# SQL学習テキスト 完全版

このテキストは、SQLの基礎から応用まで体系的に学習できるように構成されています。 学校データベースを使った実践的な例題と練習問題を通じて、実務で使えるSQLスキルを身につけることができます。

---

## 目次



---

# 13. JOIN基本：テーブル結合の概念

## はじめに

これまでの章では、単一のテーブルからデータを取得する方法を学んできました。しかし実際のデータベースでは、情報は複数のテーブルに分散して保存されています。

例えば、学校データベースでは：

- 学生の基本情報は「students」テーブルに
- 講座の情報は「courses」テーブルに
- 成績データは「grades」テーブルに

このように別々のテーブルに保存されたデータを組み合わせて取得するための機能が「JOIN（結合）」です。この章では、複数のテーブルを結合して必要な情報を取得する基本的な方法を学びます。

## JOINとは

JOIN（結合）とは、2つ以上のテーブルを関連付けて、それらのテーブルから情報を組み合わせて取得するためのSQL機能です。

> **用語解説**：
>
> - **JOIN**：「結合する」という意味のSQLコマンドで、複数のテーブルを関連付けてデータを取得するために使います。
> - **結合キー**：テーブル間の関連付けに使用されるカラムのことで、通常は主キーと外部キーの関係にあります。

## テーブル結合の必要性

なぜテーブル結合が必要なのでしょうか？いくつかの理由があります：

1. **データの正規化**：データベース設計では、情報の重複を避けるためにデータを複数のテーブルに分割します（これを「正規化」と呼びます）。
2. **データの一貫性**：例えば、教師の名前を1か所（teachersテーブル）だけで管理することで、名前変更時の更新が容易になります。
3. **効率的なデータ管理**：関連するデータをグループ化して管理できます。

例えば、「どの学生がどの講座でどのような成績を取ったか」という情報を取得するには、students、courses、gradesの3つのテーブルを結合する必要があります。

## 基本的なJOINの種類

SQLでは主に4種類のJOINが使われます：

1. **JOIN**：両方のテーブルで一致するレコードだけを取得(INNER JOIN)
2. **LEFT JOIN**：左テーブルのすべてのレコードと、右テーブルの一致するレコード
3. **RIGHT JOIN**：右テーブルのすべてのレコードと、左テーブルの一致するレコード
4. **FULL JOIN**：両方のテーブルのすべてのレコード（MySQLでは直接サポートされていません）

この章では主にJOINについて学び、次章で他の種類のJOINを学びます。

## JOIN（内部結合）

JOINは、最も基本的な結合方法で、両方のテーブルで一致するレコードのみを取得します。

> **用語解説**：
>
> - **JOIN**：「内部結合」とも呼ばれ、結合条件に一致するレコードのみを返します。条件に一致しないレコードは結果に含まれません。

### 基本構文

```
SELECT カラム名
FROM テーブル1
JOIN テーブル2 ON テーブル1.結合カラム = テーブル2.結合カラム;
```

「ON」の後には結合条件を指定します。これは通常、一方のテーブルの主キーと他方のテーブルの外部キーを一致させる条件です。

> **用語解説**：
>
> - **ON**：「～の条件で」という意味で、JOINの条件を指定するためのキーワードです。

## JOINの実践例

### 例1：講座とその担当教師の情報を取得

講座（courses）テーブルと教師（teachers）テーブルを結合して、各講座の名前と担当教師の名前を取得してみましょう：

```
SELECT c.course_id, c.course_name, t.teacher_name
FROM courses c
JOIN teachers t ON c.teacher_id = t.teacher_id;
```

この例では：

- `c`と`t`はそれぞれcourses、teachersテーブルの「テーブル別名」（短縮名）です。
- `ON c.teacher_id = t.teacher_id`が結合条件です。
- 講座テーブルの`teacher_id`（外部キー）と教師テーブルの`teacher_id`（主キー）が一致するレコードが結合されます。

実行結果：

| course_id | course_name | teacher_name |
|---|---|---|
| 1 | ITのための基礎知識 | 寺内鞍 |
| 2 | UNIX入門 | 田尻朋美 |
| 3 | Cプログラミング演習 | 寺内鞍 |
| 4 | Webアプリケーション開発 | 藤本理恵 |
| … | … | … |

### 例2：学生と受講講座の情報を取得

学生（students）テーブルと受講（student_courses）テーブルを結合して、特定の講座を受講している学生の一覧を取得してみましょう：

```
SELECT s.student_id, s.student_name, sc.course_id
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
WHERE sc.course_id = '1';
```

この例では、学生テーブルと受講テーブルを学生IDで結合し、講座ID = 1の学生だけを取得しています。

実行結果：

| student_id | student_name | course_id |
|---|---|---|
| 301 | 黒沢春馬 | 1 |
| 302 | 新垣愛留 | 1 |
| 303 | 柴崎春花 | 1 |
| 306 | 河田咲奈 | 1 |
| … | … | … |

### 例3：3つのテーブルの結合

より複雑な例として、3つのテーブルを結合してみましょう。学生、講座、成績の情報を組み合わせて表示します：

```
SELECT s.student_name, c.course_name, g.score
FROM students s
JOIN grades g ON s.student_id = g.student_id
JOIN courses c ON g.course_id = c.course_id
WHERE g.grade_type = '中間テスト'
ORDER BY g.score DESC;
```

この例では：

1. まず学生テーブルと成績テーブルを結合
2. その結果と講座テーブルを結合
3. 中間テストの成績だけを抽出
4. 点数の高い順にソート

実行結果：

| student_name | course_name | score |
|---|---|---|
| 鈴木健太 | ITのための基礎知識 | 95.0 |
| 松本さくら | ITのための基礎知識 | 93.5 |
| 新垣愛留 | ITのための基礎知識 | 92.0 |
| ... | ... | ... |

## JOIN時のカラム指定

テーブルを結合する際に、特に同じカラム名が両方のテーブルに存在する場合は、カラム名の前にテーブル名（または別名）を付けることで、どのテーブルのカラムを参照しているかを明確にする必要があります。

例えば：

```
SELECT students.student_id, students.student_name
FROM students;
```

これは次のように短縮できます：

```
SELECT s.student_id, s.student_name
FROM students s;
```

ここで`s`は`students`テーブルの別名です。

# テーブルの別名（エイリアス）

長いテーブル名は別名を使って短くすることができます。これにより、SQLが読みやすくなります。

> **用語解説**：
>
> - **テーブル別名（エイリアス）**：テーブルに一時的につける短い名前で、クエリ内でテーブルを参照するときに使用します。

構文：

```
SELECT t.カラム名
FROM テーブル名 AS t;
```

「AS」キーワードは省略可能です：

```
SELECT t.カラム名
FROM テーブル名 t;
```

### 例4：別名を使った結合

```
SELECT s.student_name, c.course_name, t.teacher_name
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
JOIN courses c ON sc.course_id = c.course_id
JOIN teachers t ON c.teacher_id = t.teacher_id
WHERE s.student_id = 301;
```

この例では4つのテーブルを結合して、学生ID=301の学生が受講しているすべての講座とその担当教師の情報を取得しています。

実行結果：

| student_name | course_name | teacher_name |
|---|---|---|
| 黒沢春馬 | ITのための基礎知識 | 寺内鞍 |
| 黒沢春馬 | UNIX入門 | 田尻朋美 |
| 黒沢春馬 | クラウドコンピューティング | 吉岡由佳 |
| … | … | … |

# 結合条件の書き方

結合条件には複数の方法があります：

### 1. 等価結合（Equi-Join）

最も一般的な結合方法で、カラムの値が等しいことを条件に使います：

```
... ON テーブル1.カラム = テーブル2.カラム
```

### 2. 非等価結合（Non-Equi Join）

「等しい」以外の条件（<, >, <=, >=など）を使う結合方法です：

```
... ON テーブル1.カラム > テーブル2.カラム
```

### 3. 複合条件結合

複数の条件を組み合わせる結合方法です：

```
... ON テーブル1.カラム1 = テーブル2.カラム1
    AND テーブル1.カラム2 = テーブル2.カラム2
```

## JOINの特徴と注意点

JOINを使用する際には、以下の点に注意が必要です：

1. **一致しないレコードは含まれない**：結合条件に一致しないレコードは結果に含まれません。
2. **NULL値の扱い**：JOIN条件で使用するカラムにNULL値があると、その行は結果に含まれません。
3. **パフォーマンス**：大きなテーブル同士を結合すると、処理に時間がかかることがあります。

## 練習問題

### 問題13-1

courses（講座）テーブルとteachers（教師）テーブルを結合して、講座ID（course_id）、講座名（course_name）、担当教師名（teacher_name）を取得するSQLを書いてください。

### 問題13-2

students（学生）テーブルとstudent_courses（受講）テーブルを結合して、講座ID（course_id）が2の講座を受講している学生の学生ID（student_id）と学生名（student_name）を取得するSQLを書いてください。

### 問題13-3

course_schedule（授業カレンダー）テーブルとclassrooms（教室）テーブルを結合して、2025年5月20日に行われる授業のスケジュールIDと教室名（classroom_name）を取得するSQLを書いてください。

### 問題13-4

courses（講座）テーブル、teachers（教師）テーブル、course_schedule（授業カレンダー）テーブルの3つを結合して、2025年5月22日に行われる授業の講座名（course_name）と担当教師名（teacher_name）を取得するSQLを書いてください。

### 問題13-5

students（学生）テーブル、grades（成績）テーブル、courses（講座）テーブルを結合して、学生ID（student_id）が301の学生の成績情報（講座名、成績タイプ、点数）を取得するSQLを書いてください。

### 問題13-6

courses（講座）テーブル、course_schedule（授業カレンダー）テーブル、classrooms（教室）テーブル、class_periods（授業時間）テーブルの4つを結合して、2025年5月21日の授業スケジュール（講座名、教室名、開始時間、終了時間）を時間順に取得するSQLを書いてください。

## 解答

### 解答13-1

```
SELECT c.course_id, c.course_name, t.teacher_name
FROM courses c
JOIN teachers t ON c.teacher_id = t.teacher_id;
```

### 解答13-2

```
SELECT s.student_id, s.student_name
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
WHERE sc.course_id = '2';
```

### 解答13-3

```
SELECT cs.schedule_id, cl.classroom_name
FROM course_schedule cs
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
WHERE cs.schedule_date = '2025-05-20';
```

### 解答13-4

```
SELECT c.course_name, t.teacher_name
FROM course_schedule cs
JOIN courses c ON cs.course_id = c.course_id
JOIN teachers t ON cs.teacher_id = t.teacher_id
WHERE cs.schedule_date = '2025-05-22';
```

解答13-5

```
SELECT c.course_name, g.grade_type, g.score
FROM students s
JOIN grades g ON s.student_id = g.student_id
JOIN courses c ON g.course_id = c.course_id
WHERE s.student_id = 301;
```

解答13-6

```
SELECT c.course_name, cl.classroom_name, cp.start_time, cp.end_time
FROM course_schedule cs
JOIN courses c ON cs.course_id = c.course_id
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
JOIN class_periods cp ON cs.period_id = cp.period_id
WHERE cs.schedule_date = '2025-05-21'
ORDER BY cp.start_time;
```

## まとめ

この章では、複数のテーブルを結合して情報を取得するための基本的なJOIN操作について学びました：

1. **JOIN（結合）の概念**：複数のテーブルの情報を組み合わせる方法
2. **JOIN（内部結合）**：両方のテーブルで一致するレコードのみを取得
3. **結合条件**：ON句を使って結合条件を指定する方法
4. **テーブル別名**：テーブルに短い名前を付けて使う方法
5. **複数テーブルの結合**：3つ以上のテーブルを結合する方法
6. **結合条件の書き方**：等価結合、非等価結合、複合条件結合

JOINは実際のデータベース操作で非常に重要な機能です。データベースの性能を維持するために、関連情報は複数のテーブルに分散して保存されることが一般的で、必要な情報を取得するためにはこれらのテーブルを結合する必要があります。

次の章では、「テーブル別名：AS句とその省略」について詳しく学び、より効率的なJOINクエリの書き方を学びます。

---

# 14. テーブル別名：AS句とその省略

## はじめに

前章では複数のテーブルを結合する基本的な方法について学びました。テーブル結合を含むSQLクエリは、複数のテーブル名を扱うため長く複雑になりがちです。また、同じテーブル名やカラム名を繰り返し記述することも多くなります。

SQLでは、このような繰り返しを避け、クエリを簡潔に書くために「テーブル別名（エイリアス）」という機能が用意されています。この章では、テーブルやカラムに一時的な名前（別名）を付ける方法と、それを効果的に使用する方法について学びます。

# テーブル別名とは

テーブル別名（エイリアス）とは、SQLクエリの中で使用する仮の短い名前のことです。特に複数のテーブルを結合する場合や同じテーブルを複数回参照する場合に便利です。

> **用語解説**：
>
> - **テーブル別名（エイリアス）**：テーブルに一時的につける短い別名です。クエリ内でテーブルを参照するときに使用します。
> - **AS句**：「〜として」という意味のSQLキーワードで、テーブルやカラムに別名を付けるために使います。

# AS句を使ったテーブル別名の指定

テーブル別名を指定するには、FROMやJOIN句の中でテーブル名の後にASキーワードとともに別名を書きます。

## 基本構文

```
SELECT 列リスト
FROM テーブル名 AS 別名
[JOIN 別のテーブル AS 別名 ON 結合条件];
```

## 例1：AS句を使ったテーブル別名の指定

例えば、コースとその担当教師を取得するクエリでASを使用して別名を付けます：

```
SELECT c.course_id, c.course_name, t.teacher_name
FROM courses AS c
JOIN teachers AS t ON c.teacher_id = t.teacher_id;
```

この例では：

- `courses`テーブルに`c`という別名
- `teachers`テーブルに`t`という別名 を付けています。

# AS句の省略

SQLではAS句を省略することができます。これにより、クエリをさらに簡潔に書くことができます。

## 基本構文（AS省略）

```
SELECT 列リスト
FROM テーブル名 別名
[JOIN 別のテーブル 別名 ON 結合条件];
```

### 例2：AS句を省略したテーブル別名の指定

先ほどの例からASを省略してみましょう：

```
SELECT c.course_id, c.course_name, t.teacher_name
FROM courses c
JOIN teachers t ON c.teacher_id = t.teacher_id;
```

この書き方は前の例と全く同じ結果を返します。ASを省略してもテーブル名の直後に別名を書くことで同じ効果が得られます。

実行結果：

| course_id | course_name | teacher_name |
|---|---|---|
| 1 | ITのための基礎知識 | 寺内鞍 |
| 2 | UNIX入門 | 田尻朋美 |
| 3 | Cプログラミング演習 | 寺内鞍 |
| … | … | … |

## テーブル別名を使う利点

テーブル別名を使用することには、いくつかの利点があります：

1. **クエリの短縮**：長いテーブル名を短い別名で参照できるため、SQLの記述量が減ります。
2. **可読性の向上**：特に複数のテーブルを結合する複雑なクエリでは、どのテーブルのカラムを参照しているかが明確になります。
3. **同一テーブルの複数回使用**：後述する「自己結合」のように、同じテーブルを複数回使う場合に区別するために必須です。
4. **タイピングの労力削減**：繰り返し長いテーブル名を入力する必要がなくなります。

### 例3：複数テーブル結合での別名の活用

4つのテーブルを結合する複雑なクエリでテーブル別名を活用してみましょう：

```
SELECT s.student_name, c.course_name, cs.schedule_date, cl.classroom_name
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
JOIN courses c ON sc.course_id = c.course_id
```

```
JOIN course_schedule cs ON c.course_id = cs.course_id
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
WHERE s.student_id = 301 AND cs.schedule_date >= '2025-05-01';
```

このクエリでは以下の別名を使用しています：

- `students`テーブル → `s`
- `student_courses`テーブル → `sc`
- `courses`テーブル → `c`
- `course_schedule`テーブル → `cs`
- `classrooms`テーブル → `cl`

もし別名を使わないと、以下のような長く読みにくいクエリになります：

```
SELECT students.student_name, courses.course_name, course_schedule.schedule_date,
classrooms.classroom_name
FROM students
JOIN student_courses ON students.student_id = student_courses.student_id
JOIN courses ON student_courses.course_id = courses.course_id
JOIN course_schedule ON courses.course_id = course_schedule.course_id
JOIN classrooms ON course_schedule.classroom_id = classrooms.classroom_id
WHERE students.student_id = 301 AND course_schedule.schedule_date >= '2025-05-01';
```

## カラム別名の指定

テーブルだけでなく、カラム（列）にも別名を付けることができます。カラム別名を使うと、結果セットの列見出しをわかりやすい名前に変更できます。

### 基本構文

```
SELECT
    カラム名 AS 別名,
    計算式 AS 別名
FROM テーブル名;
```

### 例4：カラム別名の指定

```
SELECT
    student_id AS 学生番号,
    student_name AS 氏名,
    CONCAT(student_id, ': ', student_name) AS 学生情報
FROM students
WHERE student_id < 305;
```

実行結果：

| 学生番号 | 氏名 | 学生情報 |
|---|---|---|
| 301 | 黒沢春馬 | 301: 黒沢春馬 |
| 302 | 新垣愛留 | 302: 新垣愛留 |
| 303 | 柴崎春花 | 303: 柴崎春花 |
| 304 | 森下風凛 | 304: 森下風凛 |

### カラム別名でもASは省略可能

カラム別名の場合もAS句は省略可能です：

```
SELECT
    student_id 学生番号,
    student_name 氏名,
    CONCAT(student_id, ': ', student_name) 学生情報
FROM students
WHERE student_id < 305;
```

## スペースを含む別名

テーブル名やカラム名に含まれるスペースは通常問題になりますが、別名にスペースを含めたい場合は二重引用符（"）または（データベースによっては）バッククォート（`）で囲むことで指定できます。

### 例5：スペースを含む別名

```
SELECT
    student_id AS "学生 ID",
    student_name AS "学生 氏名"
FROM students
WHERE student_id < 305;
```

実行結果：

| 学生 ID | 学生 氏名 |
|---|---|
| 301 | 黒沢春馬 |
| 302 | 新垣愛留 |
| 303 | 柴崎春花 |
| 304 | 森下風凛 |

## OrderByとテーブル別名

一般的に、ORDER BY句ではSELECT句で指定したカラム別名を使用できます：

### 例6：ORDER BY句での別名の使用

```
SELECT
    student_id AS 学生番号,
    student_name AS 氏名
FROM students
WHERE student_id < 310
ORDER BY 氏名;
```

実行結果（氏名の五十音順）：

| 学生番号 | 氏名 |
|---|---|
| 309 | 相沢吉夫 |
| 305 | 河口菜恵子 |
| 306 | 河田咲奈 |
| 301 | 黒沢春馬 |
| 304 | 森下風凛 |
| 302 | 新垣愛留 |
| 303 | 柴崎春花 |
| 307 | 織田柚夏 |
| 308 | 永田悦子 |

## 別名の命名規則とベストプラクティス

テーブル別名やカラム別名を付ける際の一般的なルールとベストプラクティスを紹介します：

1. **テーブル別名は短く**：通常、1〜3文字程度の短い名前が好まれます（例：students → s）。
2. **意味のある別名**：テーブルの内容を表す頭文字や略語を使うと良いでしょう（例：teachers → t、course_schedule → cs）。
3. **一貫性**：プロジェクト内で同じテーブルには同じ別名を使うと可読性が向上します。
4. **カラム別名は説明的に**：カラム別名は、その内容がわかりやすい名前にします。特に計算列や関数の結果には説明的な名前を付けましょう。
5. **日本語の別名**：日本語環境では、カラム別名に日本語を使うとレポートが読みやすくなることがあります。

### 例7：ベストプラクティスに基づく別名

```
SELECT
    c.course_name AS 講座名,
```

```
    t.teacher_name AS 担当教員,
    COUNT(sc.student_id) AS 受講者数
FROM courses c
JOIN teachers t ON c.teacher_id = t.teacher_id
JOIN student_courses sc ON c.course_id = sc.course_id
GROUP BY c.course_id, c.course_name, t.teacher_name
ORDER BY 受講者数 DESC, 講座名;
```

実行結果：

| 講座名 | 担当教員 | 受講者数 |
|---|---|---|
| ITのための基礎知識 | 寺内鞍 | 12 |
| データサイエンスとビジネス応用 | 星野涼子 | 11 |
| クラウドネイティブアーキテクチャ | 吉岡由佳 | 10 |
| … | … | … |

## テーブル別名を使用したJOINの応用

ここまで学んだテーブル別名の知識を使って、より実践的なJOINクエリを見てみましょう。

### 例8：出席状況と成績情報の結合

```
SELECT
    s.student_name AS 学生名,
    c.course_name AS 講座名,
    a.status AS 出席状況,
    g.score AS 得点,
    g.score / g.max_score * 100 AS 達成率
FROM students s
JOIN attendance a ON s.student_id = a.student_id
JOIN course_schedule cs ON a.schedule_id = cs.schedule_id
JOIN courses c ON cs.course_id = c.course_id
LEFT JOIN grades g ON s.student_id = g.student_id AND cs.course_id = g.course_id
WHERE cs.schedule_date = '2025-05-20'
ORDER BY c.course_name, s.student_name;
```

このクエリでは：

- 5つのテーブルを結合
- すべてのテーブルに別名を使用
- 結果のカラムにもわかりやすい別名を付与
- 成績は存在しない可能性があるためLEFT JOINを使用

## 練習問題

### 問題14-1

courses（講座）テーブルとteachers（教師）テーブルを結合して、講座名（course_name）と担当教師名（teacher_name）を「講座」と「担当者」という別名をつけて取得するSQLを書いてください。テーブル別名も使用してください。

### 問題14-2

students（学生）テーブルとgrades（成績）テーブルを結合して、学生名、成績タイプ、点数を「学生」「評価種別」「点数」という別名をつけて取得するSQLを書いてください。点数が90点以上のレコードだけを抽出し、点数の高い順にソートしてください。

### 問題14-3

course_schedule（授業カレンダー）テーブル、courses（講座）テーブル、classrooms（教室）テーブルを結合して、2025年5月21日の授業予定を「時間割」という形で取得するSQLを書いてください。結果には「講座名」「教室」「状態」という別名を付け、テーブル別名も使用してください。

### 問題14-4

学生の出席状況を確認するため、students（学生）テーブル、attendance（出席）テーブル、course_schedule（授業カレンダー）テーブルを結合して、学生ID=301の出席記録を取得するSQLを書いてください。結果には「日付」「状態」「コメント」という別名をつけ、日付順にソートしてください。

### 問題14-5

成績の集計情報を取得するため、students（学生）テーブル、grades（成績）テーブル、courses（講座）テーブルを結合し、学生ごとの平均点を計算するSQLを書いてください。結果には「学生名」「受講講座数」「平均点」という別名をつけ、平均点が高い順にソートしてください。

### 問題14-6

course_schedule（授業カレンダー）テーブル、teachers（教師）テーブル、teacher_unavailability（講師スケジュール管理）テーブルを結合して、講師が不在の日に予定されていた授業（status = 'cancelled'）を取得するSQLを書いてください。結果には「日付」「講師名」「不在理由」という別名をつけ、日付順にソートしてください。

## 解答

### 解答14-1

```
SELECT
    c.course_name AS 講座,
    t.teacher_name AS 担当者
FROM courses c
JOIN teachers t ON c.teacher_id = t.teacher_id;
```

