==========

9. 集計関数：COUNT、SUM、AVG、MAX、MIN

はじめに

これまでの章では、データベースからレコードを取得して表示する方法を学んできました。しかし、データベースを使う目的の一つは、大量のデータから有用な情報（集計、統計など）を抽出することです。例えば：

- 「学生の総数は何人か？」
- 「成績の平均点は？」
- 「最高点と最低点は？」
- 「全講座の合計受講者数は？」

このような「データの集計」を行うためのSQLの機能が「集計関数」です。この章では、最もよく使われる5つの集計関数（COUNT、SUM、AVG、MAX、MIN）について学びます。

集計関数の基本

集計関数は、複数の行のデータをまとめて一つの値を返す関数です。

> **用語解説**：
>
> - **集計関数**：複数の行（レコード）から計算された単一の値を返す関数です。データの集計や統計に使用されます。

主な集計関数

関数	説明	例
COUNT	レコード数を数える	学生の総数
SUM	値の合計を計算する	全成績の点数合計
AVG	値の平均を計算する	平均点
MAX	最大値を取得する	最高点
MIN	最小値を取得する	最低点

COUNT関数：レコード数をカウントする

COUNT関数は、条件に一致するレコードの数を数えるのに使用します。

基本構文

```
SELECT COUNT(カラム名) FROM テーブル名 [WHERE 条件];
```

または、すべての行を数える場合：

```
SELECT COUNT(*) FROM テーブル名 [WHERE 条件];
```

例1：テーブル内の全レコード数

例えば、学生（students）テーブルの全学生数を数えるには：

```
SELECT COUNT(*) AS 学生総数 FROM students;
```

実行結果：

学生総数
100

例2：条件付きのカウント

WHERE句と組み合わせることで、条件に一致するレコードだけをカウントできます。例えば、「出席（present）」状態の出席レコードの数を数えるには：

```
SELECT COUNT(*) AS 出席数 FROM attendance WHERE status = 'present';
```

実行結果：

出席数
42

例3：NULL以外の値をカウント

COUNT(カラム名)を使うと、そのカラムがNULLでない行だけがカウントされます。これは、COUNT(*)との大きな違いです。例えば、コメント（comment）が入力されている出席レコードの数を数えるには：

```
SELECT COUNT(comment) AS コメントあり FROM attendance;
```

実行結果：

コメントあり
23

例4：DISTINCTを使った重複のないカウント

DISTINCTを使うと、重複を除外してカウントできます。例えば、何種類の評価タイプ（grade_type）があるかを数えるには：

```
SELECT COUNT(DISTINCT grade_type) AS 評価タイプ数 FROM grades;
```

実行結果：

評価タイプ数
3

SUM関数：合計を計算する

SUM関数は、数値カラムの合計を計算するのに使用します。

基本構文

```
SELECT SUM(数値カラム) FROM テーブル名 [WHERE 条件];
```

例1：単純な合計

例えば、全成績レコードの得点（score）の合計を計算するには：

```
SELECT SUM(score) AS 総得点 FROM grades;
```

実行結果：

総得点
3542.5

例2：条件付きの合計

WHERE句と組み合わせることで、条件に一致するレコードだけの合計を計算できます。例えば、「中間テスト」の得点合計を計算するには：

```
SELECT SUM(score) AS 中間テスト合計点
FROM grades
WHERE grade_type = '中間テスト';
```

実行結果：

中間テスト合計点

中間テスト合計点
1728.0

例3：計算式を使った合計

計算式と組み合わせることもできます。例えば、全成績の達成率（score/max_score）の合計を計算するには：

```
SELECT SUM(score/max_score) AS 達成率合計 FROM grades;
```

実行結果：

達成率合計
39.86

AVG関数：平均を計算する

AVG関数は、数値カラムの平均値を計算するのに使用します。

基本構文

```
SELECT AVG(数値カラム) FROM テーブル名 [WHERE 条件];
```

例1：単純な平均

例えば、全成績レコードの得点（score）の平均を計算するには：

```
SELECT AVG(score) AS 平均点 FROM grades;
```

実行結果：

平均点
82.38

例2：条件付きの平均

WHERE句と組み合わせることで、条件に一致するレコードだけの平均を計算できます。例えば、「実技試験」の平均点を計算するには：

```
SELECT AVG(score) AS 実技試験平均点
FROM grades
WHERE grade_type = '実技試験';
```

実行結果：

実技試験平均点
85.6

例3：達成率（%）の計算

計算式と組み合わせることで、例えば平均達成率（%）を計算できます：

```
SELECT AVG(score/max_score * 100) AS 平均達成率 FROM grades;
```

