与ランキング CTE:** ウィンドウ関数 RANK() を使って、各社員の部署内での給与順位を計算します。
2. **上位3名 CTE:** 部署内給与ランキング CTE の結果から、部署内順位 が3以下の社員（つまり、各部署の給与ランキング上位3名）を抽出します。
3. **メインクエリ:** 上位3名 CTE の結果に対して GROUP BY と AVG() を使い、各部署の上位3名の平均給与を計算します。

このように、ウィンドウ関数とCTEを組み合わせることで、複雑な処理を段階的に実行し、クエリの可読性と保守性を高めることができます。

#### 6.2 ウィンドウ関数 + ビュー

よく使うウィンドウ関数の計算結果をビューとして保存しておくと、様々なクエリで再利用できます。

**例：各社員の部署内給与順位と部署平均給与をビューとして保存**

CREATE VIEW 社員給与情報 AS

SELECT

s.社員ID,

s.氏名,

b.部署名,

s.給与,

RANK() OVER (PARTITION BY b.部署ID ORDER BY s.給与 DESC) AS 部署内順位,

AVG(s.給与) OVER (PARTITION BY b.部署ID) AS 部署平均給与

FROM

社員 s

JOIN

部署 b ON s.部署ID = b.部署ID;

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

このビューを作成しておけば、以下のように簡単に様々な情報を取得できます。

-- 部署内順位が1位の社員を抽出

SELECT \* FROM 社員給与情報 WHERE 部署内順位 = 1;

-- 部署平均給与より給与が高い社員を抽出

SELECT \* FROM 社員給与情報 WHERE 給与 > 部署平均給与;

-- 部署ごとに、給与が平均より高く、かつ部署内順位が3位以内の社員を抽出

SELECT \*

FROM 社員給与情報

WHERE 給与 > 部署平均給与 AND 部署内順位 <= 3;

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

#### 6.3 CTE + ビュー

CTEを使って複雑なクエリを整理し、その結果をビューとして保存することもできます。

**例：各部署の売上上位3商品を抽出し、その結果をビューとして保存**

-- CTEで各商品の売上とランキングを計算

WITH 商品別売上 AS (

SELECT

p.商品ID,

p.商品名,

c.カテゴリ名,

SUM(oi.数量 \* oi.単価) AS 売上額

FROM

商品 p

JOIN

カテゴリ c ON p.カテゴリID = c.カテゴリID

JOIN

注文詳細 oi ON p.商品ID = oi.商品ID

GROUP BY

p.商品ID, p.商品名, c.カテゴリ名

), 商品別売上ランキング AS (

SELECT

\*,

RANK() OVER (PARTITION BY カテゴリ名 ORDER BY 売上額 DESC) AS 売上ランク

FROM 商品別売上

)

-- CTEの結果を基にビューを作成

CREATE VIEW カテゴリ別売上トップ3商品 AS

SELECT

カテゴリ名,

商品名,

売上額

FROM

商品別売上ランキング

WHERE

売上ランク <= 3;

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

この例では、まずCTEを使って各商品の売上とランキングを計算し、その結果を基にビューを作成しています。

#### 6.4 その他の組み合わせ例

* **再帰CTE + ウィンドウ関数:** 再帰CTEで階層構造を展開し、各階層の社員数などをウィンドウ関数で計算する。
* **CTE + CTE + ...:** 複数のCTEを連鎖的に定義し、非常に複雑なクエリを段階的に構築する。

### 7. 学習を深めるためのヒント

* **実際に手を動かす:** SQLの学習は、実際に手を動かしてクエリを書いてみることが最も重要です。
  + **練習問題:** オンラインのSQL練習問題サイト（SQLZoo, HackerRank, LeetCodeなど）を活用しましょう。
  + **仮想環境:** 自分のPCにデータベース環境（MySQL, PostgreSQL, SQLiteなど）を構築し、自由に試せる環境を作りましょう。
  + **公開データ:** 政府や自治体が公開しているオープンデータなど、実際のデータを使って分析してみましょう。
* **公式ドキュメントを読む:** 使用しているデータベースシステムの公式ドキュメントは、最も正確で詳細な情報源です。
  + **構文:** 各機能の正確な構文を確認しましょう。
  + **オプション:** サポートされているオプションや機能を把握しましょう。
  + **注意点:** パフォーマンスに関する注意点や、データベースシステム固有の挙動を確認しましょう。
* **他の人のコードを読む:**
  + **GitHub:** オープンソースのプロジェクトで、SQLがどのように使われているかを見てみましょう。
  + **Stack Overflow:** SQLに関する質問と回答が豊富にあります。
  + **Qiita, Zenn:** 日本語の技術情報共有サイトでも、SQLに関する記事が多くあります。
* **書籍を読む:**
  + 体系的にSQLを学びたい場合は、書籍を読むのも良い方法です。
  + 初心者向けから上級者向けまで、様々なレベルの書籍があります。