### 解答14-2

```
SELECT
    s.student_name AS 学生,
    g.grade_type AS 評価種別,
    g.score AS 点数
FROM students s
JOIN grades g ON s.student_id = g.student_id
WHERE g.score >= 90
ORDER BY 点数 DESC;
```

### 解答14-3

```
SELECT
    c.course_name AS 講座名,
    cl.classroom_name AS 教室,
    cs.status AS 状態
FROM course_schedule cs
JOIN courses c ON cs.course_id = c.course_id
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
WHERE cs.schedule_date = '2025-05-21'
ORDER BY cs.period_id;
```

### 解答14-4

```
SELECT
    cs.schedule_date AS 日付,
    a.status AS 状態,
    a.comment AS コメント
FROM students s
JOIN attendance a ON s.student_id = a.student_id
JOIN course_schedule cs ON a.schedule_id = cs.schedule_id
WHERE s.student_id = 301
ORDER BY 日付;
```

### 解答14-5

```
SELECT
    s.student_name AS 学生名,
    COUNT(DISTINCT g.course_id) AS 受講講座数,
    AVG(g.score) AS 平均点
FROM students s
JOIN grades g ON s.student_id = g.student_id
JOIN courses c ON g.course_id = c.course_id
GROUP BY s.student_id, s.student_name
ORDER BY 平均点 DESC;
```

解答14-6

```
SELECT
    cs.schedule_date AS 日付,
    t.teacher_name AS 講師名,
    tu.reason AS 不在理由
FROM course_schedule cs
JOIN teachers t ON cs.teacher_id = t.teacher_id
JOIN teacher_unavailability tu ON t.teacher_id = tu.teacher_id
WHERE cs.status = 'cancelled'
  AND cs.schedule_date BETWEEN tu.start_date AND tu.end_date
ORDER BY 日付;
```

## まとめ

この章では、SQLクエリを簡潔で読みやすくするためのテーブル別名とカラム別名について学びました：

1. **テーブル別名の基本**：AS句を使ってテーブルに短い別名を付ける方法
2. **AS句の省略**：テーブル別名を指定する際にASキーワードを省略する書き方
3. **テーブル別名の利点**：クエリの短縮、可読性の向上、同一テーブルの複数回使用
4. **カラム別名**：SELECT句の結果セットの列に別名を付ける方法
5. **スペースを含む別名**：引用符を使って空白を含む別名を指定する方法
6. **ORDER BYでの別名使用**：ソート条件にカラム別名を使用する方法
7. **命名規則とベストプラクティス**：効果的な別名の付け方

テーブル別名とカラム別名はSQLの読みやすさと保守性を大きく向上させる重要な機能です。特に複数のテーブルを結合する複雑なクエリでは、適切な別名を使うことでコードの量が減り、理解しやすくなります。

次の章では、「結合の種類：JOIN、LEFT JOIN、RIGHT JOIN」について学び、さまざまな結合方法とその使い分けについて理解を深めていきます。

---

# 15. 結合の種類：INNER JOIN、LEFT JOIN、RIGHT JOIN

## はじめに

これまでの章で、テーブルを結合する基本的な方法であるINNER JOINと、クエリを簡潔にするためのテーブル別名について学びました。しかし実際のデータベース操作では、テーブル間の関係はさまざまであり、結合方法もそれに合わせて選ぶ必要があります。

例えば：

- 「すべての学生と、存在すれば成績情報も取得したい」
- 「課題を提出していない学生も含めて一覧を取得したい」
- 「担当講座のない教師も含めて教師一覧を表示したい」

このような要件に対応するためには、さまざまな種類の結合方法を理解する必要があります。この章では、主要な結合の種類（INNER JOIN、LEFT JOIN、RIGHT JOIN）とその使い分けについて学びます。

## 結合の種類とは

SQLでは、テーブルの結合方法として主に以下の種類があります：

1. **INNER JOIN（内部結合）**：両方のテーブルで一致するレコードのみを返します。
2. **LEFT JOIN（左外部結合）**：左テーブルのすべてのレコードと、右テーブルの一致するレコードを返します。
3. **RIGHT JOIN（右外部結合）**：右テーブルのすべてのレコードと、左テーブルの一致するレコードを返します。
4. **FULL JOIN（完全外部結合）**：両方のテーブルのすべてのレコードを返します（MySQLでは直接サポートされていません）。

> **用語解説**：
>
> - **内部結合**：両方のテーブルで条件に一致するレコードだけを返す結合方法。
> - **外部結合**：一方のテーブルのレコードがもう一方のテーブルに一致するレコードがなくても結果に含める結合方法。
> - **左テーブル**：FROM句で最初に指定されるテーブル。
> - **右テーブル**：JOIN句で指定されるテーブル。

## INNER JOIN（内部結合）の復習

INNER JOINは最も基本的な結合タイプで、13章で学んだ通り、両方のテーブルで結合条件に一致するレコードのみを返します。

### 基本構文

```
SELECT カラム名
FROM テーブル1
INNER JOIN テーブル2 ON 結合条件;
```

### 例1：INNER JOINの基本

学生と彼らが提出した成績情報を結合してみましょう：

```
SELECT s.student_id, s.student_name, g.course_id, g.score
FROM students s
INNER JOIN grades g ON s.student_id = g.student_id
WHERE g.grade_type = '中間テスト'
ORDER BY s.student_id
LIMIT 5;
```

実行結果：

| student_id | student_name | course_id | score |
|---|---|---|---|
| 301 | 黒沢春馬 | 1 | 85.5 |
| 302 | 新垣愛留 | 1 | 92.0 |
| 303 | 柴崎春花 | 1 | 78.5 |
| 306 | 河田咲奈 | 1 | 88.0 |
| 307 | 織田柚夏 | 1 | 76.5 |

この結果には、成績テーブルに中間テストの記録がある学生だけが含まれています。成績記録のない学生は結果に含まれません。

## LEFT JOIN（左外部結合）

LEFT JOINは、左側のテーブル（FROM句のテーブル）のすべてのレコードを返し、右側のテーブル（JOIN句のテーブル）からは一致するレコードだけを返します。右側のテーブルに一致するレコードがない場合、そのカラムはNULLで埋められます。

> **用語解説**：
>
> - **LEFT JOIN**：左テーブルのすべてのレコードと、右テーブルの一致するレコードを返す結合方法です。
> - **NULL埋め**：一致するレコードがない場合、結果セット内の該当カラムはNULL値で埋められます。

### 基本構文

```
SELECT カラム名
FROM テーブル1
LEFT JOIN テーブル2 ON 結合条件;
```

### 例2：LEFT JOINの基本

すべての学生と、存在すれば彼らの成績情報を取得してみましょう：

```
SELECT s.student_id, s.student_name, g.course_id, g.score
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id AND g.grade_type = '中間テスト'
ORDER BY s.student_id
LIMIT 10;
```

実行結果：

| student_id | student_name | course_id | score |
|---|---|---|---|
| 301 | 黒沢春馬 | 1 | 85.5 |

| student_id | student_name | course_id | score |
|---|---|---|---|
| 302 | 新垣愛留 | 1 | 92.0 |
| 303 | 柴崎春花 | 1 | 78.5 |
| 304 | 森下風凛 | NULL | NULL |
| 305 | 河口菜恵子 | NULL | NULL |
| 306 | 河田咲奈 | 1 | 88.0 |
| 307 | 織田柚夏 | 1 | 76.5 |
| 308 | 永田悦子 | 1 | 91.0 |
| 309 | 相沢吉夫 | NULL | NULL |
| 310 | 吉川伽羅 | 1 | 82.5 |

この結果には、すべての学生が含まれています。成績記録がない学生（student_id = 304, 305, 309）の場合、course_idとscoreはNULLになっています。

### 例3：未提出者を見つける

LEFT JOINを使って、特定の課題を提出していない学生を見つけることができます：

```
SELECT s.student_id, s.student_name
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id AND g.grade_type = 'レポート1'
WHERE g.student_id IS NULL AND s.student_id < 320
ORDER BY s.student_id;
```

実行結果：

| student_id | student_name |
|---|---|
| 304 | 森下風凛 |
| 305 | 河口菜恵子 |
| 309 | 相沢吉夫 |
| 313 | 佐藤大輔 |
| 314 | 中村彩香 |
| 316 | 渡辺美咲 |
| 319 | 加藤悠真 |

このクエリは、grades テーブルにレポート1の記録がない学生を探しています。WHEREで「g.student_id IS NULL」を指定することで、結合後にNULLになったレコード（＝提出していない学生）だけを抽出しています。

# RIGHT JOIN（右外部結合）

RIGHT JOINは、LEFT JOINの逆で、右側のテーブル（JOIN句のテーブル）のすべてのレコードを返し、左側のテーブル（FROM句のテーブル）からは一致するレコードだけを返します。

> **用語解説**：
>
> - **RIGHT JOIN**：右テーブルのすべてのレコードと、左テーブルの一致するレコードを返す結合方法です。

## 基本構文

```
SELECT カラム名
FROM テーブル1
RIGHT JOIN テーブル2 ON 結合条件;
```

## 例4：RIGHT JOINの基本

講座テーブルを基準に、その講座を受講している学生を取得してみましょう：

```
SELECT c.course_id, c.course_name, sc.student_id
FROM student_courses sc
RIGHT JOIN courses c ON sc.course_id = c.course_id AND sc.student_id = 301
ORDER BY c.course_id
LIMIT 10;
```

実行結果：

| course_id | course_name | student_id |
|---|---|---|
| 1 | ITのための基礎知識 | 301 |
| 2 | UNIX入門 | 301 |
| 3 | Cプログラミング演習 | NULL |
| 4 | Webアプリケーション開発 | NULL |
| 5 | データベース設計と実装 | NULL |
| 6 | ネットワークセキュリティ | NULL |
| 7 | AI・機械学習入門 | NULL |
| 8 | モバイルアプリ開発 | NULL |
| 9 | クラウドコンピューティング | 301 |
| 10 | プロジェクト管理手法 | NULL |

このクエリの結果には、すべての講座が含まれています。学生ID=301（黒沢春馬）が受講している講座（course_id = 1, 2, 9）では、student_idが表示され、それ以外の講座ではNULLになっています。

## LEFT JOINとRIGHT JOINの変換

LEFT JOINとRIGHT JOINは、テーブルの順序を入れ替えることで相互に変換できます。一般的には、LEFT JOINの方が直感的に理解しやすいため、多くの場合はLEFT JOINが好まれます。

次の2つのクエリは同等です：

```
-- LEFT JOINを使用
SELECT *
FROM テーブル1
LEFT JOIN テーブル2 ON 結合条件;

-- RIGHT JOINを使用（テーブルの順序を入れ替え）
SELECT *
FROM テーブル2
RIGHT JOIN テーブル1 ON 結合条件;
```

## 複数テーブルでの外部結合

外部結合は複数のテーブルを結合する場合にも使用できます。

### 例5：3つのテーブルを使った外部結合

学生、受講テーブル、講座テーブルを結合して、すべての学生と彼らが受講している講座（あれば）を取得してみましょう：

```
SELECT s.student_id, s.student_name, c.course_id, c.course_name
FROM students s
LEFT JOIN student_courses sc ON s.student_id = sc.student_id
LEFT JOIN courses c ON sc.course_id = c.course_id
WHERE s.student_id BETWEEN 301 AND 305
ORDER BY s.student_id, c.course_id;
```

実行結果：

| student_id | student_name | course_id | course_name |
|---|---|---|---|
| 301 | 黒沢春馬 | 1 | ITのための基礎知識 |
| 301 | 黒沢春馬 | 2 | UNIX入門 |
| 301 | 黒沢春馬 | 9 | クラウドコンピューティング |
| 301 | 黒沢春馬 | 16 | クラウドネイティブアーキテクチャ |
| ... | ... | ... | ... |

| student_id | student_name | course_id | course_name |
|---|---|---|---|
| 302 | 新垣愛留 | 1 | ITのための基礎知識 |
| 302 | 新垣愛留 | 7 | AI・機械学習入門 |
| 302 | 新垣愛留 | 10 | プロジェクト管理手法 |
| ... | ... | ... | ... |
| 303 | 柴崎春花 | 1 | ITのための基礎知識 |
| 303 | 柴崎春花 | 4 | Webアプリケーション開発 |
| 303 | 柴崎春花 | 10 | プロジェクト管理手法 |
| ... | ... | ... | ... |
| 304 | 森下風凛 | 5 | データベース設計と実装 |
| 304 | 森下風凛 | 8 | モバイルアプリ開発 |
| 304 | 森下風凛 | 12 | サイバーセキュリティ対策 |
| ... | ... | ... | ... |
| 305 | 河口菜恵子 | 4 | Webアプリケーション開発 |
| 305 | 河口菜恵子 | 7 | AI・機械学習入門 |
| 305 | 河口菜恵子 | 11 | データ分析と可視化 |
| ... | ... | ... | ... |

このクエリでは2つのLEFT JOINを使用しています。最初のLEFT JOINは学生と受講テーブル、2つ目のLEFT JOINは受講テーブルと講座テーブルを結合しています。

## INNER JOINとLEFT JOINの使い分け

INNER JOINとLEFT JOIN（外部結合）は、用途に応じて使い分ける必要があります。以下のような基準で選択すると良いでしょう：

### INNER JOINを使う場合

- 両方のテーブルに対応するレコードが存在する場合のみデータを取得したい
- 関連付けられていないデータは不要な場合
- データの存在を確認したい場合

### LEFT JOIN（外部結合）を使う場合

- 主テーブルのすべてのレコードを表示したい
- 関連データがあるかどうかにかかわらず、主テーブルの情報は必要な場合
- 欠損データ（未提出、未登録など）を見つけたい場合
- レポート作成時に集計漏れを防ぎたい場合

## NULL値の処理に注意

外部結合を使用する場合、NULL値の処理に注意が必要です。特にWHERE句でフィルタリングする場合、通常の比較演算子（=, <, >など）はNULL値に対して常にFALSEを返します。

NULL値を検出するには「IS NULL」演算子を使用し、NULL値を除外するには「IS NOT NULL」演算子を使用します。

### 例6：NULL値の処理

```
-- 受講している講座がない学生を検索（NULLを検出）
SELECT s.student_id, s.student_name
FROM students s
LEFT JOIN student_courses sc ON s.student_id = sc.student_id
WHERE sc.course_id IS NULL;

-- 少なくとも1つの講座を受講している学生を検索（NULLを除外）
SELECT DISTINCT s.student_id, s.student_name
FROM students s
LEFT JOIN student_courses sc ON s.student_id = sc.student_id
WHERE sc.course_id IS NOT NULL;
```

## ON句とWHERE句の違い

LEFT JOINやRIGHT JOINを使用する場合、条件をON句に書くかWHERE句に書くかで結果が大きく変わることがあります。

- **ON句の条件**：結合操作の一部として適用され、外部結合の場合もNULL行を保持します。
- **WHERE句の条件**：結合後のすべての行に適用され、条件を満たさない行は結果から除外されます。

### 例7：ON句とWHERE句の違い

```
-- ON句に条件を書いた場合（外部結合の特性が保たれる）
SELECT s.student_id, s.student_name, g.score
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id AND g.grade_type = '中間テスト'
ORDER BY s.student_id
LIMIT 5;

-- WHERE句に条件を書いた場合（内部結合と同等になる）
SELECT s.student_id, s.student_name, g.score
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id
WHERE g.grade_type = '中間テスト'
ORDER BY s.student_id
LIMIT 5;
```

最初のクエリでは、すべての学生が結果に含まれ、中間テストのスコアがある場合はその値が表示され、ない場合はNULLになります。

2つ目のクエリでは、WHERE句でg.grade_type = '中間テスト'という条件を指定しているため、中間テストのスコアがない学生は結果から除外されます（実質的にINNER JOINと同等になります）。

## 練習問題

### 問題15-1

courses（講座）テーブルとteachers（教師）テーブルを使い、すべての講座とその担当教師（いれば）の情報を取得するSQLを書いてください。LEFT JOINを使用してください。

### 問題15-2

students（学生）テーブルとgrades（成績）テーブルを使い、講座ID（course_id）= 1の中間テストを受けていない学生を特定するSQLを書いてください。

### 問題15-3

teachers（教師）テーブルとcourses（講座）テーブルを使い、担当講座がない教師を特定するSQLを書いてください。

### 問題15-4

course_schedule（授業カレンダー）テーブルとattendance（出席）テーブルを使い、2025年5月20日の各授業に対する出席学生数と欠席学生数を集計するSQLを書いてください。

### 問題15-5

students（学生）テーブル、student_courses（受講）テーブル、courses（講座）テーブルを使い、各学生が受講している講座の数を取得するSQLを書いてください。受講していない学生も0として表示してください。

### 問題15-6

course_schedule（授業カレンダー）テーブル、teachers（教師）テーブル、teacher_unavailability（講師スケジュール管理）テーブルを使い、今後の授業予定（schedule_date >= '2025-05-20'）について、担当教師が不在期間と重なっているかどうかをチェックするSQLを書いてください。不在期間と重なっている場合は「要注意」、そうでない場合は「OK」と表示してください。

## 解答

### 解答15-1

```
SELECT c.course_id, c.course_name, t.teacher_id, t.teacher_name
FROM courses c
LEFT JOIN teachers t ON c.teacher_id = t.teacher_id
ORDER BY c.course_id;
```

### 解答15-2

```
SELECT s.student_id, s.student_name
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id
                   AND g.course_id = '1'
                   AND g.grade_type = '中間テスト'
WHERE g.student_id IS NULL
ORDER BY s.student_id;
```

### 解答15-3

```
SELECT t.teacher_id, t.teacher_name
FROM teachers t
LEFT JOIN courses c ON t.teacher_id = c.teacher_id
WHERE c.teacher_id IS NULL;
```

### 解答15-4

```
SELECT
    cs.schedule_id,
    c.course_name,
    SUM(CASE WHEN a.status = 'present' THEN 1 ELSE 0 END) AS 出席数,
    SUM(CASE WHEN a.status = 'absent' THEN 1 ELSE 0 END) AS 欠席数,
    SUM(CASE WHEN a.status = 'late' THEN 1 ELSE 0 END) AS 遅刻数,
    COUNT(a.student_id) AS 総数
FROM course_schedule cs
LEFT JOIN attendance a ON cs.schedule_id = a.schedule_id
LEFT JOIN courses c ON cs.course_id = c.course_id
WHERE cs.schedule_date = '2025-05-20'
GROUP BY cs.schedule_id, c.course_name
ORDER BY cs.schedule_id;
```

### 解答15-5

```
SELECT
    s.student_id,
    s.student_name,
    COUNT(sc.course_id) AS 受講講座数
FROM students s
LEFT JOIN student_courses sc ON s.student_id = sc.student_id
GROUP BY s.student_id, s.student_name
ORDER BY s.student_id;
```

### 解答15-6

```
SELECT
    cs.schedule_id,
    cs.schedule_date,
    t.teacher_name,
    c.course_name,
    CASE
        WHEN tu.teacher_id IS NOT NULL THEN '要注意 - ' || tu.reason
        ELSE 'OK'
    END AS 状態
FROM course_schedule cs
INNER JOIN teachers t ON cs.teacher_id = t.teacher_id
INNER JOIN courses c ON cs.course_id = c.course_id
LEFT JOIN teacher_unavailability tu ON cs.teacher_id = tu.teacher_id
                                    AND cs.schedule_date BETWEEN tu.start_date AND
tu.end_date
WHERE cs.schedule_date >= '2025-05-20'
ORDER BY cs.schedule_date, cs.period_id;
```

## まとめ

この章では、データベースのさまざまな結合方法について学びました：

1. **INNER JOIN（内部結合）**：両方のテーブルで条件に一致するレコードのみを返す
2. **LEFT JOIN（左外部結合）**：左テーブルのすべてのレコードと、右テーブルの一致するレコードを返す
3. **RIGHT JOIN（右外部結合）**：右テーブルのすべてのレコードと、左テーブルの一致するレコードを返す
4. **結合方法の使い分け**：目的に応じてINNER JOINと外部結合を使い分ける基準
5. **NULL値の処理**：外部結合結果のNULL値を適切に処理する方法
6. **ON句とWHERE句の違い**：条件の記述場所による結果の違い

適切な結合方法を選択することで、より柔軟で正確なデータの抽出が可能になります。特に、データの欠損（未提出、未登録など）を検出したい場合や、すべてのレコードを漏れなく処理したい場合には、外部結合が非常に役立ちます。

次の章では、「自己結合：同一テーブル内での関連付け」について学び、同じテーブル内でのデータ間の関係を扱う方法を学びます。

---

# 16. 自己結合：同一テーブル内での関連付け

## はじめに

これまでの章では、異なるテーブル同士を結合する方法について学びました。しかし実際のデータベース設計では、同じテーブル内にデータ同士の関連がある場合があります。例えば：

- 社員テーブルで「上司」も同じ社員テーブル内の別の社員である
- 部品表で「部品」と「その部品の構成部品」が同じテーブルに格納されている

- 家系図データで「親」と「子」が同じ人物テーブルに格納されている

このような「同一テーブル内のレコード同士の関連」を取り扱うための結合方法が「自己結合（セルフジョイン）」です。この章では、テーブル自身と結合して情報を取得する方法について学びます。

# 自己結合（セルフジョイン）とは

自己結合とは、テーブルを自分自身と結合する技術です。テーブル内のあるレコードと、同じテーブル内の別のレコードとの関係を表現するために使用されます。

> **用語解説**：
>
> - **自己結合（セルフジョイン）**：同じテーブルを異なる別名で参照し、テーブル自身と結合する手法です。
> - **再帰的関係**：同じエンティティ（テーブル）内での親子関係や階層構造などの関係のこと。

# 自己結合の基本

自己結合を行うには、同じテーブルを異なる別名で参照する必要があります。これは通常、テーブル別名（エイリアス）を使用して実現します。

### 基本構文

```
SELECT a.カラム1, a.カラム2, b.カラム1, b.カラム2
FROM テーブル名 a
JOIN テーブル名 b ON a.関連カラム = b.主キー;
```

ここで、aとbは同じテーブルに対する異なる別名です。

# 自己結合の実践例

学校データベースでは、明示的な再帰的関係を持つテーブルはありませんが、自己結合の概念を理解するために簡単な例を見てみましょう。

### 例1：同じ教師が担当する講座を検索

```
SELECT
    c1.course_id AS 講座ID,
    c1.course_name AS 講座名,
    c2.course_id AS 関連講座ID,
    c2.course_name AS 関連講座名,
    t.teacher_name AS 担当教師
FROM courses c1
JOIN courses c2 ON c1.teacher_id = c2.teacher_id AND c1.course_id < c2.course_id
JOIN teachers t ON c1.teacher_id = t.teacher_id
ORDER BY t.teacher_name, c1.course_id, c2.course_id;
```

このクエリは、同じ教師が担当している講座の組み合わせを検索しています。

- `c1`と`c2`は同じcoursesテーブルの別名
- `c1.teacher_id = c2.teacher_id`で同じ教師を担当している講座を結合
- `c1.course_id < c2.course_id`でペアの重複を避けています（例：講座1と講座2、講座2と講座1が両方表示されることを防ぐ）

実行結果：

| 講座ID | 講座名 | 関連講座ID | 関連講座名 | 担当教師 |
|---|---|---|---|---|
| 4 | Webアプリケーション開発 | 8 | モバイルアプリ開発 | 藤本理恵 |
| 4 | Webアプリケーション開発 | 22 | フルスタック開発マスタークラス | 藤本理恵 |
| 8 | モバイルアプリ開発 | 22 | フルスタック開発マスタークラス | 藤本理恵 |
| 1 | ITのための基礎知識 | 3 | Cプログラミング演習 | 寺内鞍 |
| 1 | ITのための基礎知識 | 29 | コードリファクタリングとクリーンコード | 寺内鞍 |
| 3 | Cプログラミング演習 | 29 | コードリファクタリングとクリーンコード | 寺内鞍 |
| … | … | … | … | … |