実行結果：

平均達成率
87.24

MAX関数：最大値を取得する

MAX関数は、数値、文字列、日付などのカラムから最大値を取得するのに使用します。

基本構文

```
SELECT MAX(カラム名) FROM テーブル名 [WHERE 条件];
```

例1：数値の最大値

例えば、全成績レコードの中での最高点を取得するには：

```
SELECT MAX(score) AS 最高点 FROM grades;
```

実行結果：

最高点
95.0

例2：文字列の最大値（辞書順で最後）

MAX関数は文字列にも使用でき、この場合は辞書順で「最後」の値を返します。例えば、学生名の辞書順で最後の値（最も「わ行」に近い名前）を取得するには：

```
SELECT MAX(student_name) AS 学生名最終 FROM students;
```

実行結果：

学生名最終
吉川伽羅

例3：日付の最大値（最新の日付）

日付にMAX関数を使うと、最新（最も未来）の日付が取得できます。例えば、最も新しい提出日を取得するには：

```
SELECT MAX(submission_date) AS 最新提出日 FROM grades;
```

実行結果：

最新提出日
2025-05-20

MIN関数：最小値を取得する

MIN関数は、数値、文字列、日付などのカラムから最小値を取得するのに使用します。

基本構文

```
SELECT MIN(カラム名) FROM テーブル名 [WHERE 条件];
```

例1：数値の最小値

例えば、全成績レコードの中での最低点を取得するには：

```
SELECT MIN(score) AS 最低点 FROM grades;
```

実行結果：

最低点
68.0

例2：文字列の最小値（辞書順で最初）

MIN関数は文字列にも使用でき、この場合は辞書順で「最初」の値を返します。例えば、学生名の辞書順で最初の値（最も「あ行」に近い名前）を取得するには：

```
SELECT MIN(student_name) AS 学生名最初 FROM students;
```

実行結果：

学生名最初
相沢吉夫

例3：日付の最小値（最古の日付）

日付にMIN関数を使うと、最も古い日付が取得できます。例えば、最も古い提出日を取得するには：

```
SELECT MIN(submission_date) AS 最古提出日 FROM grades;
```

実行結果：

最古提出日
2025-05-06

複数の集計関数の組み合わせ

複数の集計関数を一つのクエリで使用することもできます。

例：成績の統計情報

例えば、成績（grades）テーブルの統計情報を一度に取得するには：

```
SELECT
    COUNT(*) AS レコード数,
    AVG(score) AS 平均点,
    SUM(score) AS 合計点,
    MAX(score) AS 最高点,
    MIN(score) AS 最低点
FROM grades;
```

実行結果：

レコード数	平均点	合計点	最高点	最低点
43	82.38	3542.5	95.0	68.0

集計関数とGROUP BY（次章の内容）

集計関数をより強力に使うためには、「GROUP BY」句と組み合わせます。これにより、データをグループ化して各グループごとに集計できます。GROUP BYについては次章で詳しく学びます。

例えば、講座（course_id）ごとの平均点を計算するには：

```
SELECT course_id, AVG(score) AS 平均点
FROM grades
GROUP BY course_id;
```

実行結果：

course_id	平均点
1	86.2
2	83.8
3	80.5
...	...

この例の詳細な説明は次章で行います。

練習問題

問題9-1

students（学生）テーブルから、学生の総数を取得するSQLを書いてください。

問題9-2

grades（成績）テーブルから、全成績の平均点（score）を取得するSQLを書いてください。

問題9-3

attendance（出席）テーブルから、出席状況（status）が「absent」のレコード数を取得するSQLを書いてください。

問題9-4

grades（成績）テーブルから、評価タイプ（grade_type）が「中間テスト」の成績の最高点と最低点を取得するSQLを書いてください。

問題9-5

course_schedule（授業カレンダー）テーブルから、最新（schedule_dateが最大）の授業スケジュールの日付を取得するSQLを書いてください。

問題9-6

grades（成績）テーブルから、成績の総数、平均点、合計点、最高点、最低点を一度に取得するSQLを書いてください。

解答

解答9-1

```
SELECT COUNT(*) AS 学生総数 FROM students;
```

解答9-2

```
SELECT AVG(score) AS 全成績平均点 FROM grades;
```

解答9-3

```
SELECT COUNT(*) AS 欠席数 FROM attendance WHERE status = 'absent';
```

解答9-4

```
SELECT
    MAX(score) AS 中間テスト最高点,
    MIN(score) AS 中間テスト最低点
FROM grades
WHERE grade_type = '中間テスト';
```

解答9-5

```
SELECT MAX(schedule_date) AS 最新授業日 FROM course_schedule;
```

解答9-6

```
SELECT
    COUNT(*) AS 成績総数,
    AVG(score) AS 平均点,
    SUM(score) AS 合計点,
    MAX(score) AS 最高点,
    MIN(score) AS 最低点
FROM grades;
```

まとめ

この章では、データの集計と分析に使用される主要な集計関数について学びました：

1. **COUNT**：レコード数をカウントする関数
2. **SUM**：数値の合計を計算する関数
3. **AVG**：数値の平均を計算する関数
4. **MAX**：最大値（数値、文字列、日付など）を取得する関数
5. **MIN**：最小値（数値、文字列、日付など）を取得する関数