**おすすめの学習リソース:**

* **SQLZoo:** [https://sqlzoo.net/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fsqlzoo.net%2F)
* **HackerRank:** [https://www.hackerrank.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.hackerrank.com%2F)
* **LeetCode:** [https://leetcode.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fleetcode.com%2F)
* **ドットインストール:** [https://dotinstall.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdotinstall.com%2F) (動画学習)
* **Progate:** [https://prog-8.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fprog-8.com%2F) (スライド学習)
* **SQLBolt**: [https://sqlbolt.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fsqlbolt.com%2F)

**書籍:**

* **スッキリわかるSQL入門 第3版 ドリル256問付き! (スッキリわかる入門シリーズ):**
  + 初心者向け。SQLの基本を丁寧に解説しています。
* **達人に学ぶDB設計 徹底指南書 初級者で終わりたくないあなたへ:**
  + データベース設計に関する良書ですが、SQLの応用的な使い方についても解説されています。
* **SQLアンチパターン:**
  + SQLの間違った使い方、避けるべきパターンについて解説しています。

### 8. まとめ

この教材では、SQLのウィンドウ関数、ビュー、CTEについて、基本的な使い方から応用的な使い方まで、幅広く解説しました。これらの機能をマスターし、組み合わせることで、SQLの表現力は飛躍的に向上します。

ぜひ、実際に手を動かしてSQLを書き、データ分析やレポート作成のスキルを磨いてください！

※疑問点

ウィンドウ関数が SELECT 句と ORDER BY 句でのみ使用できるというのは、SQLの構文上の制限を意味します。具体的には、ウィンドウ関数 (OVER() 句を伴う関数) は、以下の場所でのみ使用できます。

**ウィンドウ関数の使用に関する解説**

ウィンドウ関数は、特定の行の集合に対して計算を行う関数です。主な特徴として、以下の点が挙げられます。

* **行のグループ化:**
  + GROUP BY句とは異なり、行をグループ化せずに各行に対して計算を行います。
  + PARTITION BY句を使用することで、行を特定の列の値で分割し、分割されたグループごとに計算を行うことができます。
* **計算の種類:**
  + 平均値(AVG)、合計(SUM)、最大値(MAX)、最小値(MIN)、順位(RANK)など、様々な計算を行うことができます。
* **使用できる場所:**
  + SELECT句: ウィンドウ関数の結果を新しい列として取得するために使用します。
  + ORDER BY句: ウィンドウ関数の結果に基づいて行を並べ替えるために使用します。

-- 社員テーブル (Shain)

-- 社員ID, 氏名, 部署ID, 給与

-- 正しい例 (SELECT句で使用)

SELECT 氏名, 給与, AVG(給与) OVER (PARTITION BY 部署ID) AS 部署平均給与 FROM 社員;

-- 正しい例 (ORDER BY句で使用)

SELECT 氏名, 給与 FROM 社員 ORDER BY RANK() OVER (ORDER BY 給与 DESC);

-- 間違いの例 (WHERE句で使用)

-- SELECT 氏名, 給与 FROM 社員 WHERE AVG(給与) OVER (PARTITION BY 部署ID) > 5000000; -- エラー！

-- 間違いの例 (GROUP BY句で使用)

-- SELECT 部署ID, AVG(給与) FROM 社員 GROUP BY RANK() OVER (ORDER BY 給与 DESC); -- エラー！

-- WHERE句でウィンドウ関数の結果を使いたい場合: サブクエリを使用

SELECT \* FROM (SELECT 氏名, 給与, AVG(給与) OVER (PARTITION BY 部署ID) AS 部署平均給与 FROM 社員) AS サブクエリ WHERE 部署平均給与 > 5000000;

-- WHERE句でウィンドウ関数の結果を使いたい場合: CTEを使用

WITH 部署別平均給与 AS (SELECT 氏名, 給与, AVG(給与) OVER (PARTITION BY 部署ID) AS 部署平均給与 FROM 社員) SELECT \* FROM 部署別平均給与 WHERE 部署平均給与 > 5000000;

**ウィンドウ関数が使用できない場所と理由**

* **FROM句:**
  + FROM句は、データの取得元を指定する場所であり、ウィンドウ関数は行に対する計算を行うため、使用できません。
* **WHERE句:**
  + WHERE句は、行を絞り込むための条件を指定する場所です。
  + ウィンドウ関数は、行を絞り込む前に各行に対して計算を行う必要があるため、WHERE句では使用できません。
  + WHERE句でウィンドウ関数の結果を使いたい場合は、サブクエリまたはCTEを使用する必要があります。
* **GROUP BY句:**
  + GROUP BY句は、行をグループ化するための列を指定する場所です。
  + ウィンドウ関数は、GROUP BY句とは異なり、行をグループ化せずに各行に対して計算を行うため、GROUP BY句では使用できません。
* **HAVING句:**
  + HAVING句は、GROUP BY句でグループ化された結果に対して、さらに絞り込みを行うための条件を指定する場所です。
  + ウィンドウ関数は、グループ化とは関係なく各行に対して計算を行うため、HAVING句では使用できません。

**まとめ**

ウィンドウ関数は、SELECT句とORDER BY句でのみ使用できます。これはSQLの構文上の制限であり、ウィンドウ関数が各行に対して計算を行うという性質に由来します。WHERE句などでウィンドウ関数の結果を使いたい場合は、サブクエリやCTEを使用する必要があります。