### 例2：同じ日に複数の授業がある教師を検索

```
SELECT
    cs1.schedule_date AS 日付,
    t.teacher_name AS 教師名,
    cs1.period_id AS 時限1,
    c1.course_name AS 講座1,
    cs2.period_id AS 時限2,
    c2.course_name AS 講座2
FROM course_schedule cs1
JOIN course_schedule cs2 ON cs1.teacher_id = cs2.teacher_id
                        AND cs1.schedule_date = cs2.schedule_date
                        AND cs1.period_id < cs2.period_id
JOIN teachers t ON cs1.teacher_id = t.teacher_id
JOIN courses c1 ON cs1.course_id = c1.course_id
JOIN courses c2 ON cs2.course_id = c2.course_id
WHERE cs1.schedule_date = '2025-05-21'
ORDER BY cs1.schedule_date, t.teacher_name, cs1.period_id, cs2.period_id;
```

このクエリは、同じ日に複数の授業を担当する教師と、その授業の組み合わせを検索しています。

- `cs1`と`cs2`は同じcourse_scheduleテーブルの別名

- `cs1.teacher_id = cs2.teacher_id AND cs1.schedule_date = cs2.schedule_date`で同じ教師が同じ日に担当している授業を結合
- `cs1.period_id < cs2.period_id`でペアの重複を避けています

実行結果（例）：

| 日付 | 教師名 | 時限1 | 講座1 | 時限2 | 講座2 |
|---|---|---|---|---|---|
| 2025-05-21 | 星野涼子 | 1 | 高度データ可視化技術 | 3 | データサイエンスとビジネス応用 |
| 2025-05-21 | 寺内鞍 | 2 | コードリファクタリングとクリーンコード | 4 | Cプログラミング演習 |
| … | … | … | … | … | … |

## 自己結合の応用：階層構造の表現

自己結合は、階層構造のデータを表現する際に特に役立ちます。例えば、組織図、カテゴリ階層、部品表などです。

ここでは、学校データベースにはない例として、架空の「社員」テーブルを使って階層構造を表現する例を示します：

### 例3：架空の社員テーブルを使った階層構造の表現

```
-- 架空の社員テーブル
CREATE TABLE employees (
  employee_id INT PRIMARY KEY,
  employee_name VARCHAR(100),
  manager_id INT,
  FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
);

-- サンプルデータ
INSERT INTO employees VALUES (1, '山田太郎', NULL);  -- 社長（上司なし）
INSERT INTO employees VALUES (2, '佐藤次郎', 1);     -- 山田の部下
INSERT INTO employees VALUES (3, '鈴木三郎', 1);     -- 山田の部下
INSERT INTO employees VALUES (4, '高橋四郎', 2);     -- 佐藤の部下
INSERT INTO employees VALUES (5, '田中五郎', 2);     -- 佐藤の部下
INSERT INTO employees VALUES (6, '伊藤六郎', 3);     -- 鈴木の部下

-- 社員と直属の上司を取得
SELECT
    e.employee_id,
    e.employee_name AS 社員名,
    m.employee_name AS 上司名
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.employee_id
ORDER BY e.employee_id;
```

結果：

| employee_id | 社員名 | 上司名 |
|---|---|---|
| 1 | 山田太郎 | NULL |
| 2 | 佐藤次郎 | 山田太郎 |
| 3 | 鈴木三郎 | 山田太郎 |
| 4 | 高橋四郎 | 佐藤次郎 |
| 5 | 田中五郎 | 佐藤次郎 |
| 6 | 伊藤六郎 | 鈴木三郎 |

この例では、社員テーブルの`manager_id`が同じテーブルの`employee_id`を参照しています（自己参照）。自己結合を使用して、各社員の上司の名前を取得しています。

### 例4：架空の社員テーブルを使った部下の一覧表示

```
-- ある上司の直属の部下を取得
SELECT
    m.employee_id AS 上司ID,
    m.employee_name AS 上司名,
    e.employee_id AS 部下ID,
    e.employee_name AS 部下名
FROM employees m
JOIN employees e ON m.employee_id = e.manager_id
WHERE m.employee_id = 2
ORDER BY e.employee_id;
```

結果：

| 上司ID | 上司名 | 部下ID | 部下名 |
|---|---|---|---|
| 2 | 佐藤次郎 | 4 | 高橋四郎 |
| 2 | 佐藤次郎 | 5 | 田中五郎 |

この例では、社員テーブルを`m`（上司）と`e`（部下）として自己結合し、上司ID=2の部下を取得しています。

## 学校データベースでの自己結合活用例

実際の学校データベースでは、自己結合を利用できる具体的なシナリオを見てみましょう。

### 例5：同じ教室を使用する授業の組み合わせ

```
SELECT
    cs1.schedule_date AS 日付,
```

```
    cs1.classroom_id AS 教室,
    cs1.period_id AS 時限1,
    c1.course_name AS 講座1,
    t1.teacher_name AS 教師1,
    cs2.period_id AS 時限2,
    c2.course_name AS 講座2,
    t2.teacher_name AS 教師2
FROM course_schedule cs1
JOIN course_schedule cs2 ON cs1.classroom_id = cs2.classroom_id
                        AND cs1.schedule_date = cs2.schedule_date
                        AND cs1.period_id < cs2.period_id
JOIN courses c1 ON cs1.course_id = c1.course_id
JOIN courses c2 ON cs2.course_id = c2.course_id
JOIN teachers t1 ON cs1.teacher_id = t1.teacher_id
JOIN teachers t2 ON cs2.teacher_id = t2.teacher_id
WHERE cs1.schedule_date = '2025-05-21'
ORDER BY cs1.classroom_id, cs1.period_id;
```

このクエリは、同じ日に同じ教室で行われる授業の組み合わせを検索しています。教室の利用状況や消毒・清掃のスケジュール計画などに役立ちます。

### 例6：同じ学生が受講している講座の組み合わせ

```
SELECT
    sc1.student_id,
    s.student_name,
    c1.course_name AS 講座1,
    c2.course_name AS 講座2
FROM student_courses sc1
JOIN student_courses sc2 ON sc1.student_id = sc2.student_id AND sc1.course_id <
sc2.course_id
JOIN students s ON sc1.student_id = s.student_id
JOIN courses c1 ON sc1.course_id = c1.course_id
JOIN courses c2 ON sc2.course_id = c2.course_id
WHERE sc1.student_id = 301
ORDER BY c1.course_name, c2.course_name;
```

このクエリは、学生ID=301の学生が受講しているすべての講座の組み合わせを検索しています。学生の履修パターンの分析や時間割の調整などに役立ちます。

## INNER JOINとLEFT JOINの自己結合

自己結合では、INNER JOIN、LEFT JOIN、RIGHT JOINなど、すべての結合タイプを使用できます。結合タイプの選択は、取得したいデータの性質に依存します。

### 例7：部下のいない社員を見つける（LEFT JOIN）

```sql
-- 架空の社員テーブルを使用
SELECT
    m.employee_id,
    m.employee_name,
    COUNT(e.employee_id) AS 部下の数
FROM employees m
LEFT JOIN employees e ON m.employee_id = e.manager_id
GROUP BY m.employee_id, m.employee_name
HAVING COUNT(e.employee_id) = 0
ORDER BY m.employee_id;
```

このクエリは、部下のいない社員（つまり、誰も自分を上司として参照していない社員）を検索しています。結果として、employee_id = 4, 5, 6の社員（高橋四郎、田中五郎、伊藤六郎）が表示されるでしょう。

## 自己結合の注意点

自己結合を使用する際には、以下の点に注意が必要です：

1. **テーブル別名の明確化**：同じテーブルを複数回参照するため、わかりやすいテーブル別名を使用し、混乱を避けましょう。
2. **結合条件の正確性**：自己結合では結合条件が不適切だと、結果が指数関数的に増えることがあります。必要に応じて絞り込み条件を追加しましょう。
3. **重複の排除**：例1や例2で使用した`id1 < id2`のような条件を使って、組み合わせの重複を避けることが重要です。
4. **パフォーマンス**：自己結合は、特に大きなテーブルでは処理が重くなる可能性があります。必要に応じてインデックスを使用して最適化しましょう。

## 練習問題

### 問題16-1

教師（teachers）テーブルを自己結合して、教師IDが連続する教師のペア（例：ID 101と102, 102と103）を取得するSQLを書いてください。

### 問題16-2

course_schedule（授業カレンダー）テーブルを自己結合して、2025年5月15日に同じ教室で行われる授業のペアを時限の昇順で取得するSQLを書いてください。course_id、period_id、classroom_idの3つを表示してください。

### 問題16-3

生徒（students）テーブルを自己結合して、学生名（student_name）のファーストネーム（名前の最初の文字）が同じ学生のペアを取得するSQLを書いてください。ヒント：SUBSTRING関数を使用します。

### 問題16-4

course_schedule（授業カレンダー）テーブルを自己結合して、同じ日に同じ講師が担当する授業のペアを見つけるSQLを書いてください。期間が離れていない授業（period_idの差が1または2）のみを対象とします。

### 問題16-5

grades（成績）テーブルを自己結合して、同じ学生の異なる課題タイプ（grade_type）間で点数差が10点以上あるケースを検出するSQLを書いてください。

### 問題16-6

以下のような架空の「職階」テーブルを作成し、直接の上下関係（部下と上司）だけでなく、組織階層全体を表示するSQLを書いてください。

```
-- 架空の職階テーブル
CREATE TABLE job_hierarchy (
  level_id INT PRIMARY KEY,
  level_name VARCHAR(50),
  reports_to INT,
  FOREIGN KEY (reports_to) REFERENCES job_hierarchy(level_id)
);

-- データ挿入
INSERT INTO job_hierarchy VALUES (1, '学長', NULL);
INSERT INTO job_hierarchy VALUES (2, '副学長', 1);
INSERT INTO job_hierarchy VALUES (3, '学部長', 2);
INSERT INTO job_hierarchy VALUES (4, '学科長', 3);
INSERT INTO job_hierarchy VALUES (5, '教授', 4);
INSERT INTO job_hierarchy VALUES (6, '准教授', 5);
INSERT INTO job_hierarchy VALUES (7, '講師', 6);
INSERT INTO job_hierarchy VALUES (8, '助教', 7);
```

## 解答

### 解答16-1

```
SELECT
    t1.teacher_id AS 教師ID1,
    t1.teacher_name AS 教師名1,
    t2.teacher_id AS 教師ID2,
    t2.teacher_name AS 教師名2
FROM teachers t1
JOIN teachers t2 ON t1.teacher_id + 1 = t2.teacher_id
ORDER BY t1.teacher_id;
```

### 解答16-2

```
SELECT
    cs1.course_id AS 講座ID1,
    cs1.period_id AS 時限1,
    cs2.course_id AS 講座ID2,
    cs2.period_id AS 時限2,
    cs1.classroom_id AS 教室
FROM course_schedule cs1
JOIN course_schedule cs2 ON cs1.classroom_id = cs2.classroom_id
                        AND cs1.schedule_date = cs2.schedule_date
                        AND cs1.period_id < cs2.period_id
WHERE cs1.schedule_date = '2025-05-15'
ORDER BY cs1.classroom_id, cs1.period_id, cs2.period_id;
```

解答16-3

```
SELECT
    s1.student_id AS 学生ID1,
    s1.student_name AS 学生名1,
    s2.student_id AS 学生ID2,
    s2.student_name AS 学生名2,
    SUBSTRING(s1.student_name, 1, 1) AS 共通文字
FROM students s1
JOIN students s2 ON SUBSTRING(s1.student_name, 1, 1) = SUBSTRING(s2.student_name,
1, 1)
               AND s1.student_id < s2.student_id
ORDER BY 共通文字, s1.student_id, s2.student_id;
```

解答16-4

```
SELECT
    cs1.schedule_date AS 日付,
    t.teacher_name AS 教師名,
    cs1.period_id AS 時限1,
    cs1.course_id AS 講座ID1,
    cs2.period_id AS 時限2,
    cs2.course_id AS 講座ID2,
    ABS(cs1.period_id - cs2.period_id) AS 時限差
FROM course_schedule cs1
JOIN course_schedule cs2 ON cs1.teacher_id = cs2.teacher_id
                        AND cs1.schedule_date = cs2.schedule_date
                        AND cs1.schedule_id < cs2.schedule_id
JOIN teachers t ON cs1.teacher_id = t.teacher_id
WHERE ABS(cs1.period_id - cs2.period_id) IN (1, 2)
ORDER BY cs1.schedule_date, t.teacher_name, cs1.period_id;
```

解答16-5

```
SELECT
    g1.student_id,
    s.student_name,
    g1.grade_type AS 評価タイプ1,
    g1.score AS 点数1,
    g2.grade_type AS 評価タイプ2,
    g2.score AS 点数2,
    ABS(g1.score - g2.score) AS 点数差
FROM grades g1
JOIN grades g2 ON g1.student_id = g2.student_id
               AND g1.course_id = g2.course_id
               AND g1.grade_type < g2.grade_type
JOIN students s ON g1.student_id = s.student_id
WHERE ABS(g1.score - g2.score) >= 10
ORDER BY 点数差 DESC, g1.student_id;
```

### 解答16-6

```
-- 組織階層全体を表示
WITH RECURSIVE hierarchy AS (
  -- 基点（学長）
  SELECT
    level_id,
    level_name,
    reports_to,
    0 AS depth,
    CAST(level_name AS CHAR(200)) AS path
  FROM job_hierarchy
  WHERE reports_to IS NULL

  UNION ALL

  -- 再帰部分
  SELECT
    j.level_id,
    j.level_name,
    j.reports_to,
    h.depth + 1,
    CONCAT(h.path, ' > ', j.level_name)
  FROM job_hierarchy j
  JOIN hierarchy h ON j.reports_to = h.level_id
)
SELECT
  level_id,
  level_name,
  reports_to,
  depth,
  CONCAT(REPEAT('    ', depth), level_name) AS 階層表示,
  path AS 組織経路
```

```
FROM hierarchy
ORDER BY depth, level_id;
```

注意：最後の問題は再帰共通テーブル式（Recursive CTE）を使用しています。これはMySQL 8.0以降でサポートされています。再帰CTEはまだ学習していない高度な内容ですが、階層構造を扱う効果的な方法として参考になります。

## まとめ

この章では、同一テーブル内でのレコード間の関連を扱うための「自己結合」について学びました：

1. **自己結合の概念**：テーブルを自分自身と結合して、同一テーブル内のレコード間の関係を表現する方法
2. **基本的な構文**：テーブル別名を使用して同じテーブルを複数回参照する方法
3. **実践的な例**：同じ教師が担当する講座、同じ日の複数の授業など、実用的な自己結合のケース
4. **階層構造の表現**：上司と部下、組織階層など、再帰的な関係の表現方法
5. **さまざまな結合タイプ**：INNER JOINやLEFT JOINなど、自己結合でも利用可能な結合の種類
6. **注意点**：テーブル別名の明確化、結合条件の正確性、重複の排除、パフォーマンスの考慮

自己結合は、階層構造の表現や、同一エンティティ内での関連を取り扱う強力な手法です。特に組織図、部品表、カテゴリ階層、家系図など、再帰的な関係を含むデータモデルで頻繁に活用されます。

次の章では、「複数テーブル結合：3つ以上のテーブルの連結」について学び、より複雑なデータ関係を扱う方法を学びます。

---

# 17. 複数テーブル結合：3つ以上のテーブルの連結

## はじめに

これまでの章では、2つのテーブルを結合する方法や、同じテーブルを自己結合する方法について学びました。しかし、実際のデータベース操作では、3つ以上のテーブルを一度に結合して情報を取得する必要があることが多くあります。

例えば、以下のようなケースを考えてみましょう：

- 「学生の名前、受講している講座名、その担当教師名、教室情報を一度に取得したい」
- 「授業スケジュール、講座情報、教室情報、時間情報をまとめて表示したい」
- 「成績データに学生名、講座名、提出情報を関連付けて分析したい」

これらを実現するには、3つ以上のテーブルを連結する必要があります。この章では、複数のテーブルを効果的に結合する方法について学びます。

## 複数テーブル結合の基本

3つ以上のテーブルを結合する基本的な考え方は、2つのテーブルの結合を拡張したものです。2つのテーブルを結合した結果に、さらに別のテーブルを結合していきます。

> **用語解説**：

> - **複数結合**：3つ以上のテーブルを一度に連結して情報を取得する操作のことです。
> - **結合チェーン**：複数のJOIN句を連続して記述し、テーブルをつなげていく方法です。

### 基本構文

```
SELECT カラムリスト
FROM テーブル1
JOIN テーブル2 ON 結合条件1
JOIN テーブル3 ON 結合条件2
JOIN テーブル4 ON 結合条件3
...;
```

JOINの種類（INNER JOIN、LEFT JOIN、RIGHT JOINなど）は、各結合ごとに適切に選択できます。

## 3つのテーブルを結合する例

まずは、3つのテーブルを結合する基本的な例を見てみましょう。

### 例1：学生、講座、教師の情報を連結する

```
SELECT
    s.student_id,
    s.student_name AS 学生名,
    c.course_name AS 講座名,
    t.teacher_name AS 担当教師
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
JOIN courses c ON sc.course_id = c.course_id
JOIN teachers t ON c.teacher_id = t.teacher_id
WHERE s.student_id = 301
ORDER BY c.course_id;
```

このクエリでは：

1. 学生テーブル(`students`)を基点に
2. 受講テーブル(`student_courses`)を結合し
3. 講座テーブル(`courses`)を結合し
4. 教師テーブル(`teachers`)を結合しています

実行結果：

| student_id | 学生名 | 講座名 | 担当教師 |
|---|---|---|---|
| 301 | 黒沢春馬 | ITのための基礎知識 | 寺内鞍 |
| 301 | 黒沢春馬 | UNIX入門 | 田尻朋美 |
| 301 | 黒沢春馬 | クラウドコンピューティング | 吉岡由佳 |

| student_id | 学生名 | 講座名 | 担当教師 |
|---|---|---|---|
| 301 | 黒沢春馬 | クラウドネイティブアーキテクチャ | 吉岡由佳 |
| ... | ... | ... | ... |

## 4つ以上のテーブルを結合する例

より複雑な情報を取得するために、4つ以上のテーブルを結合してみましょう。

### 例2：授業スケジュール詳細の取得

```
SELECT
    cs.schedule_date AS 日付,
    cp.start_time AS 開始時間,
    cp.end_time AS 終了時間,
    c.course_name AS 講座名,
    t.teacher_name AS 担当教師,
    cl.classroom_name AS 教室,
    cl.building AS 建物,
    cs.status AS 状態
FROM course_schedule cs
JOIN courses c ON cs.course_id = c.course_id
JOIN teachers t ON cs.teacher_id = t.teacher_id
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
JOIN class_periods cp ON cs.period_id = cp.period_id
WHERE cs.schedule_date = '2025-05-21'
ORDER BY cp.start_time, cl.classroom_id;
```

このクエリでは、5つのテーブルを結合して授業スケジュールの詳細情報を取得しています：

1. 授業カレンダーテーブル(`course_schedule`)を基点に
2. 講座テーブル(`courses`)を結合し
3. 教師テーブル(`teachers`)を結合し
4. 教室テーブル(`classrooms`)を結合し
5. 授業時間テーブル(`class_periods`)を結合しています

実行結果（例）：

| 日付 | 開始時間 | 終了時間 | 講座名 | 担当教師 | 教室 | 建物 | 状態 |
|---|---|---|---|---|---|---|---|
| 2025-05-21 | 09:00:00 | 10:30:00 | 高度データ可視化技術 | 星野涼子 | 402Hコンピュータ実習室 | 4号館 | scheduled |
| 2025-05-21 | 09:00:00 | 10:30:00 | サイバーセキュリティ対策 | 深山誠一 | 301E講義室 | 3号館 | scheduled |

| 日付 | 開始時間 | 終了時間 | 講座名 | 担当教師 | 教室 | 建物 | 状態 |
|---|---|---|---|---|---|---|---|
| 2025-05-21 | 10:40:00 | 12:10:00 | コードリファクタリングとクリーンコード | 寺内 鞍 | 101Aコンピュータ実習室 | 1号館 | scheduled |
| … | … | … | … | … | … | … | … |

## 複数テーブル結合のバリエーション

複数テーブルの結合では、さまざまな結合タイプを組み合わせることができます。

### 例3：INNER JOINとLEFT JOINの組み合わせ

```
SELECT
    s.student_id,
    s.student_name AS 学生名,
    c.course_name AS 講座名,
    g.grade_type AS 評価種別,
    g.score AS 点数,
    a.status AS 出席状況
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
JOIN courses c ON sc.course_id = c.course_id
LEFT JOIN grades g ON s.student_id = g.student_id AND sc.course_id = g.course_id
AND g.grade_type = '中間テスト'
LEFT JOIN attendance a ON s.student_id = a.student_id
WHERE s.student_id = 301
ORDER BY c.course_id;
```

このクエリでは、以下のように異なる結合タイプを組み合わせています：

- students、student_courses、coursesテーブルには**INNER JOIN**を使用（関連データが必ず存在するため）
- gradesテーブルには**LEFT JOIN**を使用（中間テストの成績がない可能性があるため）
- attendanceテーブルにも**LEFT JOIN**を使用（出席情報がない可能性があるため）

## 結合順序とパフォーマンス

複数テーブルを結合する場合、結合の順序はSQLの実行計画に影響しますが、通常はデータベースのオプティマイザが最適な実行順序を決定します。ただし、以下の点に注意すると効率的なクエリを書くことができます：

1. **フィルタリングを早めに行う**：WHERE句での絞り込みを早い段階で行うことで、結合対象のレコード数を減らせます。
2. **小さいテーブルから大きいテーブルへ**：一般的に、小さいテーブルを基点に、より大きいテーブルを結合していく方が効率的です。

3. **結合条件の最適化**：インデックスが設定されているカラムを結合条件に使用すると、パフォーマンスが向上します。

### 例4：効率的な結合順序とフィルタリングの例

```
SELECT
    cs.schedule_date,
    c.course_name,
    COUNT(a.student_id) AS 出席学生数
FROM course_schedule cs
JOIN courses c ON cs.course_id = c.course_id
LEFT JOIN attendance a ON cs.schedule_id = a.schedule_id AND a.status = 'present'
WHERE cs.schedule_date BETWEEN '2025-05-01' AND '2025-05-31'
GROUP BY cs.schedule_id, cs.schedule_date, c.course_name
ORDER BY cs.schedule_date, c.course_name;
```

このクエリでは、以下の効率化を行っています：

- 日付範囲による絞り込みを早い段階で行っている（`course_schedule`テーブルへのWHERE句）
- 比較的小さい`course_schedule`テーブルを基点にしている
- 出席状態のフィルタリングを結合条件に含めている（`a.status = 'present'`）