これらの集計関数を使うことで、大量のデータから意味のある統計情報を簡単に抽出できます。ビジネスにおける意思決定や、データ分析において非常に重要な機能です。

次の章では、データをグループ化して集計を行うための「GROUP BY：データのグループ化」について学びます。

---

10. GROUP BY：データのグループ化

はじめに

前章では、集計関数（COUNT、SUM、AVG、MAX、MIN）を使って全体的な集計や統計を計算する方法を学びました。しかし実際のデータ分析では、全体の集計だけでなく、特定のカテゴリや条件ごとに集計したい場合が多くあります。例えば：

- 「講座ごとの平均点は？」
- 「教師ごとの担当講座数は？」
- 「日付ごとの授業数は？」
- 「出席状況（出席/遅刻/欠席）の割合は？」

このような「グループごとの集計」を行うためのSQLコマンドが「GROUP BY」です。この章では、データをグループ化して集計する方法について学びます。

GROUP BYの基本

GROUP BY句は、指定したカラムの値が同じレコードをグループ化し、それぞれのグループに対して集計関数を適用するために使います。

> **用語解説**：
>
> - **GROUP BY**：「～でグループ化する」という意味のSQLコマンドで、同じ値を持つレコードをまとめてグループにします。
> - **グループ化**：データを特定の条件で分類し、それぞれの分類ごとに集計を行うこと。

基本構文

```
SELECT カラム名, 集計関数
FROM テーブル名
```

```
[WHERE 条件]
GROUP BY グループ化するカラム名;
```

重要なのは、SELECT句に含めるカラムは、GROUP BY句で指定したカラムか、集計関数（COUNT、SUM、AVG、MAX、MINなど）のいずれかでなければならないということです。

例1：単純なグループ化

例えば、成績（grades）テーブルから、評価タイプ（grade_type）ごとの成績レコード数を集計するには：

```
SELECT grade_type, COUNT(*) AS レコード数
FROM grades
GROUP BY grade_type;
```

実行結果：

grade_type	レコード数
中間テスト	12
レポート1	22
実技試験	9

例2：グループごとの平均

グループごとの平均を計算することも一般的です。例えば、各評価タイプごとの平均点を計算するには：

```
SELECT grade_type, AVG(score) AS 平均点
FROM grades
GROUP BY grade_type;
```

実行結果：

grade_type	平均点
中間テスト	86.25
レポート1	44.77
実技試験	85.61

複数のカラムによるグループ化

複数のカラムを指定してグループ化することもできます。この場合、指定したすべてのカラムの値の組み合わせがグループの単位になります。

例3：複数カラムでのグループ化

例えば、講座（course_id）と評価タイプ（grade_type）の組み合わせごとに成績の平均を計算するには：

```
SELECT course_id, grade_type, AVG(score) AS 平均点
FROM grades
GROUP BY course_id, grade_type;
```

実行結果：

course_id	grade_type	平均点
1	中間テスト	87.33
1	レポート1	45.39
2	実技試験	86.83
…	…	…

グループ化の順序

GROUP BY句でカラムを指定する順序は、結果には影響しません。しかし、可読性のためにSELECT句と同じ順序で指定することが一般的です。

WHERE句とGROUP BYの組み合わせ

WHERE句は、グループ化する前に行単位でレコードを絞り込むのに使います。

> **用語解説**：
>
> - **絞り込み順序**：WHERE句はGROUP BY句の前に評価され、条件に合うレコードだけがグループ化の対象になります。

例4：WHEREとGROUP BYの組み合わせ

例えば、得点（score）が80以上の成績だけを対象に、評価タイプごとの平均を計算するには：

```
SELECT grade_type, AVG(score) AS 平均点
FROM grades
WHERE score >= 80
GROUP BY grade_type;
```

実行結果：

grade_type	平均点
中間テスト	88.92
実技試験	87.25

この例では、score >= 80の条件に一致するレコードだけがグループ化され、集計されています。レポート1はすべての点数が80未満なので、結果には表示されていません。

GROUP BYとORDER BYの組み合わせ

GROUP BY句とORDER BY句を組み合わせることで、グループ化した結果を任意の順序で並べることができます。

例5：GROUP BYとORDER BYの組み合わせ

例えば、講座（course_id）ごとの平均点を計算し、平均点の高い順に並べるには：

```
SELECT course_id, AVG(score) AS 平均点
FROM grades
GROUP BY course_id
ORDER BY 平均点 DESC;
```

実行結果：

course_id	平均点
1	86.21
2	83.79
5	82.33
...	...

集計関数の複数使用

一つのクエリで複数の集計関数を使用することもできます。

例6：複数の集計関数の使用

例えば、評価タイプごとの成績数、平均点、最高点、最低点を一度に取得するには：

```
SELECT grade_type,
       COUNT(*) AS 成績数,
       AVG(score) AS 平均点,
       MAX(score) AS 最高点,
       MIN(score) AS 最低点
FROM grades
GROUP BY grade_type;
```

実行結果：

grade_type	成績数	平均点	最高点	最低点

grade_type	成績数	平均点	最高点	最低点
中間テスト	12	86.25	95.0	78.5
レポート1	22	44.77	49.0	37.0
実技試験	9	85.61	88.0	79.5

計算式を含むグループ化

GROUP BY句で集計した結果に対して、さらに計算を加えることができます。

例7：計算式を含むグループ化

例えば、評価タイプごとに平均達成率（score/max_score）を計算するには：

```
SELECT grade_type,
       AVG(score/max_score * 100) AS 平均達成率
FROM grades
GROUP BY grade_type;
```

実行結果：

grade_type	平均達成率
中間テスト	86.25
レポート1	89.54
実技試験	85.61

GROUP BYとNULLの扱い

GROUP BY句でグループ化する際、NULL値も一つのグループとして扱われます。

例8：NULLを含むグループ化

例えば、出席（attendance）テーブルから、コメント（comment）の有無でグループ化し、それぞれの件数を数えるには：

```
SELECT
    CASE
        WHEN comment IS NULL THEN 'コメントなし'
        ELSE 'コメントあり'
    END AS コメント状態,
    COUNT(*) AS 件数
FROM attendance
GROUP BY
    CASE
        WHEN comment IS NULL THEN 'コメントなし'
```

```
        ELSE 'コメントあり'
    END;
```

実行結果：

コメント状態	件数
コメントなし	20
コメントあり	23

HAVINGを使ったグループの絞り込み（次章の内容）

グループ化した後にさらに条件でグループを絞り込むには、「HAVING」句を使います。これについては次章で詳しく学びます。

例えば、5人以上の学生が受講している講座だけを取得するには：

```
SELECT course_id, COUNT(student_id) AS 学生数
FROM student_courses
GROUP BY course_id
HAVING COUNT(student_id) >= 5;
```

この例の詳細な説明は次章で行います。

練習問題

問題10-1

courses（講座）テーブルから、教師ID（teacher_id）ごとに担当している講座の数を取得するSQLを書いてください。

問題10-2

attendance（出席）テーブルから、出席状況（status）ごとのレコード数を取得するSQLを書いてください。

問題10-3

grades（成績）テーブルから、学生ID（student_id）ごとの平均点を計算し、平均点の高い順に並べるSQLを書いてください。

問題10-4

course_schedule（授業カレンダー）テーブルから、教室ID（classroom_id）ごとに予定されている授業の数を取得するSQLを書いてください。

問題10-5

grades（成績）テーブルから、講座ID（course_id）と評価タイプ（grade_type）の組み合わせごとの平均点を計算し、講座ID順、さらに評価タイプ順で並べるSQLを書いてください。

問題10-6

student_courses（受講）テーブルから、講座ID（course_id）ごとの受講者数を取得し、受講者数の多い順に並べるSQLを書いてください。

解答

解答10-1

```
SELECT teacher_id, COUNT(course_id) AS 担当講座数
FROM courses
GROUP BY teacher_id;
```

解答10-2

```
SELECT status, COUNT(*) AS レコード数
FROM attendance
GROUP BY status;
```

解答10-3

```
SELECT student_id, AVG(score) AS 平均点
FROM grades
GROUP BY student_id
ORDER BY 平均点 DESC;
```

解答10-4

```
SELECT classroom_id, COUNT(*) AS 授業数
FROM course_schedule
GROUP BY classroom_id;
```

解答10-5

```
SELECT course_id, grade_type, AVG(score) AS 平均点
FROM grades
GROUP BY course_id, grade_type
ORDER BY course_id, grade_type;
```

解答10-6

```
SELECT course_id, COUNT(student_id) AS 受講者数
FROM student_courses
GROUP BY course_id
ORDER BY 受講者数 DESC;
```

まとめ

この章では、データをグループ化して集計するためのGROUP BY句について学びました：

1. **GROUP BYの基本**：同じ値を持つレコードをグループ化する方法
2. **集計関数との組み合わせ**：グループごとの集計を行う方法
3. **複数カラムによるグループ化**：複数の条件でのグループ化
4. **WHERE句との組み合わせ**：グループ化前のレコード絞り込み
5. **ORDER BYとの組み合わせ**：グループ化結果の並び替え
6. **複数の集計関数の使用**：一度に複数の統計値を取得する方法
7. **計算式を含むグループ化**：より複雑な集計の実現
8. **NULLの扱い**：グループ化におけるNULL値の取り扱い