## テーブル別名の重要性

複数テーブルの結合では、テーブル別名（エイリアス）の使用が特に重要になります。テーブル別名を使うことで：

1. コードが簡潔になる
2. 同じカラム名が複数のテーブルに存在する場合の曖昧さが解消される
3. SQLの可読性が向上する

### 例5：明確なテーブル別名の使用

```
SELECT
    s.student_name AS 学生名,
    c.course_name AS 講座名,
    t.teacher_name AS 教師名,
    cs.schedule_date AS 日付,
    cp.start_time AS 開始時間,
    cl.classroom_name AS 教室
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
JOIN courses c ON sc.course_id = c.course_id
JOIN teachers t ON c.teacher_id = t.teacher_id
JOIN course_schedule cs ON c.course_id = cs.course_id
JOIN class_periods cp ON cs.period_id = cp.period_id
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
WHERE s.student_id = 301
```

```
  AND cs.schedule_date >= '2025-05-01'
ORDER BY cs.schedule_date, cp.start_time;
```

このクエリでは、7つのテーブルに明確な別名を付けています：

- students → s
- student_courses → sc
- courses → c
- teachers → t
- course_schedule → cs
- class_periods → cp
- classrooms → cl

## 複雑な多テーブル結合の実用例

実際の業務でよく使われる、より複雑な多テーブル結合の例を見てみましょう。

### 例6：学生の成績と出席状況の総合分析

```
SELECT
    s.student_id,
    s.student_name AS 学生名,
    c.course_id,
    c.course_name AS 講座名,
    t.teacher_name AS 担当教師,
    -- 中間テストの点数
    MAX(CASE WHEN g.grade_type = '中間テスト' THEN g.score ELSE NULL END) AS 中間テス
ト,
    -- レポートの点数
    MAX(CASE WHEN g.grade_type = 'レポート1' THEN g.score ELSE NULL END) AS レポート,
    -- 出席情報の集計
    COUNT(DISTINCT cs.schedule_id) AS 授業回数,
    SUM(CASE WHEN a.status = 'present' THEN 1 ELSE 0 END) AS 出席回数,
    SUM(CASE WHEN a.status = 'late' THEN 1 ELSE 0 END) AS 遅刻回数,
    SUM(CASE WHEN a.status = 'absent' THEN 1 ELSE 0 END) AS 欠席回数,
    -- 出席率の計算
    ROUND(
        SUM(CASE
            WHEN a.status = 'present' THEN 1
            WHEN a.status = 'late' THEN 0.5
            ELSE 0
        END) / COUNT(DISTINCT cs.schedule_id) * 100,
    1) AS 出席率
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
JOIN courses c ON sc.course_id = c.course_id
JOIN teachers t ON c.teacher_id = t.teacher_id
LEFT JOIN grades g ON s.student_id = g.student_id AND c.course_id = g.course_id
LEFT JOIN course_schedule cs ON c.course_id = cs.course_id
LEFT JOIN attendance a ON cs.schedule_id = a.schedule_id AND s.student_id =
```

```
    a.student_id
WHERE s.student_id BETWEEN 301 AND 305
GROUP BY s.student_id, s.student_name, c.course_id, c.course_name, t.teacher_name
ORDER BY s.student_id, c.course_id;
```

このクエリは、複数のテーブルを結合して各学生の成績と出席状況を総合的に分析しています。特徴として：

- 複数テーブルの結合
- CASE式を使った条件付き集計
- GROUP BY句による集計
- 計算式を使った派生列（出席率）

## 練習問題

### 問題17-1

学生（students）、受講（student_courses）、講座（courses）の3つのテーブルを結合して、学生ID=302の学生が受講しているすべての講座名とその講座IDを取得するSQLを書いてください。

### 問題17-2

講座（courses）、授業カレンダー（course_schedule）、教室（classrooms）、授業時間（class_periods）の4つのテーブルを結合して、2025年5月22日のすべての授業スケジュールを、開始時間順に取得するSQLを書いてください。結果には講座名、教室名、開始時間、終了時間、担当教師名を含めてください。

### 問題17-3

学生（students）テーブル、成績（grades）テーブル、講座（courses）テーブルを結合して、「ITのための基礎知識」（course_id=1）の講座を受講している学生の中間テスト成績を、点数の高い順に取得するSQLを書いてください。結果には学生名、得点、および講座名を含めてください。

### 問題17-4

教師（teachers）テーブル、講座（courses）テーブル、授業カレンダー（course_schedule）テーブル、教室（classrooms）テーブルを結合して、教師ID=106（星野涼子）が2025年5月に担当するすべての授業の詳細を取得するSQLを書いてください。結果には授業日、講座名、教室名を含めてください。

### 問題17-5

学生（students）テーブル、受講（student_courses）テーブル、講座（courses）テーブル、成績（grades）テーブルを結合して、各学生の受講講座数と成績の平均点を計算するSQLを書いてください。成績がない場合も学生と講座は表示してください。結果を平均点の高い順に並べてください。

### 問題17-6

授業カレンダー（course_schedule）テーブル、講座（courses）テーブル、教師（teachers）テーブル、教師スケジュール管理（teacher_unavailability）テーブルを結合して、2025年5月20日以降に予定されている授業のうち、担当教師が不在期間と重なっている授業を特定するSQLを書いてください。結果には授業日、講座名、教師名、不在理由を含めてください。

# 解答

## 解答17-1

```
SELECT
    s.student_id,
    s.student_name AS 学生名,
    c.course_id,
    c.course_name AS 講座名
FROM students s
JOIN student_courses sc ON s.student_id = sc.student_id
JOIN courses c ON sc.course_id = c.course_id
WHERE s.student_id = 302
ORDER BY c.course_id;
```

## 解答17-2

```
SELECT
    c.course_name AS 講座名,
    cl.classroom_name AS 教室名,
    cp.start_time AS 開始時間,
    cp.end_time AS 終了時間,
    t.teacher_name AS 担当教師名
FROM course_schedule cs
JOIN courses c ON cs.course_id = c.course_id
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
JOIN class_periods cp ON cs.period_id = cp.period_id
JOIN teachers t ON cs.teacher_id = t.teacher_id
WHERE cs.schedule_date = '2025-05-22'
ORDER BY cp.start_time, cl.classroom_id;
```

## 解答17-3

```
SELECT
    s.student_name AS 学生名,
    g.score AS 得点,
    c.course_name AS 講座名
FROM students s
JOIN grades g ON s.student_id = g.student_id
JOIN courses c ON g.course_id = c.course_id
WHERE c.course_id = '1'
  AND g.grade_type = '中間テスト'
ORDER BY g.score DESC;
```

## 解答17-4

```
SELECT
    cs.schedule_date AS 授業日,
    c.course_name AS 講座名,
    cl.classroom_name AS 教室名,
    cp.start_time AS 開始時間,
    cp.end_time AS 終了時間
FROM teachers t
JOIN course_schedule cs ON t.teacher_id = cs.teacher_id
JOIN courses c ON cs.course_id = c.course_id
JOIN classrooms cl ON cs.classroom_id = cl.classroom_id
JOIN class_periods cp ON cs.period_id = cp.period_id
WHERE t.teacher_id = 106
  AND cs.schedule_date BETWEEN '2025-05-01' AND '2025-05-31'
ORDER BY cs.schedule_date, cp.start_time;
```

### 解答17-5

```
SELECT
    s.student_id,
    s.student_name AS 学生名,
    COUNT(DISTINCT sc.course_id) AS 受講講座数,
    AVG(g.score) AS 平均点
FROM students s
LEFT JOIN student_courses sc ON s.student_id = sc.student_id
LEFT JOIN courses c ON sc.course_id = c.course_id
LEFT JOIN grades g ON s.student_id = g.student_id AND c.course_id = g.course_id
GROUP BY s.student_id, s.student_name
ORDER BY 平均点 DESC NULLS LAST;
```

注：NULLS LASTはデータベースによってはサポートされていない場合があります。その場合は以下のように書き換えます：

```
ORDER BY CASE WHEN AVG(g.score) IS NULL THEN 0 ELSE 1 END DESC, AVG(g.score) DESC
```

### 解答17-6

```
SELECT
    cs.schedule_date AS 授業日,
    c.course_name AS 講座名,
    t.teacher_name AS 教師名,
    tu.reason AS 不在理由
FROM course_schedule cs
JOIN courses c ON cs.course_id = c.course_id
JOIN teachers t ON cs.teacher_id = t.teacher_id
JOIN teacher_unavailability tu ON cs.teacher_id = tu.teacher_id
WHERE cs.schedule_date >= '2025-05-20'
```

```
    AND cs.schedule_date BETWEEN tu.start_date AND tu.end_date
ORDER BY cs.schedule_date, cs.period_id;
```

## まとめ

この章では、3つ以上のテーブルを結合して複雑な情報を取得する方法について学びました：

1. **複数テーブル結合の基本構文**：JOINをチェーンさせて3つ以上のテーブルを連結する方法
2. **さまざまな結合タイプの組み合わせ**：INNER JOIN、LEFT JOIN、RIGHT JOINを状況に応じて組み合わせる方法
3. **結合順序とパフォーマンスの考慮点**：効率的なクエリを作成するためのヒント
4. **テーブル別名の重要性**：複数テーブル結合での可読性と明確さの向上
5. **複雑な結合の実用例**：実際の業務で使われるような多テーブル結合の例

複数テーブルの結合は、関連するデータを効率的に取得し、有意義な情報を抽出するための重要なスキルです。特に正規化されたデータベースでは、必要な情報を得るためには複数のテーブルを結合することが必須となります。

次の章では、「サブクエリ：WHERE句内のサブクエリ」について学び、クエリの中に別のクエリを埋め込む高度なテクニックを習得していきます。

---

# SQL学習テキスト 完全版

---

このテキストは、SQLの基礎から応用まで体系的に学習できるように構成されています。 学校データベースを使った実践的な例題と練習問題を通じて、実務で使えるSQLスキルを身につけることができます。

---

## 目次



---

# 1. SELECT基本：単一テーブルから特定カラムを取得する

---

## はじめに

データベースからデータを取り出す作業は、料理人が大きな冷蔵庫から必要な材料だけを取り出すようなものです。SQLでは、この「取り出す」作業を「SELECT文」で行います。

SELECT文はSQLの中で最も基本的で、最もよく使われる命令です。この章では、単一のテーブル（データの表）から必要な情報だけを取り出す方法を学びます。

## 基本構文

SELECT文の最も基本的な形は次のとおりです：

```
SELECT カラム名 FROM テーブル名;
```

この文は「テーブル名というテーブルからカラム名という列のデータを取り出してください」という意味です。

> **用語解説**：
>
> - **SELECT**：「選択する」という意味のSQLコマンドで、データを取り出すときに使います。
> - **カラム**：テーブルの縦の列のことで、同じ種類のデータが並んでいます（例：名前のカラム、年齢のカラムなど）。
> - **FROM**：「～から」という意味で、どのテーブルからデータを取るかを指定します。
> - **テーブル**：データベース内の表のことで、行と列で構成されています。

## 実践例：単一カラムの取得

学校データベースの中の「teachers」テーブル（教師テーブル）から、教師の名前だけを取得してみましょう。

```
SELECT teacher_name FROM teachers;
```

実行結果：

| teacher_name |
| --- |
| 寺内鞍 |
| 田尻朋美 |
| 内村海凪 |
| 藤本理恵 |
| 黒木大介 |
| 星野涼子 |
| 深山誠一 |
| 吉岡由佳 |

| teacher_name |
| --- |
| 山田太郎 |
| 佐藤花子 |
| … |

これは「teachers」テーブルの「teacher_name」という列（先生の名前）だけを取り出しています。

## 複数のカラムを取得する

料理に複数の材料が必要なように、データを取り出すときも複数の列が必要なことがよくあります。複数のカラムを取得するには、カラム名をカンマ（,）で区切って指定します。

```
SELECT カラム名1, カラム名2, カラム名3 FROM テーブル名;
```

例えば、教師の番号（ID）と名前を一緒に取得してみましょう：

```
SELECT teacher_id, teacher_name FROM teachers;
```

実行結果：

| teacher_id | teacher_name |
| --- | --- |
| 101 | 寺内鞍 |
| 102 | 田尻朋美 |
| 103 | 内村海凪 |
| 104 | 藤本理恵 |
| 105 | 黒木大介 |
| 106 | 星野涼子 |
| … | … |

## すべてのカラムを取得する

テーブルのすべての列を取得したい場合は、アスタリスク（*）を使います。これは「すべての列」を意味するワイルドカードです。

```
SELECT * FROM テーブル名;
```

例：

```
SELECT * FROM teachers;
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 101 | 寺内鞍 |
| 102 | 田尻朋美 |
| 103 | 内村海凪 |
| … | … |

> **注意**：SELECT *は便利ですが、実際の業務では必要なカラムだけを指定する方が良いとされています。これは、データ量が多いときに処理速度が遅くなるのを防ぐためです。

## カラムに別名をつける（AS句）

取得したカラムに分かりやすい名前（別名）をつけることができます。これは「AS」句を使います。

```
SELECT カラム名 AS 別名 FROM テーブル名;
```

> **用語解説**：
>
> - **AS**：「～として」という意味で、カラムに別名をつけるときに使います。この別名は結果を表示するときだけ使われます。

例えば、教師IDを「番号」、教師名を「名前」として表示してみましょう：

```
SELECT teacher_id AS 番号, teacher_name AS 名前 FROM teachers;
```

実行結果：

| 番号 | 名前 |
|---|---|
| 101 | 寺内鞍 |
| 102 | 田尻朋美 |
| 103 | 内村海凪 |
| … | … |

ASは省略することも可能です：

```
SELECT teacher_id 番号, teacher_name 名前 FROM teachers;
```

## 計算式を使う

SELECT文では、カラムの値を使った計算もできます。例えば、成績テーブルから点数と満点を取得して、達成率（パーセント）を計算してみましょう。

```
SELECT student_id, course_id, grade_type,
       score, max_score,
       (score / max_score) * 100 AS 達成率
FROM grades;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | 達成率 |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 85.5 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 92.0 |
| … | … | … | … | … | … |

## 重複を除外する（DISTINCT）

同じ値が複数ある場合に、重複を除いて一意の値だけを表示するには「DISTINCT」キーワードを使います。

> **用語解説**：
>
> - **DISTINCT**：「異なる」「区別された」という意味で、重複する値を除外して一意の値だけを取得します。

例えば、どの講座にどの教師が担当しているかを重複なしで確認してみましょう：

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

実行結果：

| teacher_id |
|---|
| 101 |
| 102 |
| 103 |
| 104 |
| … |

これにより、courses（講座）テーブルで使われている教師IDが重複なく表示されます。

## 文字列の結合

文字列を結合するには、MySQLでは「CONCAT」関数を使います。例えば、教師のIDと名前を組み合わせて表示してみましょう：

```
SELECT CONCAT('教師ID:', teacher_id, ' 名前:', teacher_name) AS 教師情報
FROM teachers;
```

実行結果：

| 教師情報 |
| --- |
| 教師ID:101 名前:寺内鞍 |
| 教師ID:102 名前:田尻朋美 |

...

> **用語解説**：
>
> - **CONCAT**：複数の文字列を一つにつなげる関数です。

## SELECT文と終了記号

SQLの文は通常、セミコロン（;）で終わります。これは「この命令はここで終わりです」という合図です。

複数のSQL文を一度に実行する場合は、それぞれの文の最後にセミコロンをつけます。

## 練習問題

### 問題1-1

students（学生）テーブルから、すべての学生の名前（student_name）を取得するSQLを書いてください。

### 問題1-2

classrooms（教室）テーブルから、教室ID（classroom_id）と教室名（classroom_name）を取得するSQLを書いてください。

### 問題1-3

courses（講座）テーブルから、すべての列（カラム）を取得するSQLを書いてください。

### 問題1-4

class_periods（授業時間）テーブルから、時限ID（period_id）、開始時間（start_time）、終了時間（end_time）を取得し、開始時間には「開始」、終了時間には「終了」という別名をつけるSQLを書いてください。

### 問題1-5

grades（成績）テーブルから、学生ID（student_id）、講座ID（course_id）、評価タイプ（grade_type）、得点（score）、満点（max_score）、そして得点を満点で割って100を掛けた値を「パーセント」という別名で取得するSQLを書いてください。

### 問題1-6

course_schedule（授業カレンダー）テーブルから、schedule_date（予定日）カラムだけを重複なしで取得するSQLを書いてください。

## 解答

### 解答1-1

```
SELECT student_name FROM students;
```

### 解答1-2

```
SELECT classroom_id, classroom_name FROM classrooms;
```

### 解答1-3

```
SELECT * FROM courses;
```

### 解答1-4

```
SELECT period_id, start_time AS 開始, end_time AS 終了 FROM class_periods;
```

### 解答1-5

```
SELECT student_id, course_id, grade_type, score, max_score,
       (score / max_score) * 100 AS パーセント
FROM grades;
```

### 解答1-6

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

## まとめ

この章では、SQLのSELECT文の基本を学びました：

1. 単一カラムの取得: `SELECT カラム名 FROM テーブル名;`
2. 複数カラムの取得: `SELECT カラム名1, カラム名2 FROM テーブル名;`
3. すべてのカラムの取得: `SELECT * FROM テーブル名;`
4. カラムに別名をつける: `SELECT カラム名 AS 別名 FROM テーブル名;`
5. 計算式を使う: `SELECT カラム名, (計算式) AS 別名 FROM テーブル名;`
6. 重複を除外する: `SELECT DISTINCT カラム名 FROM テーブル名;`
7. 文字列の結合: `SELECT CONCAT(文字列1, カラム名, 文字列2) FROM テーブル名;`

これらの基本操作を使いこなせるようになれば、データベースから必要な情報を効率よく取り出せるようになります。次の章では、WHERE句を使って条件に合ったデータだけを取り出す方法を学びます。

---

# 2. WHERE句：条件に合ったレコードを絞り込む

## はじめに

前章では、テーブルからデータを取得する基本的な方法を学びました。しかし実際の業務では、すべてのデータではなく、特定の条件に合ったデータだけを取得したいことがほとんどです。

例えば、「全生徒の情報」ではなく「特定の学科の生徒だけ」や「成績が80点以上の学生だけ」といった形で、データを絞り込みたい場合があります。

このような場合に使用するのが「WHERE句」です。WHERE句は、SELECTコマンドの後に追加して使い、条件に合致するレコード（行）だけを取得します。

## 基本構文

WHERE句の基本的な形は次のとおりです：

```
SELECT カラム名 FROM テーブル名 WHERE 条件式;
```

> **用語解説**：
>
> - **WHERE**：「～の場所で」「～の条件で」という意味のSQLコマンドで、条件に合うデータだけを抽出するために使います。
> - **条件式**：データが満たすべき条件を指定するための式です。例えば「age > 20」（年齢が20より大きい）などです。
> - **レコード**：テーブルの横の行のことで、1つのデータの集まりを表します。

## 基本的な比較演算子

WHERE句では、様々な比較演算子を使って条件を指定できます：

| 演算子 | 意味 | 例 |
|---|---|---|
| = | 等しい | age = 25 |

| 演算子 | 意味 | 例 |
|---|---|---|
| <> | 等しくない（≠と同じ） | gender <> 'male' |
| > | より大きい | score > 80 |
| < | より小さい | price < 1000 |
| >= | 以上 | height >= 170 |
| <= | 以下 | weight <= 70 |

## 実践例：基本的な条件での絞り込み

### 例1：等しい（=）

例えば、教師ID（teacher_id）が101の教師のみを取得するには：

```
SELECT * FROM teachers WHERE teacher_id = 101;
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 101 | 寺内鞍 |

### 例2：より大きい（>）

成績（grades）テーブルから、90点を超える成績だけを取得するには：

```
SELECT * FROM grades WHERE score > 90;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

### 例3：等しくない（<>）

講座IDが3ではない講座に関する成績を取得するには：

```
SELECT * FROM grades WHERE course_id <> '3';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 301 | 2 | 実技試験 | 88.0 | 100.0 | 2025-05-18 |
| ... | ... | ... | ... | ... | ... |

## 文字列の比較

テキスト（文字列）を条件にする場合は、シングルクォーテーション（'）またはダブルクォーテーション（"）で囲みます。MySQLではどちらも使えますが、多くの場合シングルクォーテーションが推奨されます。

```
SELECT * FROM テーブル名 WHERE テキストカラム = 'テキスト値';
```

例えば、教師名（teacher_name）が「田尻朋美」の教師を検索するには：

```
SELECT * FROM teachers WHERE teacher_name = '田尻朋美';
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 102 | 田尻朋美 |

## 日付の比較

日付の比較も同様にシングルクォーテーションで囲みます。日付の形式はデータベースの設定によって異なりますが、一般的にはISO形式（YYYY-MM-DD）が使われます。

```
SELECT * FROM テーブル名 WHERE 日付カラム = '日付';
```

例えば、2025年5月20日に提出された成績を検索するには：

```
SELECT * FROM grades WHERE submission_date = '2025-05-20';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 303 | 1 | 中間テスト | 78.5 | 100.0 | 2025-05-20 |
| ... | ... | ... | ... | ... | ... |

また、日付同士の大小関係も比較できます：

```
SELECT * FROM grades WHERE submission_date < '2025-05-15';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | レポート1 | 45.0 | 50.0 | 2025-05-10 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| 304 | 5 | ER図作成課題 | 27.5 | 30.0 | 2025-05-14 |
| ... | ... | ... | ... | ... | ... |

## 複数の条件を指定する（次章の内容）

WHERE句では、複数の条件を組み合わせることもできます。この詳細は次章「論理演算子：AND、OR、NOTを使った複合条件」で説明します。

例えば、講座IDが1で、かつ、得点が90以上の成績を取得するには：

```
SELECT * FROM grades WHERE course_id = '1' AND score >= 90;
```

この例の詳細な説明は次章で行います。

## 練習問題

### 問題2-1

students（学生）テーブルから、学生ID（student_id）が「310」の学生情報を取得するSQLを書いてください。

### 問題2-2

classrooms（教室）テーブルから、収容人数（capacity）が30人より多い教室の情報をすべて取得するSQLを書いてください。

### 問題2-3

courses（講座）テーブルから、教師ID（teacher_id）が「105」の講座情報を取得するSQLを書いてください。

### 問題2-4

course_schedule（授業カレンダー）テーブルから、「2025-05-15」の授業スケジュールをすべて取得するSQLを書いてください。

### 問題2-5

grades（成績）テーブルから、評価タイプ（grade_type）が「中間テスト」で、得点（score）が80点未満の成績を取得するSQLを書いてください。

### 問題2-6

teachers（教師）テーブルから、教師ID（teacher_id）が「101」ではない教師の名前を取得するSQLを書いてください。

## 解答

### 解答2-1

```sql
SELECT * FROM students WHERE student_id = 310;
```

### 解答2-2

```sql
SELECT * FROM classrooms WHERE capacity > 30;
```

### 解答2-3

```sql
SELECT * FROM courses WHERE teacher_id = 105;
```

### 解答2-4

```sql
SELECT * FROM course_schedule WHERE schedule_date = '2025-05-15';
```

### 解答2-5

```sql
SELECT * FROM grades WHERE grade_type = '中間テスト' AND score < 80;
```

### 解答2-6

```
SELECT teacher_name FROM teachers WHERE teacher_id <> 101;
```

## まとめ

この章では、WHERE句を使って条件に合ったレコードを取得する方法を学びました：

1. 基本的な比較演算子（=, <>, >, <, >=, <=）の使い方
2. 数値による条件絞り込み
3. 文字列（テキスト）による条件絞り込み
4. 日付による条件絞り込み