GROUP BY句を使うことで、データのさまざまな側面から分析が可能になり、意思決定のための有用な情報を抽出できるようになります。

次の章では、グループ化した結果をさらに条件で絞り込むための「HAVING：グループ化後の絞り込み」について学びます。

---

11. HAVING：グループ化後の絞り込み

はじめに

前章では、GROUP BY句を使ってデータをグループ化し、各グループごとに集計を行う方法を学びました。しかし、すべてのグループの集計結果を表示するのではなく、特定の条件を満たすグループだけを表示したい場合があります。例えば：

- 「平均点が80点以上の講座だけを表示したい」
- 「5人以上の学生が登録している講座だけを取得したい」
- 「出席率が90%を超える学生だけをリストアップしたい」

このような「グループ化した結果をさらに絞り込む」ためのSQLコマンドが「HAVING」句です。この章では、グループ化した結果に条件を適用する方法について学びます。

HAVINGの基本

HAVING句は、GROUP BY句でグループ化した後の結果に対して条件を適用し、条件を満たすグループだけを取得するために使います。

> **用語解説：**

> - **HAVING**：「～を持っている」という意味のSQLコマンドで、グループ化した結果に対して条件を適用します。

基本構文

```
SELECT カラム名, 集計関数
FROM テーブル名
[WHERE 行レベルの条件]
GROUP BY グループ化するカラム名
HAVING グループレベルの条件;
```

例1：基本的なHAVINGの使用

例えば、受講者数が5人以上の講座だけを取得するには：

```
SELECT course_id, COUNT(student_id) AS 受講者数
FROM student_courses
GROUP BY course_id
HAVING COUNT(student_id) >= 5;
```

実行結果：

course_id	受講者数
1	12
2	8
4	7
5	7
…	…

この例では、まず講座ID（course_id）ごとに学生ID（student_id）の数を数え、その後でHAVING句を使って受講者数が5人以上のグループだけを抽出しています。

WHEREとHAVINGの違い

WHERE句とHAVING句は似ていますが、適用されるタイミングと対象が異なります：

- **WHERE**：グループ化される前の個々の行（レコード）に対して条件を適用します。
- **HAVING**：グループ化された後のグループに対して条件を適用します。

> **用語解説**：
>
> - **行レベルのフィルタリング**：WHEREによる個々のレコードの絞り込み
> - **グループレベルのフィルタリング**：HAVINGによるグループの絞り込み

例2：WHEREとHAVINGの違い

以下の例で違いを確認しましょう：

```
-- WHERE（行レベル）：まず80点以上の成績だけを選び、それからグループ化
SELECT course_id, AVG(score) AS 平均点
FROM grades
WHERE score >= 80
GROUP BY course_id;

-- HAVING（グループレベル）：まずグループ化して平均点を計算し、それから平均点が80以上のグループを選ぶ
SELECT course_id, AVG(score) AS 平均点
FROM grades
GROUP BY course_id
HAVING AVG(score) >= 80;
```

1つ目のクエリは、「80点以上の成績だけ」を対象に講座ごとの平均点を計算します。 2つ目のクエリは、「講座の平均点が80点以上」の講座だけを取得します。

WHERE句の結果：

course_id	平均点
1	88.92
2	87.25
…	…

HAVING句の結果：

course_id	平均点
1	86.21
2	83.79
5	82.33
…	…

HAVINGで使用できる条件

HAVING句では、集計関数（COUNT、SUM、AVG、MAX、MINなど）を使った条件や、GROUP BY句で指定したカラムに対する条件を指定できます。

例3：集計関数を使った条件

例えば、平均点が85点以上の講座を取得するには：

```
SELECT course_id, AVG(score) AS 平均点
FROM grades
GROUP BY course_id
HAVING AVG(score) >= 85;
```

実行結果：

course_id	平均点
1	86.21
…	…

例4：複数条件の指定

HAVING句も、WHERE句と同様に複数の条件を組み合わせることができます：

```
SELECT grade_type, COUNT(*) AS 件数, AVG(score) AS 平均点
FROM grades
GROUP BY grade_type
HAVING COUNT(*) > 10 AND AVG(score) > 80;
```

実行結果：

grade_type	件数	平均点
中間テスト	12	86.25
…	…	…

WHEREとHAVINGの組み合わせ

実際のクエリでは、WHERE句とHAVING句を組み合わせることが多いです。WHERE句でグループ化前の行を絞り込み、HAVING句でグループ化後の結果をさらに絞り込みます。

例5：WHEREとHAVINGの組み合わせ

例えば、「中間テストのみを対象として、平均点が85点以上の講座」を取得するには：

```
SELECT course_id, AVG(score) AS 平均点
FROM grades
WHERE grade_type = '中間テスト'
GROUP BY course_id
HAVING AVG(score) >= 85;
```

実行結果：

course_id	平均点
1	87.33
...	...