WHERE句は、大量のデータから必要な情報だけを取り出すための非常に重要な機能です。実際のデータベース操作では、この条件絞り込みを頻繁に使います。

次の章では、複数の条件を組み合わせるための「論理演算子（AND、OR、NOT）」について学びます。

---

# 3. 論理演算子：AND、OR、NOTを使った複合条件

## はじめに

前章では、WHERE句を使って単一の条件でデータを絞り込む方法を学びました。しかし実際の業務では、より複雑な条件でデータを絞り込む必要があることがよくあります。

例えば：

- 「成績が80点以上**かつ**出席率が90%以上の学生」
- 「数学**または**英語の成績が優秀な学生」
- 「課題を**まだ提出していない**学生」

このような複合条件を指定するために使うのが論理演算子です。主な論理演算子は次の3つです：

- **AND**：両方の条件を満たす（かつ）
- **OR**：いずれかの条件を満たす（または）
- **NOT**：条件を満たさない（～ではない）

## AND演算子

AND演算子は、指定した**すべての条件を満たす**レコードだけを取得したいときに使います。

> **用語解説**：
>
> - **AND**：「かつ」「そして」という意味の論理演算子です。複数の条件をすべて満たすデータを取得します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE 条件1 AND 条件2;
```

### 例：ANDを使った複合条件

例えば、中間テストで90点以上かつ満点が100点の成績レコードを取得するには：

```
SELECT * FROM grades
WHERE score >= 90 AND max_score = 100;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

この例では、「score >= 90」と「max_score = 100」の両方の条件を満たすレコードだけが取得されます。

### 3つ以上の条件の組み合わせ

ANDを使って3つ以上の条件を組み合わせることもできます：

```
SELECT * FROM grades
WHERE score >= 90 AND max_score = 100 AND grade_type = '中間テスト';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

## OR演算子

OR演算子は、指定した条件の**いずれか一つでも満たす**レコードを取得したいときに使います。

> **用語解説**：
>
> - **OR**：「または」「もしくは」という意味の論理演算子です。複数の条件のうち少なくとも1つを満たすデータを取得します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE 条件1 OR 条件2;
```

### 例：ORを使った複合条件

例えば、教師IDが101または102の講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id = 101 OR teacher_id = 102;
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 1 | ITのための基礎知識 | 101 |
| 2 | UNIX入門 | 102 |
| 3 | Cプログラミング演習 | 101 |
| 29 | コードリファクタリングとクリーンコード | 101 |
| 40 | ソフトウェアアーキテクチャパターン | 102 |
| … | … | … |

この例では、「teacher_id = 101」または「teacher_id = 102」のいずれかの条件を満たすレコードが取得されます。

## NOT演算子

NOT演算子は、指定した条件を**満たさない**レコードを取得したいときに使います。

> **用語解説**：
>
> - **NOT**：「～ではない」という意味の論理演算子です。条件を否定して、その条件を満たさないデータを取得します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE NOT 条件;
```

### 例：NOTを使った否定条件

例えば、完了（completed）状態ではない授業スケジュールを取得するには：

```
SELECT * FROM course_schedule
WHERE NOT status = 'completed';
```

実行結果：

| schedule_id | course_id | schedule_date | period_id | classroom_id | teacher_id | status |
|---|---|---|---|---|---|---|
| 45 | 11 | 2025-05-20 | 5 | 401G | 106 | scheduled |
| 46 | 12 | 2025-05-21 | 1 | 301E | 107 | scheduled |
| 50 | 2 | 2025-05-23 | 3 | 101A | 102 | cancelled |
| … | … | … | … | … | … | … |

この例では、statusが「completed」ではないレコード（scheduled状態やcancelled状態）が取得されます。

NOT演算子は、「～ではない」という否定の条件を作るために使われます。例えば次の2つの書き方は同じ意味になります：

```
SELECT * FROM teachers WHERE NOT teacher_id = 101;
SELECT * FROM teachers WHERE teacher_id <> 101;
```

## 複合論理条件（AND、OR、NOTの組み合わせ）

AND、OR、NOTを組み合わせて、より複雑な条件を指定することもできます。

### 例：ANDとORの組み合わせ

例えば、「成績が90点以上で中間テストである」または「成績が45点以上でレポートである」レコードを取得するには：

```
SELECT * FROM grades
WHERE (score >= 90 AND grade_type = '中間テスト')
   OR (score >= 45 AND grade_type = 'レポート1');
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| 308 | 1 | レポート1 | 47.0 | 50.0 | 2025-05-09 |

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |

## 優先順位と括弧の使用

論理演算子を組み合わせる場合、演算子の優先順位に注意が必要です。基本的に**ANDはORよりも優先順位が高い**です。つまり、ANDが先に処理されます。

例えば：

```
WHERE 条件1 OR 条件2 AND 条件3
```

これは次のように解釈されます：

```
WHERE 条件1 OR (条件2 AND 条件3)
```

意図した条件と異なる場合は、**括弧（）を使って明示的にグループ化**することが重要です：

```
WHERE (条件1 OR 条件2) AND 条件3
```

### 例：括弧を使った条件のグループ化

教師IDが101または102で、かつ、講座名に「プログラミング」という単語が含まれる講座を取得するには：

```
SELECT * FROM courses
WHERE (teacher_id = 101 OR teacher_id = 102)
  AND course_name LIKE '%プログラミング%';
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 3 | Cプログラミング演習 | 101 |
| ... | ... | ... |

この例では、括弧を使って「teacher_id = 101 OR teacher_id = 102」の部分をグループ化し、その条件と「course_name LIKE '%プログラミング%'」の条件をANDで結合しています。

## 練習問題

問題3-1

grades（成績）テーブルから、課題タイプ（grade_type）が「中間テスト」かつ点数（score）が85点以上のレコードを取得するSQLを書いてください。

### 問題3-2

classrooms（教室）テーブルから、収容人数（capacity）が40人以下または建物（building）が「1号館」の教室を取得するSQLを書いてください。

### 問題3-3

teachers（教師）テーブルから、教師ID（teacher_id）が101、102、103ではない教師の情報を取得するSQLを書いてください。

### 問題3-4

course_schedule（授業カレンダー）テーブルから、2025年5月20日の授業で、時限（period_id）が1か2で、かつ状態（status）が「scheduled」の授業を取得するSQLを書いてください。

### 問題3-5

students（学生）テーブルから、学生名（student_name）に「田」または「山」を含む学生を取得するSQLを書いてください。

### 問題3-6

grades（成績）テーブルから、提出日（submission_date）が2025年5月15日以降で、かつ（「中間テスト」で90点以上または「レポート1」で45点以上）の成績を取得するSQLを書いてください。

## 解答

### 解答3-1

```sql
SELECT * FROM grades
WHERE grade_type = '中間テスト' AND score >= 85;
```

### 解答3-2

```sql
SELECT * FROM classrooms
WHERE capacity <= 40 OR building = '1号館';
```

### 解答3-3

```sql
SELECT * FROM teachers
WHERE NOT (teacher_id = 101 OR teacher_id = 102 OR teacher_id = 103);
```

または

```
SELECT * FROM teachers
WHERE teacher_id <> 101 AND teacher_id <> 102 AND teacher_id <> 103;
```

解答3-4

```
SELECT * FROM course_schedule
WHERE schedule_date = '2025-05-20'
  AND (period_id = 1 OR period_id = 2)
  AND status = 'scheduled';
```

解答3-5

```
SELECT * FROM students
WHERE student_name LIKE '%田%' OR student_name LIKE '%山%';
```

解答3-6

```
SELECT * FROM grades
WHERE submission_date >= '2025-05-15'
  AND ((grade_type = '中間テスト' AND score >= 90)
       OR (grade_type = 'レポート1' AND score >= 45));
```

# まとめ

この章では、論理演算子（AND、OR、NOT）を使って複合条件を作る方法を学びました：

1. **AND**：すべての条件を満たすレコードを取得（条件1 AND 条件2）
2. **OR**：いずれかの条件を満たすレコードを取得（条件1 OR 条件2）
3. **NOT**：指定した条件を満たさないレコードを取得（NOT 条件）
4. **複合条件**：AND、OR、NOTを組み合わせたより複雑な条件
5. **括弧（）**：条件をグループ化して優先順位を明示的に指定

これらの論理演算子を使いこなすことで、より複雑で細かな条件でデータを絞り込むことができるようになります。実際のデータベース操作では、複数の条件を組み合わせることが頻繁にあるため、この章で学んだ内容は非常に重要です。

次の章では、テキストデータに対する検索を行う「パターンマッチング」について学びます。

---

# 4. パターンマッチング：LIKE演算子と%、_ワイルドカード

## はじめに

前章までは、データの完全一致や数値の比較といった条件での絞り込みを学びました。しかし実際の業務では、もっと柔軟な検索が必要なケースがあります。例えば：

- 「山」で始まる名前の学生を検索したい
- 「プログラミング」という単語を含む講座名を探したい
- 電話番号の一部だけ覚えているデータを探したい

このような「部分一致」や「パターン一致」の検索を行うためのSQLの機能が「パターンマッチング」です。この章では、パターンマッチングを行うための「LIKE演算子」と「ワイルドカード文字」について学びます。

## LIKE演算子の基本

LIKE演算子は、文字列のパターンマッチングを行うための演算子です。WHERE句と組み合わせて使用します。

> **用語解説**：
>
> - **LIKE**：「～のような」という意味の演算子で、パターンに一致する文字列を検索します。
> - **パターンマッチング**：完全一致ではなく、一定のパターンに合致するデータを検索する方法です。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE 文字列カラム LIKE 'パターン';
```

パターンには、通常の文字に加えて、特別な意味を持つ「ワイルドカード文字」を使用できます。

## ワイルドカード文字

SQLでは主に2つのワイルドカード文字があります：

1. **%（パーセント）**：0文字以上の任意の文字列に一致します。
2. _**（アンダースコア）**：任意の1文字に一致します。

> **用語解説**：
>
> - **ワイルドカード**：任意の文字や文字列に一致する特殊な文字記号です。トランプのジョーカーのように、様々な値に代用できます。

## LIKE演算子の使い方：実践例

### 例1：%（パーセント）を使ったパターンマッチング

**「〜で始まる」パターン：前方一致**

例えば、「山」で始まる学生名を検索するには：

```
SELECT * FROM students WHERE student_name LIKE '山%';
```

実行結果：

| student_id | student_name |
|---|---|
| 312 | 山本裕子 |
| 325 | 山田翔太 |
| … | … |

ここでの「山%」は「山で始まり、その後に0文字以上の任意の文字が続く」という意味です。

**「〜で終わる」パターン：後方一致**

例えば、「子」で終わる教師名を検索するには：

```
SELECT * FROM teachers WHERE teacher_name LIKE '%子';
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 102 | 田尻朋美 |
| 106 | 星野涼子 |
| 108 | 吉岡由佳 |
| 110 | 佐藤花子 |
| … | … |

**「〜を含む」パターン：部分一致**

例えば、「プログラミング」という単語を含む講座名を検索するには：

```
SELECT * FROM courses WHERE course_name LIKE '%プログラミング%';
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|

| course_id | course_name | teacher_id |
|---|---|---|
| 3 | Cプログラミング演習 | 101 |
| 14 | IoTデバイスプログラミング実践 | 110 |
| … | … | … |

### 例2：_（アンダースコア）を使ったパターンマッチング

アンダースコアは任意の1文字に一致します。例えば、教室IDが「10_A」パターン（最初の2文字が「10」、3文字目が任意の1文字、最後が「A」）の教室を検索するには：

```
SELECT * FROM classrooms WHERE classroom_id LIKE '10_A';
```

実行結果：

| classroom_id | classroom_name | capacity | building | facilities |
|---|---|---|---|---|
| 101A | 1号館コンピュータ実習室A | 30 | 1号館 | パソコン30台、プロジェクター |
| … | … | … | … | … |

### 例3：%と_の組み合わせ

ワイルドカード文字は組み合わせて使うこともできます。例えば、「2文字目が田」の学生を検索するには：

```
SELECT * FROM students WHERE student_name LIKE '_田%';
```

実行結果：

| student_id | student_name |
|---|---|
| 321 | 井上竜也 |
| 384 | 櫻井翼 |
| … | … |

## NOT LIKEを使った否定形のパターンマッチング

特定のパターンに一致しないレコードを検索したい場合は、「NOT LIKE」を使います。

> **用語解説**：
>
> - **NOT LIKE**：「～のパターンに一致しない」という意味で、指定したパターンに一致しないデータを検索します。

例えば、講座名に「入門」を含まない講座を検索するには：

```
SELECT * FROM courses WHERE course_name NOT LIKE '%入門%';
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 1 | ITのための基礎知識 | 101 |
| 3 | Cプログラミング演習 | 101 |
| ... | ... | ... |

## エスケープ文字の使用

もし検索したいパターンに「%」や「_」自体が含まれている場合は、それらを特別な文字としてではなく、通常の文字として扱うために「エスケープ文字」を使います。

> **用語解説**：
>
> - **エスケープ文字**：特別な意味を持つ文字を通常の文字として扱うための印です。

MySQLでは、バックスラッシュ（\）をエスケープ文字として使用できます。例えば、「50%」という値そのものを検索するには：

```
SELECT * FROM テーブル名 WHERE カラム LIKE '50\%';
```

または、ESCAPE句を使って明示的にエスケープ文字を指定することもできます：

```
SELECT * FROM テーブル名 WHERE カラム LIKE '50!%' ESCAPE '!';
```

この例では「!」をエスケープ文字として指定しています。

## 大文字と小文字の区別

MySQLのデフォルト設定では、LIKE演算子は大文字と小文字を区別しません（大文字小文字を同じものとして扱います）。

例えば、次の2つのクエリは同じ結果を返します：

```
SELECT * FROM courses WHERE course_name LIKE '%web%';
SELECT * FROM courses WHERE course_name LIKE '%Web%';
```

もし大文字と小文字を区別した検索が必要な場合は、「BINARY」キーワードを使用します：

```

```

```
SELECT * FROM courses WHERE course_name LIKE BINARY '%Web%';
```

この場合、「Web」は「web」とは一致しません。

## 複合条件との組み合わせ

LIKE演算子は、これまで学んだAND、OR、NOTなどの論理演算子と組み合わせて使うこともできます。

例えば、「田」で始まる名前で、かつ教師IDが102から105の間の教師を検索するには：

```
SELECT * FROM teachers
WHERE teacher_name LIKE '田%'
  AND teacher_id BETWEEN 102 AND 105;
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 102 | 田尻朋美 |
| … | … |

## 練習問題

### 問題4-1

students（学生）テーブルから、学生名（student_name）が「佐藤」で始まる学生の情報をすべて取得するSQLを書いてください。

### 問題4-2

courses（講座）テーブルから、講座名（course_name）に「データ」という単語を含む講座の情報を取得するSQLを書いてください。

### 問題4-3

classrooms（教室）テーブルから、教室名（classroom_name）が「コンピュータ実習室」で終わる教室の情報を取得するSQLを書いてください。

### 問題4-4

teachers（教師）テーブルから、教師名（teacher_name）の2文字目が「木」である教師の情報を取得するSQLを書いてください。

### 問題4-5

courses（講座）テーブルから、講座名（course_name）に「入門」または「基礎」を含む講座を取得するSQLを書いてください。

### 問題4-6

students（学生）テーブルから、学生名（student_name）が「山」で始まり、かつ「子」で終わらない学生を取得するSQLを書いてください。

## 解答

### 解答4-1

```
SELECT * FROM students WHERE student_name LIKE '佐藤%';
```

### 解答4-2

```
SELECT * FROM courses WHERE course_name LIKE '%データ%';
```

### 解答4-3

```
SELECT * FROM classrooms WHERE classroom_name LIKE '%コンピュータ実習室';
```

### 解答4-4

```
SELECT * FROM teachers WHERE teacher_name LIKE '_木%';
```

### 解答4-5

```
SELECT * FROM courses
WHERE course_name LIKE '%入門%' OR course_name LIKE '%基礎%';
```

### 解答4-6

```
SELECT * FROM students
WHERE student_name LIKE '山%' AND student_name NOT LIKE '%子';
```

## まとめ

この章では、パターンマッチングを行うためのLIKE演算子と、その中で使用するワイルドカード文字（%と_）について学びました：

1. **LIKE演算子**：文字列パターンに一致するデータを検索するための演算子

2. **%（パーセント）**：0文字以上の任意の文字列に一致するワイルドカード
3. **_（アンダースコア）**：任意の1文字に一致するワイルドカード
4. **前方一致**：「パターン%」で「パターンで始まる」文字列に一致
5. **後方一致**：「%パターン」で「パターンで終わる」文字列に一致
6. **部分一致**：「%パターン%」で「パターンを含む」文字列に一致
7. **NOT LIKE**：指定したパターンに一致しないデータを検索
8. **エスケープ文字**：特殊文字（%や_）を通常の文字として扱うための方法
9. **複合条件との組み合わせ**：AND、ORなどと組み合わせたより複雑な条件

パターンマッチングは、特にテキストデータを扱う際に非常に便利な機能です。部分的な情報しか持っていない場合や、特定のパターンを持つデータを探す場合に活用できます。

次の章では、範囲指定のための「BETWEEN演算子」と「IN演算子」について学びます。

---

# 5. 範囲指定：BETWEEN、IN演算子

## はじめに

これまでの章では、等号（=）や不等号（>、<）を使って条件を指定する方法や、LIKE演算子を使ったパターンマッチングを学びました。この章では、値の範囲を指定する「BETWEEN演算子」と、複数の値を一度に指定できる「IN演算子」について学びます。

これらの演算子を使うと、次のような検索がより簡単になります：

- 「80点から90点の間の成績」
- 「2025年4月から6月の間のスケジュール」
- 「特定の教師IDリストに該当する講座」

## BETWEEN演算子：範囲を指定する

BETWEEN演算子は、ある値が指定した範囲内にあるかどうかを調べるために使います。

> **用語解説**：
>
> - **BETWEEN**：「～の間に」という意味の演算子で、ある値が指定した最小値と最大値の間（両端の値を含む）にあるかどうかを判定します。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 BETWEEN 最小値 AND 最大値;
```

この構文は次の条件と同じ意味です：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 >= 最小値 AND カラム名 <= 最大値;
```

### 例1：数値範囲の指定

例えば、成績（grades）テーブルから、80点から90点の間の成績を取得するには：

```
SELECT * FROM grades
WHERE score BETWEEN 80 AND 90;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |
| 308 | 6 | 小テスト1 | 89.0 | 100.0 | 2025-05-15 |
| … | … | … | … | … | … |

### 例2：日付範囲の指定

日付にもBETWEEN演算子が使えます。例えば、2025年5月10日から2025年5月20日までに提出された成績を取得するには：

```
SELECT * FROM grades
WHERE submission_date BETWEEN '2025-05-10' AND '2025-05-20';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | レポート1 | 45.0 | 50.0 | 2025-05-10 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

## NOT BETWEEN：範囲外を指定する

NOT BETWEENを使うと、指定した範囲の外にある値を持つレコードを取得できます。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 NOT BETWEEN 最小値 AND 最大値;
```

これは次の条件と同じです：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 < 最小値 OR カラム名 > 最大値;
```

### 例：範囲外の値を取得

例えば、80点未満または90点より高い成績を取得するには：

```
SELECT * FROM grades
WHERE score NOT BETWEEN 80 AND 90;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 303 | 1 | 中間テスト | 78.5 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

## IN演算子：複数の値を指定する

IN演算子は、ある値が指定した複数の値のリストのいずれかに一致するかどうかを調べるために使います。

> **用語解説**：
>
> - **IN**：「～の中に含まれる」という意味の演算子で、ある値が指定したリストの中に含まれているかどうかを判定します。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 IN (値1, 値2, 値3, ...);
```

この構文は次のOR条件の組み合わせと同じ意味です：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 = 値1 OR カラム名 = 値2 OR カラム名 = 値3 OR ...;
```

### 例1：数値リストの指定

例えば、教師ID（teacher_id）が101、103、105のいずれかである講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id IN (101, 103, 105);
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 1 | ITのための基礎知識 | 101 |
| 3 | Cプログラミング演習 | 101 |
| 5 | データベース設計と実装 | 105 |
| 10 | プロジェクト管理手法 | 103 |
| … | … | … |

### 例2：文字列リストの指定

文字列のリストにも適用できます。例えば、特定の教室ID（classroom_id）のみの教室情報を取得するには：

```
SELECT * FROM classrooms
WHERE classroom_id IN ('101A', '202D', '301E');
```

実行結果：

| classroom_id | classroom_name | capacity | building | facilities |
|---|---|---|---|---|
| 101A | 1号館コンピュータ実習室A | 30 | 1号館 | パソコン30台、プロジェクター |
| 202D | 2号館コンピュータ実習室D | 25 | 2号館 | パソコン25台、プロジェクター、3Dプリンター |
| 301E | 3号館講義室E | 80 | 3号館 | プロジェクター、マイク設備、録画設備 |

## NOT IN：リストに含まれない値を指定する

NOT IN演算子を使うと、指定したリストに含まれない値を持つレコードを取得できます。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 NOT IN (値1, 値2, 値3, ...);
```

これは次の条件と同じです：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 <> 値1 AND カラム名 <> 値2 AND カラム名 <> 値3 AND ...;
```

### 例：リストに含まれない値を取得

例えば、教師IDが101、102、103以外の教師が担当する講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id NOT IN (101, 102, 103);
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 4 | Webアプリケーション開発 | 104 |
| 5 | データベース設計と実装 | 105 |
| 6 | ネットワークセキュリティ | 107 |
| ... | ... | ... |

## IN演算子でのサブクエリの利用（基本）

IN演算子の括弧内には、直接値を書く代わりに、サブクエリ（別のSELECT文）を指定することもできます。これにより、動的に値のリストを生成できます。

> **用語解説**：
>
> - **サブクエリ**：SQL文の中に含まれる別のSQL文のことで、外側のSQL文（メインクエリ）に値や条件を提供します。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 IN (SELECT カラム名 FROM 別テーブル WHERE 条件);
```

### 例：サブクエリを使ったIN条件

例えば、教師名（teacher_name）に「田」を含む教師が担当している講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id IN (
```

```
    SELECT teacher_id FROM teachers
    WHERE teacher_name LIKE '%田%'
);
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 2 | UNIX入門 | 102 |
| 10 | プロジェクト管理手法 | 103 |
| … | … | … |

この例では、まず「teachers」テーブルから名前に「田」を含む教師のIDを取得し、それらのIDを持つ講座を「courses」テーブルから取得しています。

## BETWEEN演算子とIN演算子の組み合わせ

BETWEEN演算子とIN演算子は、論理演算子（AND、OR）と組み合わせて、さらに複雑な条件を作ることができます。

### 例：BETWEENとINの組み合わせ

例えば、「教師IDが101、103、105のいずれかで、かつ、2025年5月15日から2025年6月15日の間に実施される授業」を取得するには：

```
SELECT * FROM course_schedule
WHERE teacher_id IN (101, 103, 105)
  AND schedule_date BETWEEN '2025-05-15' AND '2025-06-15';
```

実行結果：

| schedule_id | course_id | schedule_date | period_id | classroom_id | teacher_id | status |
|---|---|---|---|---|---|---|
| 38 | 1 | 2025-05-20 | 1 | 102B | 101 | scheduled |
| 42 | 10 | 2025-05-23 | 3 | 201C | 103 | scheduled |
| 45 | 5 | 2025-05-28 | 2 | 402H | 105 | scheduled |
| … | … | … | … | … | … | … |

## 練習問題

### 問題5-1

grades（成績）テーブルから、得点（score）が85点から95点の間にある成績を取得するSQLを書いてください。

### 問題5-2

course_schedule（授業カレンダー）テーブルから、2025年5月1日から2025年5月31日までの授業スケジュールを取得するSQLを書いてください。

### 問題5-3

courses（講座）テーブルから、教師ID（teacher_id）が104、106、108のいずれかである講座の情報を取得するSQLを書いてください。

### 問題5-4

classrooms（教室）テーブルから、教室ID（classroom_id）が「101A」、「201C」、「301E」、「401G」以外の教室情報を取得するSQLを書いてください。

### 問題5-5

grades（成績）テーブルから、評価タイプ（grade_type）が「中間テスト」または「実技試験」で、かつ得点（score）が80点から90点の間にない成績を取得するSQLを書いてください。

### 問題5-6

course_schedule（授業カレンダー）テーブルから、教室ID（classroom_id）が「101A」、「202D」のいずれかで、かつ2025年5月15日から2025年5月30日の間に実施される授業スケジュールを取得するSQLを書いてください。

## 解答

### 解答5-1

```
SELECT * FROM grades
WHERE score BETWEEN 85 AND 95;
```

### 解答5-2

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

### 解答5-3

```
SELECT * FROM courses
WHERE teacher_id IN (104, 106, 108);
```

### 解答5-4

```
SELECT * FROM classrooms
WHERE classroom_id NOT IN ('101A', '201C', '301E', '401G');
```

解答5-5

```
SELECT * FROM grades
WHERE grade_type IN ('中間テスト', '実技試験')
  AND score NOT BETWEEN 80 AND 90;
```

解答5-6

```
SELECT * FROM course_schedule
WHERE classroom_id IN ('101A', '202D')
  AND schedule_date BETWEEN '2025-05-15' AND '2025-05-30';
```

## まとめ

この章では、範囲指定のための「BETWEEN演算子」と複数値指定のための「IN演算子」について学びました：

1. **BETWEEN演算子**：値が指定した範囲内（両端を含む）にあるかどうかをチェック
2. **NOT BETWEEN**：値が指定した範囲外にあるかどうかをチェック
3. **IN演算子**：値が指定したリストのいずれかに一致するかどうかをチェック
4. **NOT IN**：値が指定したリストのいずれにも一致しないかどうかをチェック
5. **サブクエリとIN**：動的に生成された値のリストを使用する方法
6. **複合条件**：BETWEEN、IN、論理演算子を組み合わせたより複雑な条件

これらの演算子を使うことで、複数の条件を指定する場合に、SQLをより簡潔に書くことができます。特に、多くの値を指定する場合や範囲条件を指定する場合に便利です。

次の章では、「NULL値の処理：IS NULL、IS NOT NULL」について学びます。

---

# 6. NULL値の処理：IS NULL、IS NOT NULL

## はじめに

データベースの世界では、データがない状態を表すために「NULL」という特別な値が使われます。NULLは「空」や「0」や「空白文字」とは異なる、「値が存在しない」または「不明」であることを表す特殊な概念です。

例えば、学校データベースでは次のようなシナリオがあります：

- まだ成績が付けられていない（NULL）

- コメントが入力されていない（NULL）
- 授業がキャンセルされたため教室が割り当てられていない（NULL）

この章では、NULL値を正しく処理するための「IS NULL」と「IS NOT NULL」演算子について学びます。

## NULLとは何か？

NULL値には、いくつかの特徴があります：

1. **値がない**：NULLは値がないことを表します。0でも空文字列（''）でもなく、値そのものが存在しないことを示します。
2. **不明**：データが不明であることを表す場合もあります。
3. **未設定**：まだ値が設定されていないことを表す場合もあります。
4. **比較できない**：NULLは通常の比較演算子（=, <, >など）で比較できません。

> **用語解説**：
>
> - **NULL**：データベースにおいて「値がない」または「不明」を表す特殊な値です。0や空文字とは異なります。

## NULL値と通常の比較演算子

通常の比較演算子（=, <>, >, <, >=, <=）では、NULL値を正しく検出できません。例えば：

```
-- この条件はNULL値に対して常にFALSEを返す
SELECT * FROM テーブル名 WHERE カラム名 = NULL;

-- この条件もNULL値に対して常にFALSEを返す
SELECT * FROM テーブル名 WHERE カラム名 <> NULL;
```

これは、NULL値との等価比較は「不明」と評価されるためです。つまり、NULL = NULLでさえFALSEではなく「不明」になります。

## IS NULL演算子

NULL値を持つレコードを検索するには、「IS NULL」演算子を使います。

> **用語解説**：
>
> - **IS NULL**：カラムの値がNULLかどうかを調べる演算子です。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE カラム名 IS NULL;
```

### 例：IS NULLの使用

例えば、コメントが入力されていない（NULL）出席レコードを検索するには：

```
SELECT * FROM attendance WHERE comment IS NULL;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 301 | present | NULL |
| 1 | 306 | present | NULL |
| 1 | 307 | present | NULL |
| ... | ... | ... | NULL |

## IS NOT NULL演算子

逆に、NULL値を持たないレコード（つまり、何らかの値を持つレコード）を検索するには、「IS NOT NULL」演算子を使います。

> **用語解説**：
>
> - **IS NOT NULL**：カラムの値がNULLでないかどうかを調べる演算子です。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE カラム名 IS NOT NULL;
```

### 例：IS NOT NULLの使用

例えば、コメントが入力されている（NOT NULL）出席レコードを検索するには：

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

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 302 | late | 15分遅刻 |
| 1 | 303 | absent | 事前連絡あり |
| 1 | 308 | late | 5分遅刻 |
| ... | ... | ... | ... |

## NULL値の論理的な扱い

NULL値は論理演算（AND、OR、NOT）でも特殊な扱いを受けます。

- **NULL AND TRUE** → NULL（不明）
- **NULL AND FALSE** → FALSE
- **NULL OR TRUE** → TRUE
- **NULL OR FALSE** → NULL（不明）
- **NOT NULL** → NULL（不明）

この特殊な振る舞いが、バグや誤った結果の原因になることがあります。

## NULL値と結合条件

テーブル結合（JOINなど、後の章で学習）の際も、NULL値は特殊な扱いを受けます。NULL値同士は「等しい」とは判定されないため、通常の結合条件ではNULL値を持つレコードは結合されません。

## IS NULLとIS NOT NULLを使った複合条件

IS NULLとIS NOT NULLも、他の条件と組み合わせて使用できます。

### 例：複合条件でのIS NULLの使用

例えば、「出席状態が "absent"（欠席）で、コメントがNULLでない（理由が入力されている）レコード」を検索するには：

```
SELECT * FROM attendance
WHERE status = 'absent' AND comment IS NOT NULL;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 303 | absent | 事前連絡あり |
| 1 | 317 | absent | 体調不良 |
| … | … | … | … |

## NVL/IFNULL/COALESCE関数：NULL値の置換

NULL値を別の値に置き換えるための関数が用意されています。データベースによって関数名が異なることがありますが、機能は似ています：

- MySQL/MariaDB: **IFNULL(expr, replace_value)**
- Oracle: **NVL(expr, replace_value)**
- SQL Server: **ISNULL(expr, replace_value)**
- 標準SQL: **COALESCE(expr1, expr2, ..., exprN)** - 最初のNULLでない式を返します

### 例：IFNULL関数の使用（MySQL）

例えば、コメントがNULLの場合は「特記事項なし」と表示するには：

```
SELECT schedule_id, student_id, status,
       IFNULL(comment, '特記事項なし') AS comment
FROM attendance;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 301 | present | 特記事項なし |
| 1 | 302 | late | 15分遅刻 |
| 1 | 303 | absent | 事前連絡あり |
| ... | ... | ... | ... |

## NULLを使う際の注意点

1. **除外の罠**：`WHERE カラム名 <> 値` だけでは、NULL値を持つレコードは含まれません。すべてのレコードを対象にするには：

   ```
   WHERE カラム名 <> 値 OR カラム名 IS NULL
   ```

2. **集計関数**：COUNT(*)はすべての行を数えますが、COUNT(カラム名)はそのカラムがNULLでない行だけを数えます。

3. **インデックス**：多くのデータベースでは、NULL値にもインデックスを適用できますが、データベースによって動作が異なる場合があります。

4. **一意性制約**：一般的に、UNIQUE制約ではNULL値は重複としてカウントされません（複数のNULL値が許可されます）。

## 練習問題

### 問題6-1

attendance（出席）テーブルから、コメント（comment）がNULLの出席レコードをすべて取得するSQLを書いてください。

### 問題6-2

course_schedule（授業カレンダー）テーブルから、状態（status）が「cancelled」で、かつ教室ID（classroom_id）がNULLでないレコードを取得するSQLを書いてください。

### 問題6-3

grades（成績）テーブルから、提出日（submission_date）がNULLの成績レコードを取得するSQLを書いてください。

### 問題6-4

attendance（出席）テーブルから、出席状態（status）が「present」か「late」で、かつコメント（comment）がNULLのレコードを取得するSQLを書いてください。

### 問題6-5

以下のSQLで教師（teachers）テーブルから「佐藤」という名前を持つ教師を検索する場合、NULL値を持つレコードも含めるにはどう修正すべきですか？

```
SELECT * FROM teachers WHERE teacher_name <> '佐藤花子';
```

### 問題6-6

attendance（出席）テーブルのすべてのレコードを取得し、コメント（comment）がNULLの場合は「記録なし」と表示するSQLを書いてください。

## 解答

### 解答6-1

```
SELECT * FROM attendance WHERE comment IS NULL;
```

### 解答6-2

```
SELECT * FROM course_schedule
WHERE status = 'cancelled' AND classroom_id IS NOT NULL;
```

### 解答6-3

```
SELECT * FROM grades WHERE submission_date IS NULL;
```

### 解答6-4

```
SELECT * FROM attendance
WHERE (status = 'present' OR status = 'late') AND comment IS NULL;
```

または

```

```

```sql
SELECT * FROM attendance
WHERE status IN ('present', 'late') AND comment IS NULL;
```

解答6-5

```sql
SELECT * FROM teachers
WHERE teacher_name <> '佐藤花子' OR teacher_name IS NULL;
```

解答6-6

```sql
SELECT schedule_id, student_id, status,
       IFNULL(comment, '記録なし') AS comment
FROM attendance;
```

## まとめ

この章では、データベースにおけるNULL値の概念と、NULL値を扱うための演算子や関数について学びました：

1. **NULL値の概念**：値がない、不明、未設定を表す特殊な値
2. **IS NULL演算子**：NULL値を持つレコードを検索する方法
3. **IS NOT NULL演算子**：NULL値を持たないレコードを検索する方法
4. **NULL値の論理的扱い**：論理演算（AND、OR、NOT）におけるNULLの振る舞い
5. **複合条件**：IS NULL/IS NOT NULLと他の条件の組み合わせ
6. **NULL値の置換**：IFNULL/NVL/COALESCE関数の使用方法
7. **注意点**：NULL値を扱う際の一般的な落とし穴

NULL値の正確な理解と適切な処理は、SQLプログラミングの重要な部分です。不適切なNULL処理は、予期しない結果やバグの原因になります。

次の章では、クエリ結果の並び替えを行うための「ORDER BY：結果の並び替え」について学びます。

---

# 7. ORDER BY：結果の並び替え

## はじめに

これまでの章では、データベースから条件に合ったレコードを取得する方法を学んできました。しかし実際の業務では、取得したデータを見やすく整理する必要があります。例えば：

- 成績を高い順に表示したい
- 学生を名前の五十音順に並べたい
- 日付の新しい順にスケジュールを確認したい

このようなデータの「並び替え」を行うためのSQLコマンドが「ORDER BY」です。この章では、クエリ結果を特定の順序で並べる方法を学びます。

## ORDER BYの基本

ORDER BY句は、SELECT文の結果を指定したカラムの値に基づいて並び替えるために使います。

> **用語解説：**
>
> - **ORDER BY**：「〜の順に並べる」という意味のSQLコマンドで、クエリ結果の並び順を指定します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] ORDER BY 並び替えカラム;
```

ORDER BY句は通常、SELECT文の最後に記述します。

### 例1：単一カラムでの並び替え

例えば、学生（students）テーブルから、学生名（student_name）の五十音順（辞書順）でデータを取得するには：

```
SELECT * FROM students ORDER BY student_name;
```

実行結果：

| student_id | student_name |
|---|---|
| 309 | 相沢吉夫 |
| 303 | 柴崎春花 |
| 306 | 河田咲奈 |
| 305 | 河口菜恵子 |
| … | … |

### デフォルトの並び順

ORDER BYを使わない場合、結果の順序は保証されません。多くの場合、データがデータベースに保存された順序で返されますが、これは信頼できるものではありません。

## 昇順と降順の指定

ORDER BY句では、並び順を「昇順」か「降順」のどちらかで指定できます。

> **用語解説：**

> - **昇順（ASC）**：小さい値から大きい値へ（A→Z、1→9）の順に並べます。
> - **降順（DESC）**：大きい値から小さい値へ（Z→A、9→1）の順に並べます。

### 構文

```
SELECT カラム名 FROM テーブル名 ORDER BY 並び替えカラム [ASC|DESC];
```

ASC（昇順）がデフォルトのため、省略可能です。

### 例2：降順での並び替え

例えば、成績（grades）テーブルから、得点（score）の高い順（降順）に成績を取得するには：

```
SELECT * FROM grades ORDER BY score DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

## 複数カラムでの並び替え

複数のカラムを使って並び替えることもできます。最初に指定したカラムで並び替え、値が同じレコードがある場合は次のカラムで並び替えます。

### 構文

```
SELECT カラム名 FROM テーブル名
ORDER BY 並び替えカラム1 [ASC|DESC], 並び替えカラム2 [ASC|DESC], ...;
```

### 例3：複数カラムでの並び替え

例えば、成績（grades）テーブルから、課題タイプ（grade_type）の五十音順に並べ、同じ課題タイプ内では得点（score）の高い順に成績を取得するには：

```
SELECT * FROM grades
ORDER BY grade_type ASC, score DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 2 | 実技試験 | 88.0 | 100.0 | 2025-05-18 |
| 321 | 2 | 実技試験 | 85.5 | 100.0 | 2025-05-18 |
| … | … | … | … | … | … |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |
| 311 | 1 | レポート1 | 49.0 | 50.0 | 2025-05-08 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| … | … | … | … | … | … |

### 例4：昇順と降順の混合

各カラムごとに並び順を指定することもできます。例えば、講座ID（course_id）の昇順、評価タイプ（grade_type）の昇順、得点（score）の降順で並べるには：

```
SELECT * FROM grades
ORDER BY course_id ASC, grade_type ASC, score DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 311 | 1 | レポート1 | 49.0 | 50.0 | 2025-05-08 |
| … | … | … | … | … | … |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |
| 301 | 2 | 実技試験 | 88.0 | 100.0 | 2025-05-18 |
| … | … | … | … | … | … |

## NULLの扱い

ORDER BYでNULL値を並び替える場合、データベース製品によって動作が異なります。多くのデータベースでは、NULL値は最小値または最大値として扱われます。

- MySQL/MariaDBでは、NULL値は昇順（ASC）の場合は最小値として（最初に表示）、降順（DESC）の場合は最大値として（最後に表示）扱われます。

一部のデータベース（PostgreSQLなど）では、NULL値の位置を明示的に指定するための「NULLS FIRST」「NULLS LAST」構文がサポートされています。

### 例5：NULL値の扱い

例えば、出席（attendance）テーブルからコメント（comment）でソートすると、NULLが最初に来ます：

```
SELECT * FROM attendance ORDER BY comment;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 301 | present | NULL |
| 1 | 306 | present | NULL |
| ... | ... | ... | NULL |
| 1 | 308 | late | 5分遅刻 |
| 1 | 323 | late | 電車遅延 |
| ... | ... | ... | ... |

## カラム番号を使った並び替え

カラム名の代わりに、SELECT文の結果セットにおけるカラムの位置（番号）を使って並び替えることもできます。最初のカラムは1、2番目のカラムは2、という具合です。

### 構文

```
SELECT カラム名1, カラム名2, ... FROM テーブル名 ORDER BY カラム位置;
```

### 例6：カラム番号を使った並び替え

例えば、学生（students）テーブルから学生ID（student_id）と名前（student_name）を取得し、名前（2番目のカラム）で並べ替えるには：

```
SELECT student_id, student_name FROM students ORDER BY 2;
```

この場合、「ORDER BY 2」は「ORDER BY student_name」と同じ意味になります。

> **注意**：カラム番号を使う方法は、カラムの順序を変更すると問題が起きるため、実際の業務では使用を避けた方が良いとされています。

## 式や関数を使った並び替え

ORDER BY句で式や関数を使うことにより、計算結果に基づいて並び替えることもできます。

### 例7：式を使った並び替え

例えば、成績（grades）テーブルから、得点の達成率（score/max_score）の高い順に並べるには：

```
SELECT student_id, course_id, grade_type, score, max_score,
       (score/max_score)*100 AS 達成率
FROM grades
ORDER BY (score/max_score) DESC;
```

または

```
SELECT student_id, course_id, grade_type, score, max_score,
       (score/max_score)*100 AS 達成率
FROM grades
ORDER BY 達成率 DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | 達成率 |
|---|---|---|---|---|---|
| 311 | 1 | レポート1 | 49.0 | 50.0 | 98.0 |
| 320 | 1 | レポート1 | 48.5 | 50.0 | 97.0 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 95.0 |
| … | … | … | … | … | … |

### 例8：関数を使った並び替え

文字列関数を使って並び替えることもできます。例えば、月名で並べることを考えましょう：

```
SELECT schedule_date, MONTH(schedule_date) AS month
FROM course_schedule
ORDER BY MONTH(schedule_date);
```

実行結果：

| schedule_date | month |
|---|---|
| 2025-04-07 | 4 |
| 2025-04-08 | 4 |
| … | … |

| schedule_date | month |
|---|---|
| 2025-05-01 | 5 |
| 2025-05-02 | 5 |
| … | … |
| 2025-06-01 | 6 |
| … | … |

## CASE式を使った条件付き並び替え

さらに高度な並び替えとして、CASE式を使って条件に応じた並び順を定義することもできます。

### 例9：CASE式を使った並び替え

例えば、出席（attendance）テーブルから、出席状況（status）を「欠席→遅刻→出席」の順に優先して表示するには：

```
SELECT * FROM attendance
ORDER BY CASE
           WHEN status = 'absent' THEN 1
           WHEN status = 'late' THEN 2
           WHEN status = 'present' THEN 3
           ELSE 4
         END;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 303 | absent | 事前連絡あり |
| 1 | 317 | absent | 体調不良 |
| … | … | … | … |
| 1 | 302 | late | 15分遅刻 |
| 1 | 308 | late | 5分遅刻 |
| … | … | … | … |
| 1 | 301 | present | NULL |
| 1 | 306 | present | NULL |
| … | … | … | … |

## 練習問題

### 問題7-1

students（学生）テーブルから、すべての学生情報を学生名（student_name）の降順（逆五十音順）で取得するSQLを書いてください。

### 問題7-2

grades（成績）テーブルから、得点（score）が85点以上の成績を得点の高い順に取得するSQLを書いてください。

### 問題7-3

course_schedule（授業カレンダー）テーブルから、2025年5月の授業スケジュールを日付（schedule_date）の昇順で取得するSQLを書いてください。

### 問題7-4

teachers（教師）テーブルから、教師IDと名前を取得し、名前（teacher_name）の五十音順で並べるSQLを書いてください。

### 問題7-5

grades（成績）テーブルから、講座ID（course_id）ごとに、成績を評価タイプ（grade_type）の五十音順に、同じ評価タイプ内では得点（score）の高い順に並べて取得するSQLを書いてください。

### 問題7-6

attendance（出席）テーブルから、すべての出席情報を出席状況（status）が「absent」「late」「present」の順番で、同じ状態内ではコメント（comment）の有無（NULLが後）で並べて取得するSQLを書いてください。

## 解答

### 解答7-1

```sql
SELECT * FROM students ORDER BY student_name DESC;
```

### 解答7-2

```sql
SELECT * FROM grades WHERE score >= 85 ORDER BY score DESC;
```

### 解答7-3

```sql
SELECT * FROM course_schedule
WHERE schedule_date BETWEEN '2025-05-01' AND '2025-05-31'
ORDER BY schedule_date;
```

解答7-4

```
SELECT teacher_id, teacher_name FROM teachers ORDER BY teacher_name;
```

解答7-5

```
SELECT * FROM grades
ORDER BY course_id, grade_type, score DESC;
```

解答7-6

```
SELECT * FROM attendance
ORDER BY
  CASE
    WHEN status = 'absent' THEN 1
    WHEN status = 'late' THEN 2
    WHEN status = 'present' THEN 3
    ELSE 4
  END,
  CASE
    WHEN comment IS NULL THEN 2
    ELSE 1
  END;
```

## まとめ

この章では、クエリ結果を特定の順序で並べるための「ORDER BY」句について学びました：

1. **基本的な並び替え**：指定したカラムの値に基づいて結果を並べる方法
2. **昇順と降順**：ASC（昇順）とDESC（降順）の指定方法
3. **複数カラムでの並び替え**：優先順位の高いカラムから順に指定する方法
4. **NULL値の扱い**：NULL値が並び替えでどのように扱われるか
5. **カラム番号**：カラム名の代わりに位置で指定する方法（あまり推奨されない）
6. **式や関数**：計算結果に基づいて並べる方法
7. **CASE式**：条件付きの複雑な並び替え

ORDER BY句は、データを見やすく整理するために非常に重要です。特に大量のデータを扱う場合、適切な並び順はデータの理解を大きく助けます。

次の章では、取得する結果の件数を制限する「LIMIT句：結果件数の制限とページネーション」について学びます。

---

# 8. LIMIT句：結果件数の制限とページネーション

# はじめに

これまでの章では、条件に合うデータを取得し、それを特定の順序で並べる方法を学びました。しかし実際のアプリケーションでは、大量のデータがあるときに、その一部だけを表示したいことがよくあります。例えば：

- 成績上位10件だけを表示したい
- Webページで一度に20件ずつ表示したい（ページネーション）
- 最新の5件のお知らせだけを取得したい

このような「結果の件数を制限する」ためのSQLコマンドが「LIMIT句」です。この章では、クエリ結果の件数を制限する方法と、ページネーションの実装方法を学びます。

# LIMIT句の基本

LIMIT句は、SELECT文の結果から指定した件数だけを取得するために使います。

> **用語解説**：
>
> - **LIMIT**：「制限する」という意味のSQLコマンドで、取得する行数を制限します。

## 基本構文（MySQL/MariaDB）

MySQLやMariaDBでのLIMIT句の基本構文は次のとおりです：

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] [ORDER BY 並び順] LIMIT 件数;
```

LIMIT句は通常、SELECT文の最後に記述します（ORDER BYの後）。

## 例1：単純なLIMIT

例えば、学生（students）テーブルから最初の5人だけを取得するには：

```
SELECT * FROM students LIMIT 5;
```

実行結果：

| student_id | student_name |
|---|---|
| 301 | 黒沢春馬 |
| 302 | 新垣愛留 |
| 303 | 柴崎春花 |
| 304 | 森下風凛 |
| 305 | 河口菜恵子 |

### ORDER BYとLIMITの組み合わせ

通常、LIMIT句はORDER BY句と組み合わせて使用します。これにより、「上位N件」「最新N件」などの操作が可能になります。

### 例2：ORDER BYとLIMITの組み合わせ

例えば、成績（grades）テーブルから得点（score）の高い順に上位3件を取得するには：

```
SELECT * FROM grades ORDER BY score DESC LIMIT 3;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |

### 例3：最新のレコードを取得

日付でソートして最新のデータを取得することもよくあります。例えば、最新の3つの授業スケジュールを取得するには：

```
SELECT * FROM course_schedule
ORDER BY schedule_date DESC LIMIT 3;
```