HAVINGとORDER BYの組み合わせ

HAVING句で絞り込んだ結果を並べ替えるには、ORDER BY句を追加します。

例6：HAVINGとORDER BYの組み合わせ

例えば、受講者数が5人以上の講座を、受講者数の多い順に並べて取得するには：

```
SELECT course_id, COUNT(student_id) AS 受講者数
FROM student_courses
GROUP BY course_id
HAVING COUNT(student_id) >= 5
ORDER BY 受講者数 DESC;
```

実行結果：

course_id	受講者数
1	12
20	11
9	10
...	...

実践的なHAVINGの使用例

例7：「平均達成率が90%を超える評価タイプ」の抽出

各評価タイプごとの平均達成率（score/max_score）を計算し、90%を超えるものだけを取得します：

```
SELECT grade_type,
       AVG(score/max_score * 100) AS 平均達成率
FROM grades
GROUP BY grade_type
HAVING AVG(score/max_score * 100) > 90;
```

実行結果：

grade_type	平均達成率
レポート1	93.54

grade_type	平均達成率
...	...

例8：「特定の講座で成績データが存在する学生」の抽出

例えば、講座ID「1」の成績が2件以上ある学生を取得します：

```
SELECT student_id, COUNT(*) AS 成績件数
FROM grades
WHERE course_id = '1'
GROUP BY student_id
HAVING COUNT(*) >= 2;
```

実行結果：

student_id	成績件数
301	2
302	2
...	...

HAVINGでの注意点

1. **集計関数の使用**: HAVINGで使用する条件には、通常、集計関数（COUNT、SUM、AVG、MAX、MINなど）が含まれます。単純なカラム比較だけならWHERE句を使用すべきです。
2. **パフォーマンスの考慮**: WHERE句はグループ化前に適用されるため、処理対象のレコード数を減らすことができます。可能な限り、グループ化前の条件はWHERE句で指定し、グループ化後の条件だけをHAVING句で指定するようにすると、効率的なクエリになります。
3. **グループレベルの条件**: HAVING句は、GROUP BY句で指定したカラムや集計関数の結果に対する条件でなければならないことを覚えておきましょう。

練習問題

問題11-1

student_courses（受講）テーブルから、受講者数が8人以上の講座ID（course_id）を取得するSQLを書いてください。

問題11-2

grades（成績）テーブルから、平均点（score）が85点以上の評価タイプ（grade_type）を取得するSQLを書いてください。

問題11-3

courses（講座）テーブルから、各教師（teacher_id）が担当する講座数を計算し、担当講座が3つ以上の教師だけを取得するSQLを書いてください。

問題11-4

grades（成績）テーブルから、講座ID（course_id）ごとの最高点（score）を計算し、最高点が90点以上の講座を、最高点の高い順に並べて取得するSQLを書いてください。

問題11-5

attendance（出席）テーブルから、欠席（status = 'absent'）の回数が2回以上ある学生ID（student_id）を取得するSQLを書いてください。

問題11-6

grades（成績）テーブルを使って、中間テスト（grade_type = '中間テスト'）のデータのみを対象に、平均点が85点以上で、かつ最低点が75点以上の講座ID（course_id）を取得するSQLを書いてください。

解答

解答11-1

```
SELECT course_id, COUNT(student_id) AS 受講者数
FROM student_courses
GROUP BY course_id
HAVING COUNT(student_id) >= 8;
```

解答11-2

```
SELECT grade_type, AVG(score) AS 平均点
FROM grades
GROUP BY grade_type
HAVING AVG(score) >= 85;
```

解答11-3

```
SELECT teacher_id, COUNT(course_id) AS 担当講座数
FROM courses
GROUP BY teacher_id
HAVING COUNT(course_id) >= 3;
```

解答11-4

```

```

```sql
SELECT course_id, MAX(score) AS 最高点
FROM grades
GROUP BY course_id
HAVING MAX(score) >= 90
ORDER BY 最高点 DESC;
```

解答11-5

```sql
SELECT student_id, COUNT(*) AS 欠席回数
FROM attendance
WHERE status = 'absent'
GROUP BY student_id
HAVING COUNT(*) >= 2;
```

解答11-6

```sql
SELECT course_id, AVG(score) AS 平均点, MIN(score) AS 最低点
FROM grades
WHERE grade_type = '中間テスト'
GROUP BY course_id
HAVING AVG(score) >= 85 AND MIN(score) >= 75;
```

まとめ

この章では、グループ化した結果に条件を適用するためのHAVING句について学びました：

1. **HAVINGの基本**：グループ化した結果に条件を適用する方法
2. **WHEREとHAVINGの違い**：
   - WHERE：グループ化前の行レベルのフィルタリング
   - HAVING：グループ化後のグループレベルのフィルタリング
3. **HAVING句での条件**：集計関数を使った条件設定
4. **複合条件**：ANDやORを使った複数条件の組み合わせ
5. **WHERE、HAVING、ORDER BYの組み合わせ**：複雑なデータ抽出の手法
6. **実践的な使用例**：実際の業務で使われるようなクエリパターン