実行結果：

| schedule_id | course_id | schedule_date | period_id | classroom_id | teacher_id | status |
|---|---|---|---|---|---|---|
| 95 | 28 | 2026-12-21 | 3 | 202D | 119 | scheduled |
| 94 | 1 | 2026-12-21 | 1 | 102B | 101 | scheduled |
| 93 | 14 | 2026-12-15 | 4 | 202D | 110 | scheduled |

## OFFSETとページネーション

Webアプリケーションなどでは、大量のデータを「ページ」に分けて表示することがよくあります（ページネーション）。この機能を実現するためには、「OFFSET」（オフセット）という機能が必要です。

> **用語解説**：
>
> - **OFFSET**：「ずらす」という意味で、結果セットの先頭から指定した数だけ行をスキップします。
> - **ページネーション**：大量のデータを複数のページに分割して表示する技術です。

### 基本構文（MySQL/MariaDB）

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] [ORDER BY 並び順] LIMIT 件数 OFFSET スキップ数;
```

または、短縮形として：

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] [ORDER BY 並び順] LIMIT スキップ数, 件数;
```

### 例4：OFFSETを使ったスキップ

例えば、学生（students）テーブルから6番目から10番目までの学生を取得するには：

```
SELECT * FROM students LIMIT 5 OFFSET 5;
```

または：

```
SELECT * FROM students LIMIT 5, 5;
```

実行結果：

| student_id | student_name |
|---|---|
| 306 | 河田咲奈 |
| 307 | 織田柚夏 |
| 308 | 永田悦子 |
| 309 | 相沢吉夫 |
| 310 | 吉川伽羅 |

### 例5：ページネーションの実装

ページネーションを実装する場合、通常は以下の式を使ってOFFSETを計算します：

```
OFFSET = (ページ番号 - 1) × ページあたりの件数
```

例えば、1ページあたり10件表示で、3ページ目のデータを取得するには：

```
SELECT * FROM students ORDER BY student_id LIMIT 10 OFFSET 20;
```

または：

```
SELECT * FROM students ORDER BY student_id LIMIT 20, 10;
```

実行結果：

| student_id | student_name |
|---|---|
| 321 | 井上竜也 |
| 322 | 木村結衣 |
| 323 | 林正義 |
| 324 | 清水香織 |
| 325 | 山田翔太 |
| 326 | 葉山陽太 |
| 327 | 青山凛 |
| 328 | 沢村大和 |
| 329 | 白石優月 |
| 330 | 月岡星奈 |

## LIMIT句を使用する際の注意点

### 1. ORDER BYの重要性

LIMIT句を使用する場合、通常はORDER BY句も一緒に使うべきです。ORDER BYがなければ、どのレコードが取得されるかは保証されません。

```
-- 良い例：結果が予測可能
SELECT * FROM students ORDER BY student_id LIMIT 5;

-- 悪い例：結果が不確定
SELECT * FROM students LIMIT 5;
```

### 2. パフォーマンスへの影響

大規模なテーブルで大きなOFFSET値を使用すると、パフォーマンスが低下する可能性があります。これは、データベースがOFFSET分のレコードを読み込んでから破棄する必要があるためです。

### 3. データベース製品による構文の違い

LIMIT句の構文はデータベース製品によって異なります：

- **MySQL/MariaDB/SQLite**：`LIMIT 件数 OFFSET スキップ数` または `LIMIT スキップ数, 件数`
- **PostgreSQL**：`LIMIT 件数 OFFSET スキップ数`
- **Oracle**：`OFFSET スキップ数 ROWS FETCH NEXT 件数 ROWS ONLY`
- **SQL Server**：`OFFSET スキップ数 ROWS FETCH NEXT 件数 ROWS ONLY` または旧バージョンでは `TOP`句

この章では主にMySQL/MariaDBの構文を使用します。

## 実践的なページネーションの実装

実際のアプリケーションでページネーションを実装する場合、以下のようなコードになります（疑似コード）：

```
ページ番号 = URLから取得またはデフォルト値（例：1）
1ページあたりの件数 = 設定値（例：10）
総レコード数 = SELECTで取得（COUNT(*)を使用）
総ページ数 = CEILING(総レコード数 ÷ 1ページあたりの件数)
OFFSET = (ページ番号 - 1) × 1ページあたりの件数

SQLクエリ = "SELECT * FROM テーブル ORDER BY カラム LIMIT " + 1ページあたりの件数 + " OFFSET
" + OFFSET
```

### 例6：総レコード数と総ページ数の取得

総レコード数を取得するには：

```sql
SELECT COUNT(*) AS total_records FROM students;
```

実行結果：

| total_records |
|---|
| 100 |

この場合、1ページあたり10件表示なら、総ページ数は10ページ（CEILING(100 ÷ 10)）になります。

## 練習問題

### 問題8-1

grades（成績）テーブルから、得点（score）の高い順に上位5件の成績レコードを取得するSQLを書いてください。

### 問題8-2

course_schedule（授業カレンダー）テーブルから、日付（schedule_date）の新しい順に3件のスケジュールを取得するSQLを書いてください。

### 問題8-3

students（学生）テーブルを学生ID（student_id）の昇順で並べ、11番目から15番目までの学生（5件）を取得するSQLを書いてください。

### 問題8-4

teachers（教師）テーブルから、教師名（teacher_name）の五十音順で6番目から10番目までの教師情報を取得するSQLを書いてください。

### 問題8-5

1ページあたり20件表示で、grades（成績）テーブルの3ページ目のデータを得点（score）の高い順に取得するSQLを書いてください。

### 問題8-6

course_schedule（授業カレンダー）テーブルから、状態（status）が「scheduled」のスケジュールを日付（schedule_date）の昇順で並べ、先頭から10件スキップして次の5件を取得するSQLを書いてください。

## 解答

### 解答8-1

```
SELECT * FROM grades ORDER BY score DESC LIMIT 5;
```

### 解答8-2

```
SELECT * FROM course_schedule ORDER BY schedule_date DESC LIMIT 3;
```

### 解答8-3

```
SELECT * FROM students ORDER BY student_id LIMIT 5 OFFSET 10;
```

または

```
SELECT * FROM students ORDER BY student_id LIMIT 10, 5;
```

### 解答8-4

```
SELECT * FROM teachers ORDER BY teacher_name LIMIT 5 OFFSET 5;
```

または

```
SELECT * FROM teachers ORDER BY teacher_name LIMIT 5, 5;
```

解答8-5

```
SELECT * FROM grades ORDER BY score DESC LIMIT 20 OFFSET 40;
```

または

```
SELECT * FROM grades ORDER BY score DESC LIMIT 40, 20;
```

解答8-6

```
SELECT * FROM course_schedule
WHERE status = 'scheduled'
ORDER BY schedule_date
LIMIT 5 OFFSET 10;
```

または

```
SELECT * FROM course_schedule
WHERE status = 'scheduled'
ORDER BY schedule_date
LIMIT 10, 5;
```

# まとめ

この章では、クエリ結果の件数を制限するためのLIMIT句と、ページネーションの実装方法について学びました：

1. **LIMIT句の基本**：指定した件数だけのレコードを取得する方法
2. **ORDER BYとの組み合わせ**：順序付けされた結果から一部だけを取得する方法
3. **OFFSET**：結果の先頭から指定した数だけレコードをスキップする方法
4. **ページネーション**：大量のデータを複数のページに分けて表示する実装方法
5. **注意点**：LIMIT句を使用する際の留意事項
6. **データベース製品による違い**：異なるデータベースでの構文の違い

LIMIT句は特にWebアプリケーションの開発で重要な機能です。大量のデータを効率よく表示するためのページネーション機能を実装するために欠かせません。また、トップN（上位N件）やボトムN（下位N件）のデータを取得する際にも使われます。

次の章では、データの集計分析を行うための「集計関数：COUNT、SUM、AVG、MAX、MIN」について学びます。

---

# SQL学習テキスト 完全版

このテキストは、SQLの基礎から応用まで体系的に学習できるように構成されています。 学校データベースを使った実践的な例題と練習問題を通じて、実務で使えるSQLスキルを身につけることができます。

---

## 目次



---

# 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 |

| 関数 | 説明 | 例 |
|---|---|---|
| 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 |
|---|

| 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 |
| --- |
| 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：

```sql
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：

```sql
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 |
|---|---|---|---|---|
| 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 |
| … | … |

### 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

```
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

```
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

```
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 |

| grade_type |
| --- |
| 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 |

| course_id | period_id |
|---|---|
| 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 |
| --- |
| 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。

---

# SQL学習テキスト 完全版

このテキストは、SQLの基礎から応用まで体系的に学習できるように構成されています。 学校データベースを使った実践的な例題と練習問題を通じて、実務で使えるSQLスキルを身につけることができます。

---

## 目次



---

# 1. SELECT基本：単一テーブルから特定カラムを取得する

## はじめに

データベースからデータを取り出す作業は、料理人が大きな冷蔵庫から必要な材料だけを取り出すようなものです。SQLでは、この「取り出す」作業を「SELECT文」で行います。

SELECT文はSQLの中で最も基本的で、最もよく使われる命令です。この章では、単一のテーブル（データの表）から必要な情報だけを取り出す方法を学びます。

## 基本構文

SELECT文の最も基本的な形は次のとおりです：

```
SELECT カラム名 FROM テーブル名;
```

この文は「テーブル名というテーブルからカラム名という列のデータを取り出してください」という意味です。

> **用語解説：**
>
> - **SELECT**：「選択する」という意味のSQLコマンドで、データを取り出すときに使います。
> - **カラム**：テーブルの縦の列のことで、同じ種類のデータが並んでいます（例：名前のカラム、年齢のカラムなど）。
> - **FROM**：「～から」という意味で、どのテーブルからデータを取るかを指定します。
> - **テーブル**：データベース内の表のことで、行と列で構成されています。

## 実践例：単一カラムの取得

学校データベースの中の「teachers」テーブル（教師テーブル）から、教師の名前だけを取得してみましょう。

```
SELECT teacher_name FROM teachers;
```

実行結果：

| teacher_name |
| --- |
| 寺内鞍 |
| 田尻朋美 |
| 内村海凪 |
| 藤本理恵 |
| 黒木大介 |
| 星野涼子 |
| 深山誠一 |
| 吉岡由佳 |
| 山田太郎 |
| 佐藤花子 |

…

これは「teachers」テーブルの「teacher_name」という列（先生の名前）だけを取り出しています。

## 複数のカラムを取得する

料理に複数の材料が必要なように、データを取り出すときも複数の列が必要なことがよくあります。複数のカラムを取得するには、カラム名をカンマ（,）で区切って指定します。

```
SELECT カラム名1, カラム名2, カラム名3 FROM テーブル名;
```

例えば、教師の番号（ID）と名前を一緒に取得してみましょう：

```
SELECT teacher_id, teacher_name FROM teachers;
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 101 | 寺内鞍 |
| 102 | 田尻朋美 |
| 103 | 内村海凪 |
| 104 | 藤本理恵 |
| 105 | 黒木大介 |
| 106 | 星野涼子 |
| … | … |

## すべてのカラムを取得する

テーブルのすべての列を取得したい場合は、アスタリスク（*）を使います。これは「すべての列」を意味するワイルドカードです。

```
SELECT * FROM テーブル名;
```

例：

```
SELECT * FROM teachers;
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 101 | 寺内鞍 |
| 102 | 田尻朋美 |
| 103 | 内村海凪 |
| … | … |

> **注意**：`SELECT *`は便利ですが、実際の業務では必要なカラムだけを指定する方が良いとされています。これは、データ量が多いときに処理速度が遅くなるのを防ぐためです。

## カラムに別名をつける（AS句）

取得したカラムに分かりやすい名前（別名）をつけることができます。これは「AS」句を使います。

```
SELECT カラム名 AS 別名 FROM テーブル名;
```

> **用語解説**：
>
> - **AS**：「～として」という意味で、カラムに別名をつけるときに使います。この別名は結果を表示するときだけ使われます。

例えば、教師IDを「番号」、教師名を「名前」として表示してみましょう：

```
SELECT teacher_id AS 番号, teacher_name AS 名前 FROM teachers;
```

実行結果：

| 番号 | 名前 |
|---|---|
| 101 | 寺内鞍 |
| 102 | 田尻朋美 |
| 103 | 内村海凪 |
| … | … |

ASは省略することも可能です：

```
SELECT teacher_id 番号, teacher_name 名前 FROM teachers;
```

## 計算式を使う

SELECT文では、カラムの値を使った計算もできます。例えば、成績テーブルから点数と満点を取得して、達成率（パーセント）を計算してみましょう。

```
SELECT student_id, course_id, grade_type,
       score, max_score,
       (score / max_score) * 100 AS 達成率
FROM grades;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | 達成率 |
|---|---|---|---|---|---|

| student_id | course_id | grade_type | score | max_score | 達成率 |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 85.5 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 92.0 |
| ... | ... | ... | ... | ... | ... |

## 重複を除外する（DISTINCT）

同じ値が複数ある場合に、重複を除いて一意の値だけを表示するには「DISTINCT」キーワードを使います。

> **用語解説**：
>
> - **DISTINCT**：「異なる」「区別された」という意味で、重複する値を除外して一意の値だけを取得します。

例えば、どの講座にどの教師が担当しているかを重複なしで確認してみましょう：

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

実行結果：

| teacher_id |
|---|
| 101 |
| 102 |
| 103 |
| 104 |
| ... |

これにより、courses（講座）テーブルで使われている教師IDが重複なく表示されます。

## 文字列の結合

文字列を結合するには、MySQLでは「CONCAT」関数を使います。例えば、教師のIDと名前を組み合わせて表示してみましょう：

```
SELECT CONCAT('教師ID:', teacher_id, ' 名前:', teacher_name) AS 教師情報
FROM teachers;
```

実行結果：

| 教師情報 |
|---|
| 教師ID:101 名前:寺内鞍 |

| 教師情報 |
| --- |
| 教師ID:102 名前:田尻朋美 |

...

> **用語解説：**
>
> - **CONCAT**：複数の文字列を一つにつなげる関数です。

## SELECT文と終了記号

SQLの文は通常、セミコロン（;）で終わります。これは「この命令はここで終わりです」という合図です。

複数のSQL文を一度に実行する場合は、それぞれの文の最後にセミコロンをつけます。

## 練習問題

### 問題1-1

students（学生）テーブルから、すべての学生の名前（student_name）を取得するSQLを書いてください。

### 問題1-2

classrooms（教室）テーブルから、教室ID（classroom_id）と教室名（classroom_name）を取得するSQLを書いてください。

### 問題1-3

courses（講座）テーブルから、すべての列（カラム）を取得するSQLを書いてください。

### 問題1-4

class_periods（授業時間）テーブルから、時限ID（period_id）、開始時間（start_time）、終了時間（end_time）を取得し、開始時間には「開始」、終了時間には「終了」という別名をつけるSQLを書いてください。

### 問題1-5

grades（成績）テーブルから、学生ID（student_id）、講座ID（course_id）、評価タイプ（grade_type）、得点（score）、満点（max_score）、そして得点を満点で割って100を掛けた値を「パーセント」という別名で取得するSQLを書いてください。

### 問題1-6

course_schedule（授業カレンダー）テーブルから、schedule_date（予定日）カラムだけを重複なしで取得するSQLを書いてください。

## 解答

### 解答1-1

```
SELECT student_name FROM students;
```

解答1-2

```
SELECT classroom_id, classroom_name FROM classrooms;
```

解答1-3

```
SELECT * FROM courses;
```

解答1-4

```
SELECT period_id, start_time AS 開始, end_time AS 終了 FROM class_periods;
```

解答1-5

```
SELECT student_id, course_id, grade_type, score, max_score,
       (score / max_score) * 100 AS パーセント
FROM grades;
```

解答1-6

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

## まとめ

この章では、SQLのSELECT文の基本を学びました：

1. 単一カラムの取得: `SELECT カラム名 FROM テーブル名;`
2. 複数カラムの取得: `SELECT カラム名1, カラム名2 FROM テーブル名;`
3. すべてのカラムの取得: `SELECT * FROM テーブル名;`
4. カラムに別名をつける: `SELECT カラム名 AS 別名 FROM テーブル名;`
5. 計算式を使う: `SELECT カラム名, (計算式) AS 別名 FROM テーブル名;`
6. 重複を除外する: `SELECT DISTINCT カラム名 FROM テーブル名;`
7. 文字列の結合: `SELECT CONCAT(文字列1, カラム名, 文字列2) FROM テーブル名;`

これらの基本操作を使いこなせるようになれば、データベースから必要な情報を効率よく取り出せるようになります。次の章では、WHERE句を使って条件に合ったデータだけを取り出す方法を学びます。

# 2. WHERE句：条件に合ったレコードを絞り込む

## はじめに

前章では、テーブルからデータを取得する基本的な方法を学びました。しかし実際の業務では、すべてのデータではなく、特定の条件に合ったデータだけを取得したいことがほとんどです。

例えば、「全生徒の情報」ではなく「特定の学科の生徒だけ」や「成績が80点以上の学生だけ」といった形で、データを絞り込みたい場合があります。

このような場合に使用するのが「WHERE句」です。WHERE句は、SELECTコマンドの後に追加して使い、条件に合致するレコード（行）だけを取得します。

## 基本構文

WHERE句の基本的な形は次のとおりです：

```
SELECT カラム名 FROM テーブル名 WHERE 条件式;
```

> **用語解説**：
>
> - **WHERE**：「〜の場所で」「〜の条件で」という意味のSQLコマンドで、条件に合うデータだけを抽出するために使います。
> - **条件式**：データが満たすべき条件を指定するための式です。例えば「age > 20」（年齢が20より大きい）などです。
> - **レコード**：テーブルの横の行のことで、1つのデータの集まりを表します。

## 基本的な比較演算子

WHERE句では、様々な比較演算子を使って条件を指定できます：

| 演算子 | 意味 | 例 |
|---|---|---|
| = | 等しい | age = 25 |
| <> | 等しくない（≠と同じ） | gender <> 'male' |
| > | より大きい | score > 80 |
| < | より小さい | price < 1000 |
| >= | 以上 | height >= 170 |
| <= | 以下 | weight <= 70 |

## 実践例：基本的な条件での絞り込み

### 例1：等しい（=）

例えば、教師ID（teacher_id）が101の教師のみを取得するには：

```
SELECT * FROM teachers WHERE teacher_id = 101;
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 101 | 寺内鞍 |

### 例2：より大きい（>）

成績（grades）テーブルから、90点を超える成績だけを取得するには：

```
SELECT * FROM grades WHERE score > 90;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

### 例3：等しくない（<>）

講座IDが3ではない講座に関する成績を取得するには：

```
SELECT * FROM grades WHERE course_id <> '3';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 301 | 2 | 実技試験 | 88.0 | 100.0 | 2025-05-18 |
| … | … | … | … | … | … |

## 文字列の比較

テキスト（文字列）を条件にする場合は、シングルクォーテーション（'）またはダブルクォーテーション（"）で囲みます。MySQLではどちらも使えますが、多くの場合シングルクォーテーションが推奨されます。

```
SELECT * FROM テーブル名 WHERE テキストカラム = 'テキスト値';
```

例えば、教師名（teacher_name）が「田尻朋美」の教師を検索するには：

```
SELECT * FROM teachers WHERE teacher_name = '田尻朋美';
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 102 | 田尻朋美 |

## 日付の比較

日付の比較も同様にシングルクォーテーションで囲みます。日付の形式はデータベースの設定によって異なりますが、一般的にはISO形式（YYYY-MM-DD）が使われます。

```
SELECT * FROM テーブル名 WHERE 日付カラム = '日付';
```

例えば、2025年5月20日に提出された成績を検索するには：

```
SELECT * FROM grades WHERE submission_date = '2025-05-20';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 303 | 1 | 中間テスト | 78.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

また、日付同士の大小関係も比較できます：

```
SELECT * FROM grades WHERE submission_date < '2025-05-15';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | レポート1 | 45.0 | 50.0 | 2025-05-10 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| 304 | 5 | ER図作成課題 | 27.5 | 30.0 | 2025-05-14 |
| … | … | … | … | … | … |

## 複数の条件を指定する（次章の内容）

WHERE句では、複数の条件を組み合わせることもできます。この詳細は次章「論理演算子：AND、OR、NOTを使った複合条件」で説明します。

例えば、講座IDが1で、かつ、得点が90以上の成績を取得するには：

```
SELECT * FROM grades WHERE course_id = '1' AND score >= 90;
```

この例の詳細な説明は次章で行います。

## 練習問題

### 問題2-1

students（学生）テーブルから、学生ID（student_id）が「310」の学生情報を取得するSQLを書いてください。

### 問題2-2

classrooms（教室）テーブルから、収容人数（capacity）が30人より多い教室の情報をすべて取得するSQLを書いてください。

### 問題2-3

courses（講座）テーブルから、教師ID（teacher_id）が「105」の講座情報を取得するSQLを書いてください。

### 問題2-4

course_schedule（授業カレンダー）テーブルから、「2025-05-15」の授業スケジュールをすべて取得するSQLを書いてください。

### 問題2-5

grades（成績）テーブルから、評価タイプ（grade_type）が「中間テスト」で、得点（score）が80点未満の成績を取得するSQLを書いてください。

### 問題2-6

teachers（教師）テーブルから、教師ID（teacher_id）が「101」ではない教師の名前を取得するSQLを書いてください。

## 解答

### 解答2-1

```
SELECT * FROM students WHERE student_id = 310;
```

### 解答2-2

```
SELECT * FROM classrooms WHERE capacity > 30;
```

### 解答2-3

```
SELECT * FROM courses WHERE teacher_id = 105;
```

### 解答2-4

```
SELECT * FROM course_schedule WHERE schedule_date = '2025-05-15';
```

### 解答2-5

```
SELECT * FROM grades WHERE grade_type = '中間テスト' AND score < 80;
```

### 解答2-6

```
SELECT teacher_name FROM teachers WHERE teacher_id <> 101;
```

## まとめ

この章では、WHERE句を使って条件に合ったレコードを取得する方法を学びました：

1. 基本的な比較演算子（=, <>, >, <, >=, <=）の使い方
2. 数値による条件絞り込み
3. 文字列（テキスト）による条件絞り込み
4. 日付による条件絞り込み

WHERE句は、大量のデータから必要な情報だけを取り出すための非常に重要な機能です。実際のデータベース操作では、この条件絞り込みを頻繁に使います。

次の章では、複数の条件を組み合わせるための「論理演算子（AND、OR、NOT）」について学びます。

---

# 3. 論理演算子：AND、OR、NOTを使った複合条件

## はじめに

前章では、WHERE句を使って単一の条件でデータを絞り込む方法を学びました。しかし実際の業務では、より複雑な条件でデータを絞り込む必要があることがよくあります。

例えば：

- 「成績が80点以上**かつ**出席率が90%以上の学生」
- 「数学**または**英語の成績が優秀な学生」
- 「課題を**まだ提出していない**学生」

このような複合条件を指定するために使うのが論理演算子です。主な論理演算子は次の3つです：

- **AND**：両方の条件を満たす（かつ）
- **OR**：いずれかの条件を満たす（または）
- **NOT**：条件を満たさない（〜ではない）

## AND演算子

AND演算子は、指定した**すべての条件を満たす**レコードだけを取得したいときに使います。

> **用語解説**：
>
> - **AND**：「かつ」「そして」という意味の論理演算子です。複数の条件をすべて満たすデータを取得します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE 条件1 AND 条件2;
```

### 例：ANDを使った複合条件

例えば、中間テストで90点以上かつ満点が100点の成績レコードを取得するには：