HAVING句はデータ分析において非常に重要です。グループ化されたデータに対して「どのグループが重要か」「どのグループに注目すべきか」という条件を設定することで、より意味のある情報を抽出することができます。

次の章では、重複データを除外するための「DISTINCT：重複の除外」について学びます。

---

12. DISTINCT：重複の除外

はじめに

データベースから情報を取得する際、同じ値が複数回出現することがよくあります。例えば、「どの講座がどの教室で行われているか」を調べると、同じ教室が複数回リストされるでしょう。しかし、時には重複を除いた一意の値のリストだけが必要な場合があります。例えば：

- 「学校にはどんな教室があるか」（重複なく知りたい）
- 「どの教師が授業を担当しているか」（重複なく知りたい）
- 「どのような評価タイプがあるか」（重複なく知りたい）

このような「重複を除外」するためのSQLキーワードが「DISTINCT」です。この章では、クエリ結果から重複するデータを除外する方法について学びます。

DISTINCTの基本

DISTINCT句は、SELECT文の結果から重複する行を除外するために使います。

> **用語解説**：
>
> - **DISTINCT**：「異なる」「区別される」という意味のSQLキーワードで、クエリ結果から重複する行を除外します。
> - **重複除外**：同じ値を持つレコードを1つだけにして、残りを除外すること。

基本構文

```
SELECT DISTINCT カラム名 FROM テーブル名 [WHERE 条件];
```

DISTINCTは、SELECT文の直後に配置され、すべての指定されたカラムの組み合わせに対して働きます。

例1：単一カラムでの重複除外

例えば、成績テーブル（grades）から、どのような評価タイプ（grade_type）があるかを重複なしで取得するには：

```
SELECT DISTINCT grade_type FROM grades;
```

実行結果：

grade_type
中間テスト
レポート1
実技試験

通常のSELECT文では、テーブル内の各行の評価タイプが表示されますが、DISTINCTを使うと、一意の評価タイプだけが表示されます。

例2：出席状況の種類の取得

出席（attendance）テーブルから、どのような出席状況（status）があるかを重複なしで取得するには：

```
SELECT DISTINCT status FROM attendance;
```

実行結果：

status
present
late
absent

複数カラムでのDISTINCT

DISTINCTは複数のカラムにも適用できます。この場合、指定したすべてのカラムの値の組み合わせが一意であるレコードだけが返されます。

基本構文

```
SELECT DISTINCT カラム名1, カラム名2, ... FROM テーブル名 [WHERE 条件];
```

例3：複数カラムでの重複除外

例えば、授業スケジュール（course_schedule）テーブルから、どの講座（course_id）がどの時限（period_id）に開講されているかを重複なしで取得するには：

```
SELECT DISTINCT course_id, period_id FROM course_schedule;
```

実行結果：

course_id	period_id
1	1
2	3
3	4
4	2
...	...

この結果は、course_idとperiod_idの組み合わせが一意のレコードのみを表示します。同じ講座が異なる時限に開講される場合や、異なる講座が同じ時限に開講される場合は、別々のレコードとして表示されます。

COUNT関数との組み合わせ

DISTINCTはCOUNT関数と組み合わせることで、一意の値の数を数えることができます。

例4：一意の値の数を数える

例えば、学生（students）テーブルに何人の学生が登録されているかを数えるには：

```
SELECT COUNT(*) AS 学生総数 FROM students;
```

実行結果：

学生総数
100

一方、受講（student_courses）テーブルに登録されている一意の学生数を数えるには：

```
SELECT COUNT(DISTINCT student_id) AS 受講学生数 FROM student_courses;
```

実行結果：

受講学生数
85

この2つの結果の差は、まだ1つも講座を受講していない学生が15人いることを意味します。

DISTINCTとNULLの扱い

DISTINCTを使う場合、NULL値も1つの値として扱われます。複数のNULL値は1つのNULL値として集約されます。

例5：NULLを含むデータでのDISTINCT

例えば、出席（attendance）テーブルから、コメント（comment）の一意の値を取得するとします：

```
SELECT DISTINCT comment FROM attendance;
```

実行結果：

comment

comment
NULL
15分遅刻
5分遅刻
事前連絡あり
体調不良

...

この結果には、NULL値も1つの行として含まれています。

DISTINCTとORDER BYの組み合わせ

DISTINCTはORDER BY句と組み合わせて使うことができます。これにより、重複を除外した後で結果を並べ替えることができます。

例6：DISTINCTとORDER BYの組み合わせ

例えば、講座（courses）テーブルから、どの教師（teacher_id）が講座を担当しているかを重複なしで取得し、教師IDの順に並べるには：

```
SELECT DISTINCT teacher_id FROM courses ORDER BY teacher_id;
```

実行結果：

teacher_id
101
102
103
104

...

DISTINCTとWHEREの組み合わせ

DISTINCTはWHERE句と組み合わせて使うこともできます。これにより、特定の条件に合うレコードだけを対象に重複を除外できます。

例7：DISTINCTとWHEREの組み合わせ

例えば、2025年5月に授業がある教室（classroom_id）の一覧を重複なしで取得するには：

```
SELECT DISTINCT classroom_id
FROM course_schedule
WHERE schedule_date BETWEEN '2025-05-01' AND '2025-05-31';
```

実行結果：

classroom_id
101A
102B
201C
202D
...

DISTINCTとGROUP BYの違い

DISTINCTとGROUP BYはどちらも「重複を除外する」という点では似ていますが、目的と使い方に違いがあります。

- **DISTINCT**：単純に重複する行を除外します。
- **GROUP BY**：グループごとに集計を行うために使います。集計関数（COUNT、SUM、AVG、MAX、MINなど）と一緒に使うことが一般的です。

例8：DISTINCTとGROUP BYの比較

例えば、講座（courses）テーブルから、どの教師（teacher_id）が講座を担当しているかを調べる場合：

DISTINCTを使う方法：

```
SELECT DISTINCT teacher_id FROM courses;
```

GROUP BYを使う方法：

```
SELECT teacher_id FROM courses GROUP BY teacher_id;
```

両方とも同じ結果を返しますが、目的が異なります。GROUP BYは集計を行うためのもので、例えば各教師が担当する講座数を数えたい場合は：

```
SELECT teacher_id, COUNT(*) AS 担当講座数
FROM courses
GROUP BY teacher_id;
```

このように、GROUP BYと集計関数を組み合わせて使います。

パフォーマンスへの影響と注意点

1. **処理コスト**：DISTINCTはすべての行を調査して重複を除外するため、大量のデータがある場合は処理コストが高くなる可能性があります。
2. **代替手段の検討**：場合によっては、DISTINCTの代わりにGROUP BYを使ったり、EXISTS/NOT EXISTSを使ったりと、より効率的な方法があることがあります。
3. **部分一致には使えない**：DISTINCTは完全に一致するレコードのみを対象とします。部分的な一致や似ているレコードの除外には使えません。

練習問題

問題12-1

course_schedule（授業カレンダー）テーブルから、授業が行われる日付（schedule_date）のリストを重複なしで取得するSQLを書いてください。

問題12-2

grades（成績）テーブルから、何種類の評価タイプ（grade_type）があるかを数えるSQLを書いてください。

問題12-3

student_courses（受講）テーブルから、講座を受講している学生ID（student_id）を重複なしで取得し、IDの昇順に並べるSQLを書いてください。

問題12-4

course_schedule（授業カレンダー）テーブルから、どの教室（classroom_id）でどの時限（period_id）に授業が行われているかの組み合わせを重複なしで取得するSQLを書いてください。

問題12-5

attendance（出席）テーブルから、コメント（comment）が入力されている（NULLでない）一意のコメント内容を取得するSQLを書いてください。

問題12-6

grades（成績）テーブルから、どの学生（student_id）がどの講座（course_id）を受講したかの組み合わせを重複なしで取得し、学生ID、講座IDの順に並べるSQLを書いてください。

解答

解答12-1

```
SELECT DISTINCT schedule_date FROM course_schedule;
```

解答12-2

```
SELECT COUNT(DISTINCT grade_type) AS 評価タイプ数 FROM grades;
```

解答12-3

```
SELECT DISTINCT student_id FROM student_courses ORDER BY student_id;
```

解答12-4

```
SELECT DISTINCT classroom_id, period_id FROM course_schedule;
```

解答12-5

```
SELECT DISTINCT comment FROM attendance WHERE comment IS NOT NULL;
```

解答12-6

```
SELECT DISTINCT student_id, course_id
FROM grades
ORDER BY student_id, course_id;
```

まとめ

この章では、クエリ結果から重複を除外するためのDISTINCTキーワードについて学びました：

1. **DISTINCTの基本**：クエリ結果から重複する行を除外する方法
2. **単一カラムでの使用**：1つのカラムの重複を除外する方法
3. **複数カラムでの使用**：複数カラムの組み合わせの重複を除外する方法
4. **COUNT関数との組み合わせ**：一意の値の数を数える方法
5. **NULLの扱い**：DISTINCT使用時のNULL値の取り扱い
6. **ORDER BYとの組み合わせ**：重複除外後の並べ替え
7. **WHEREとの組み合わせ**：条件付きでの重複除外
8. **GROUP BYとの違い**：似た機能を持つGROUP BYとの使い分け
9. **パフォーマンスへの影響**：DISTINCTを使用する際の注意点

DISTINCTは、データベースから一意の値のリストを取得するための便利な機能です。特に、テーブル内の特定のカラムにどのような値が存在するかを調べたい場合や、重複のないマスターリストを作成したい場合に役立ちます。

次の章では、複数のテーブルを結合して情報を取得するための「JOIN基本：テーブル結合の概念」について学びます。