```
SELECT * FROM grades
WHERE score >= 90 AND max_score = 100;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

この例では、「score >= 90」と「max_score = 100」の両方の条件を満たすレコードだけが取得されます。

### 3つ以上の条件の組み合わせ

ANDを使って3つ以上の条件を組み合わせることもできます：

```
SELECT * FROM grades
WHERE score >= 90 AND max_score = 100 AND grade_type = '中間テスト';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

## OR演算子

OR演算子は、指定した条件の**いずれか一つでも満たす**レコードを取得したいときに使います。

> **用語解説**：
>
> - **OR**：「または」「もしくは」という意味の論理演算子です。複数の条件のうち少なくとも1つを満たすデータを取得します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE 条件1 OR 条件2;
```

### 例：ORを使った複合条件

例えば、教師IDが101または102の講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id = 101 OR teacher_id = 102;
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 1 | ITのための基礎知識 | 101 |
| 2 | UNIX入門 | 102 |
| 3 | Cプログラミング演習 | 101 |
| 29 | コードリファクタリングとクリーンコード | 101 |
| 40 | ソフトウェアアーキテクチャパターン | 102 |
| … | … | … |

この例では、「teacher_id = 101」または「teacher_id = 102」のいずれかの条件を満たすレコードが取得されます。

## NOT演算子

NOT演算子は、指定した条件を**満たさない**レコードを取得したいときに使います。

> **用語解説**：
>
> - **NOT**：「～ではない」という意味の論理演算子です。条件を否定して、その条件を満たさないデータを取得します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE NOT 条件;
```

### 例：NOTを使った否定条件

例えば、完了（completed）状態ではない授業スケジュールを取得するには：

```
SELECT * FROM course_schedule
WHERE NOT status = 'completed';
```

実行結果：

| schedule_id | course_id | schedule_date | period_id | classroom_id | teacher_id | status |
|---|---|---|---|---|---|---|
| 45 | 11 | 2025-05-20 | 5 | 401G | 106 | scheduled |

| schedule_id | course_id | schedule_date | period_id | classroom_id | teacher_id | status |
|---|---|---|---|---|---|---|
| 46 | 12 | 2025-05-21 | 1 | 301E | 107 | scheduled |
| 50 | 2 | 2025-05-23 | 3 | 101A | 102 | cancelled |
| ... | ... | ... | ... | ... | ... | ... |

この例では、statusが「completed」ではないレコード（scheduled状態やcancelled状態）が取得されます。

NOT演算子は、「～ではない」という否定の条件を作るために使われます。例えば次の2つの書き方は同じ意味になります：

```
SELECT * FROM teachers WHERE NOT teacher_id = 101;
SELECT * FROM teachers WHERE teacher_id <> 101;
```

## 複合論理条件（AND、OR、NOTの組み合わせ）

AND、OR、NOTを組み合わせて、より複雑な条件を指定することもできます。

### 例：ANDとORの組み合わせ

例えば、「成績が90点以上で中間テストである」または「成績が45点以上でレポートである」レコードを取得するには：

```
SELECT * FROM grades
WHERE (score >= 90 AND grade_type = '中間テスト')
   OR (score >= 45 AND grade_type = 'レポート1');
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| 308 | 1 | レポート1 | 47.0 | 50.0 | 2025-05-09 |
| ... | ... | ... | ... | ... | ... |

## 優先順位と括弧の使用

論理演算子を組み合わせる場合、演算子の優先順位に注意が必要です。基本的に**ANDはORよりも優先順位が高い**です。つまり、ANDが先に処理されます。

例えば：

```
WHERE 条件1 OR 条件2 AND 条件3
```

これは次のように解釈されます：

```
WHERE 条件1 OR (条件2 AND 条件3)
```

意図した条件と異なる場合は、**括弧（）を使って明示的にグループ化**することが重要です：

```
WHERE (条件1 OR 条件2) AND 条件3
```

### 例：括弧を使った条件のグループ化

教師IDが101または102で、かつ、講座名に「プログラミング」という単語が含まれる講座を取得するには：

```sql
SELECT * FROM courses
WHERE (teacher_id = 101 OR teacher_id = 102)
  AND course_name LIKE '%プログラミング%';
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 3 | Cプログラミング演習 | 101 |
| … | … | … |

この例では、括弧を使って「teacher_id = 101 OR teacher_id = 102」の部分をグループ化し、その条件と「course_name LIKE '%プログラミング%'」の条件をANDで結合しています。

## 練習問題

### 問題3-1

grades（成績）テーブルから、課題タイプ（grade_type）が「中間テスト」かつ点数（score）が85点以上のレコードを取得するSQLを書いてください。

### 問題3-2

classrooms（教室）テーブルから、収容人数（capacity）が40人以下または建物（building）が「1号館」の教室を取得するSQLを書いてください。

### 問題3-3

teachers（教師）テーブルから、教師ID（teacher_id）が101、102、103ではない教師の情報を取得するSQLを書いてください。

### 問題3-4

course_schedule（授業カレンダー）テーブルから、2025年5月20日の授業で、時限（period_id）が1か2で、かつ状態（status）が「scheduled」の授業を取得するSQLを書いてください。

### 問題3-5

students（学生）テーブルから、学生名（student_name）に「田」または「山」を含む学生を取得するSQLを書いてください。

### 問題3-6

grades（成績）テーブルから、提出日（submission_date）が2025年5月15日以降で、かつ（「中間テスト」で90点以上または「レポート1」で45点以上）の成績を取得するSQLを書いてください。

## 解答

### 解答3-1

```
SELECT * FROM grades
WHERE grade_type = '中間テスト' AND score >= 85;
```

### 解答3-2

```
SELECT * FROM classrooms
WHERE capacity <= 40 OR building = '1号館';
```

### 解答3-3

```
SELECT * FROM teachers
WHERE NOT (teacher_id = 101 OR teacher_id = 102 OR teacher_id = 103);
```

または

```
SELECT * FROM teachers
WHERE teacher_id <> 101 AND teacher_id <> 102 AND teacher_id <> 103;
```

### 解答3-4

```sql
SELECT * FROM course_schedule
WHERE schedule_date = '2025-05-20'
  AND (period_id = 1 OR period_id = 2)
  AND status = 'scheduled';
```

解答3-5

```sql
SELECT * FROM students
WHERE student_name LIKE '%田%' OR student_name LIKE '%山%';
```

解答3-6

```sql
SELECT * FROM grades
WHERE submission_date >= '2025-05-15'
  AND ((grade_type = '中間テスト' AND score >= 90)
       OR (grade_type = 'レポート1' AND score >= 45));
```

## まとめ

この章では、論理演算子（AND、OR、NOT）を使って複合条件を作る方法を学びました：

1. **AND**：すべての条件を満たすレコードを取得（条件1 AND 条件2）
2. **OR**：いずれかの条件を満たすレコードを取得（条件1 OR 条件2）
3. **NOT**：指定した条件を満たさないレコードを取得（NOT 条件）
4. **複合条件**：AND、OR、NOTを組み合わせたより複雑な条件
5. **括弧（）**：条件をグループ化して優先順位を明示的に指定

これらの論理演算子を使いこなすことで、より複雑で細かな条件でデータを絞り込むことができるようになります。実際のデータベース操作では、複数の条件を組み合わせることが頻繁にあるため、この章で学んだ内容は非常に重要です。

次の章では、テキストデータに対する検索を行う「パターンマッチング」について学びます。

---

# 4. パターンマッチング：LIKE演算子と%、_ワイルドカード

---

## はじめに

前章までは、データの完全一致や数値の比較といった条件での絞り込みを学びました。しかし実際の業務では、もっと柔軟な検索が必要なケースがあります。例えば：

- 「山」で始まる名前の学生を検索したい
- 「プログラミング」という単語を含む講座名を探したい

- 電話番号の一部だけ覚えているデータを探したい

このような「部分一致」や「パターン一致」の検索を行うためのSQLの機能が「パターンマッチング」です。この章では、パターンマッチングを行うための「LIKE演算子」と「ワイルドカード文字」について学びます。

## LIKE演算子の基本

LIKE演算子は、文字列のパターンマッチングを行うための演算子です。WHERE句と組み合わせて使用します。

> **用語解説**：
>
> - **LIKE**：「～のような」という意味の演算子で、パターンに一致する文字列を検索します。
> - **パターンマッチング**：完全一致ではなく、一定のパターンに合致するデータを検索する方法です。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE 文字列カラム LIKE 'パターン';
```

パターンには、通常の文字に加えて、特別な意味を持つ「ワイルドカード文字」を使用できます。

## ワイルドカード文字

SQLでは主に2つのワイルドカード文字があります：

1. **%（パーセント）**：0文字以上の任意の文字列に一致します。
2. _**（アンダースコア）**：任意の1文字に一致します。

> **用語解説**：
>
> - **ワイルドカード**：任意の文字や文字列に一致する特殊な文字記号です。トランプのジョーカーのように、様々な値に代用できます。

## LIKE演算子の使い方：実践例

### 例1：%（パーセント）を使ったパターンマッチング

#### 「～で始まる」パターン：前方一致

例えば、「山」で始まる学生名を検索するには：

```
SELECT * FROM students WHERE student_name LIKE '山%';
```

実行結果：

| student_id | student_name |
|---|---|
| 312 | 山本裕子 |
| 325 | 山田翔太 |
| … | … |

ここでの「山%」は「山で始まり、その後に0文字以上の任意の文字が続く」という意味です。

**「〜で終わる」パターン：後方一致**

例えば、「子」で終わる教師名を検索するには：

```
SELECT * FROM teachers WHERE teacher_name LIKE '%子';
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 102 | 田尻朋美 |
| 106 | 星野涼子 |
| 108 | 吉岡由佳 |
| 110 | 佐藤花子 |
| … | … |

**「〜を含む」パターン：部分一致**

例えば、「プログラミング」という単語を含む講座名を検索するには：

```
SELECT * FROM courses WHERE course_name LIKE '%プログラミング%';
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 3 | Cプログラミング演習 | 101 |
| 14 | IoTデバイスプログラミング実践 | 110 |
| … | … | … |

### 例2：_（アンダースコア）を使ったパターンマッチング

アンダースコアは任意の1文字に一致します。例えば、教室IDが「10_A」パターン（最初の2文字が「10」、3文字目が任意の1文字、最後が「A」）の教室を検索するには：

```
SELECT * FROM classrooms WHERE classroom_id LIKE '10_A';
```

実行結果：

| classroom_id | classroom_name | capacity | building | facilities |
|---|---|---|---|---|
| 101A | 1号館コンピュータ実習室A | 30 | 1号館 | パソコン30台、プロジェクター |
| … | … | … | … | … |

### 例3：%と_の組み合わせ

ワイルドカード文字は組み合わせて使うこともできます。例えば、「2文字目が田」の学生を検索するには：

```
SELECT * FROM students WHERE student_name LIKE '_田%';
```

実行結果：

| student_id | student_name |
|---|---|
| 321 | 井上竜也 |
| 384 | 櫻井翼 |
| … | … |

## NOT LIKEを使った否定形のパターンマッチング

特定のパターンに一致しないレコードを検索したい場合は、「NOT LIKE」を使います。

> **用語解説**：
>
> - **NOT LIKE**：「～のパターンに一致しない」という意味で、指定したパターンに一致しないデータを検索します。

例えば、講座名に「入門」を含まない講座を検索するには：

```
SELECT * FROM courses WHERE course_name NOT LIKE '%入門%';
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 1 | ITのための基礎知識 | 101 |
| 3 | Cプログラミング演習 | 101 |
| … | … | … |

## エスケープ文字の使用

もし検索したいパターンに「%」や「_」自体が含まれている場合は、それらを特別な文字としてではなく、通常の文字として扱うために「エスケープ文字」を使います。

> **用語解説**：
>
> - **エスケープ文字**：特別な意味を持つ文字を通常の文字として扱うための印です。

MySQLでは、バックスラッシュ（\）をエスケープ文字として使用できます。例えば、「50%」という値そのものを検索するには：

```
SELECT * FROM テーブル名 WHERE カラム LIKE '50\%';
```

または、ESCAPE句を使って明示的にエスケープ文字を指定することもできます：

```
SELECT * FROM テーブル名 WHERE カラム LIKE '50!%' ESCAPE '!';
```

この例では「!」をエスケープ文字として指定しています。

## 大文字と小文字の区別

MySQLのデフォルト設定では、LIKE演算子は大文字と小文字を区別しません（大文字小文字を同じものとして扱います）。

例えば、次の2つのクエリは同じ結果を返します：

```
SELECT * FROM courses WHERE course_name LIKE '%web%';
SELECT * FROM courses WHERE course_name LIKE '%Web%';
```

もし大文字と小文字を区別した検索が必要な場合は、「BINARY」キーワードを使用します：

```
SELECT * FROM courses WHERE course_name LIKE BINARY '%Web%';
```

この場合、「Web」は「web」とは一致しません。

## 複合条件との組み合わせ

LIKE演算子は、これまで学んだAND、OR、NOTなどの論理演算子と組み合わせて使うこともできます。

例えば、「田」で始まる名前で、かつ教師IDが102から105の間の教師を検索するには：

```
SELECT * FROM teachers
WHERE teacher_name LIKE '田%'
```

```
    AND teacher_id BETWEEN 102 AND 105;
```

実行結果：

| teacher_id | teacher_name |
|---|---|
| 102 | 田尻朋美 |
| ... | ... |

## 練習問題

### 問題4-1

students（学生）テーブルから、学生名（student_name）が「佐藤」で始まる学生の情報をすべて取得するSQLを書いてください。

### 問題4-2

courses（講座）テーブルから、講座名（course_name）に「データ」という単語を含む講座の情報を取得するSQLを書いてください。

### 問題4-3

classrooms（教室）テーブルから、教室名（classroom_name）が「コンピュータ実習室」で終わる教室の情報を取得するSQLを書いてください。

### 問題4-4

teachers（教師）テーブルから、教師名（teacher_name）の2文字目が「木」である教師の情報を取得するSQLを書いてください。

### 問題4-5

courses（講座）テーブルから、講座名（course_name）に「入門」または「基礎」を含む講座を取得するSQLを書いてください。

### 問題4-6

students（学生）テーブルから、学生名（student_name）が「山」で始まり、かつ「子」で終わらない学生を取得するSQLを書いてください。

## 解答

### 解答4-1

```
SELECT * FROM students WHERE student_name LIKE '佐藤%';
```

### 解答4-2

```sql
SELECT * FROM courses WHERE course_name LIKE '%データ%';
```

解答4-3

```sql
SELECT * FROM classrooms WHERE classroom_name LIKE '%コンピュータ実習室';
```

解答4-4

```sql
SELECT * FROM teachers WHERE teacher_name LIKE '_木%';
```

解答4-5

```sql
SELECT * FROM courses
WHERE course_name LIKE '%入門%' OR course_name LIKE '%基礎%';
```

解答4-6

```sql
SELECT * FROM students
WHERE student_name LIKE '山%' AND student_name NOT LIKE '%子';
```

## まとめ

この章では、パターンマッチングを行うためのLIKE演算子と、その中で使用するワイルドカード文字（%と_）について学びました：

1. **LIKE演算子**：文字列パターンに一致するデータを検索するための演算子
2. **%（パーセント）**：0文字以上の任意の文字列に一致するワイルドカード
3. **_（アンダースコア）**：任意の1文字に一致するワイルドカード
4. **前方一致**：「パターン%」で「パターンで始まる」文字列に一致
5. **後方一致**：「%パターン」で「パターンで終わる」文字列に一致
6. **部分一致**：「%パターン%」で「パターンを含む」文字列に一致
7. **NOT LIKE**：指定したパターンに一致しないデータを検索
8. **エスケープ文字**：特殊文字（%や_）を通常の文字として扱うための方法
9. **複合条件との組み合わせ**：AND、ORなどと組み合わせたより複雑な条件

パターンマッチングは、特にテキストデータを扱う際に非常に便利な機能です。部分的な情報しか持っていない場合や、特定のパターンを持つデータを探す場合に活用できます。

次の章では、範囲指定のための「BETWEEN演算子」と「IN演算子」について学びます。

# 5. 範囲指定：BETWEEN、IN演算子

## はじめに

これまでの章では、等号（=）や不等号（>、<）を使って条件を指定する方法や、LIKE演算子を使ったパターンマッチングを学びました。この章では、値の範囲を指定する「BETWEEN演算子」と、複数の値を一度に指定できる「IN演算子」について学びます。

これらの演算子を使うと、次のような検索がより簡単になります：

- 「80点から90点の間の成績」
- 「2025年4月から6月の間のスケジュール」
- 「特定の教師IDリストに該当する講座」

## BETWEEN演算子：範囲を指定する

BETWEEN演算子は、ある値が指定した範囲内にあるかどうかを調べるために使います。

> **用語解説**：
>
> - **BETWEEN**：「～の間に」という意味の演算子で、ある値が指定した最小値と最大値の間（両端の値を含む）にあるかどうかを判定します。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 BETWEEN 最小値 AND 最大値;
```

この構文は次の条件と同じ意味です：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 >= 最小値 AND カラム名 <= 最大値;
```

### 例1：数値範囲の指定

例えば、成績（grades）テーブルから、80点から90点の間の成績を取得するには：

```
SELECT * FROM grades
WHERE score BETWEEN 80 AND 90;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 308 | 6 | 小テスト1 | 89.0 | 100.0 | 2025-05-15 |
| ... | ... | ... | ... | ... | ... |

### 例2：日付範囲の指定

日付にもBETWEEN演算子が使えます。例えば、2025年5月10日から2025年5月20日までに提出された成績を取得するには：

```
SELECT * FROM grades
WHERE submission_date BETWEEN '2025-05-10' AND '2025-05-20';
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 1 | レポート1 | 45.0 | 50.0 | 2025-05-10 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| 301 | 1 | 中間テスト | 85.5 | 100.0 | 2025-05-20 |
| ... | ... | ... | ... | ... | ... |

## NOT BETWEEN：範囲外を指定する

NOT BETWEENを使うと、指定した範囲の外にある値を持つレコードを取得できます。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 NOT BETWEEN 最小値 AND 最大値;
```

これは次の条件と同じです：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 < 最小値 OR カラム名 > 最大値;
```

### 例：範囲外の値を取得

例えば、80点未満または90点より高い成績を取得するには：

```
SELECT * FROM grades
WHERE score NOT BETWEEN 80 AND 90;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| 303 | 1 | 中間テスト | 78.5 | 100.0 | 2025-05-20 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| ... | ... | ... | ... | ... | ... |

## IN演算子：複数の値を指定する

IN演算子は、ある値が指定した複数の値のリストのいずれかに一致するかどうかを調べるために使います。

> **用語解説**：
>
> - **IN**：「～の中に含まれる」という意味の演算子で、ある値が指定したリストの中に含まれているかどうかを判定します。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 IN (値1, 値2, 値3, ...);
```

この構文は次のOR条件の組み合わせと同じ意味です：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 = 値1 OR カラム名 = 値2 OR カラム名 = 値3 OR ...;
```

### 例1：数値リストの指定

例えば、教師ID（teacher_id）が101、103、105のいずれかである講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id IN (101, 103, 105);
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 1 | ITのための基礎知識 | 101 |
| 3 | Cプログラミング演習 | 101 |

| course_id | course_name | teacher_id |
|---|---|---|
| 5 | データベース設計と実装 | 105 |
| 10 | プロジェクト管理手法 | 103 |
| ... | ... | ... |

### 例2：文字列リストの指定

文字列のリストにも適用できます。例えば、特定の教室ID（classroom_id）のみの教室情報を取得するには：

```
SELECT * FROM classrooms
WHERE classroom_id IN ('101A', '202D', '301E');
```

実行結果：

| classroom_id | classroom_name | capacity | building | facilities |
|---|---|---|---|---|
| 101A | 1号館コンピュータ実習室A | 30 | 1号館 | パソコン30台、プロジェクター |
| 202D | 2号館コンピュータ実習室D | 25 | 2号館 | パソコン25台、プロジェクター、3Dプリンター |
| 301E | 3号館講義室E | 80 | 3号館 | プロジェクター、マイク設備、録画設備 |

## NOT IN：リストに含まれない値を指定する

NOT IN演算子を使うと、指定したリストに含まれない値を持つレコードを取得できます。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 NOT IN (値1, 値2, 値3, ...);
```

これは次の条件と同じです：

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 <> 値1 AND カラム名 <> 値2 AND カラム名 <> 値3 AND ...;
```

### 例：リストに含まれない値を取得

例えば、教師IDが101、102、103以外の教師が担当する講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id NOT IN (101, 102, 103);
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 4 | Webアプリケーション開発 | 104 |
| 5 | データベース設計と実装 | 105 |
| 6 | ネットワークセキュリティ | 107 |
| … | … | … |

## IN演算子でのサブクエリの利用（基本）

IN演算子の括弧内には、直接値を書く代わりに、サブクエリ（別のSELECT文）を指定することもできます。これにより、動的に値のリストを生成できます。

> **用語解説**：
>
> - **サブクエリ**：SQL文の中に含まれる別のSQL文のことで、外側のSQL文（メインクエリ）に値や条件を提供します。

### 基本構文

```
SELECT カラム名 FROM テーブル名
WHERE カラム名 IN (SELECT カラム名 FROM 別テーブル WHERE 条件);
```

### 例：サブクエリを使ったIN条件

例えば、教師名（teacher_name）に「田」を含む教師が担当している講座を取得するには：

```
SELECT * FROM courses
WHERE teacher_id IN (
    SELECT teacher_id FROM teachers
    WHERE teacher_name LIKE '%田%'
);
```

実行結果：

| course_id | course_name | teacher_id |
|---|---|---|
| 2 | UNIX入門 | 102 |
| 10 | プロジェクト管理手法 | 103 |

| course_id | course_name | teacher_id |
|---|---|---|
| ... | ... | ... |

この例では、まず「teachers」テーブルから名前に「田」を含む教師のIDを取得し、それらのIDを持つ講座を「courses」テーブルから取得しています。

## BETWEEN演算子とIN演算子の組み合わせ

BETWEEN演算子とIN演算子は、論理演算子（AND、OR）と組み合わせて、さらに複雑な条件を作ることができます。

### 例：BETWEENとINの組み合わせ

例えば、「教師IDが101、103、105のいずれかで、かつ、2025年5月15日から2025年6月15日の間に実施される授業」を取得するには：

```
SELECT * FROM course_schedule
WHERE teacher_id IN (101, 103, 105)
  AND schedule_date BETWEEN '2025-05-15' AND '2025-06-15';
```

実行結果：

| schedule_id | course_id | schedule_date | period_id | classroom_id | teacher_id | status |
|---|---|---|---|---|---|---|
| 38 | 1 | 2025-05-20 | 1 | 102B | 101 | scheduled |
| 42 | 10 | 2025-05-23 | 3 | 201C | 103 | scheduled |
| 45 | 5 | 2025-05-28 | 2 | 402H | 105 | scheduled |
| ... | ... | ... | ... | ... | ... | ... |

## 練習問題

### 問題5-1

grades（成績）テーブルから、得点（score）が85点から95点の間にある成績を取得するSQLを書いてください。

### 問題5-2

course_schedule（授業カレンダー）テーブルから、2025年5月1日から2025年5月31日までの授業スケジュールを取得するSQLを書いてください。

### 問題5-3

courses（講座）テーブルから、教師ID（teacher_id）が104、106、108のいずれかである講座の情報を取得するSQLを書いてください。

### 問題5-4

classrooms（教室）テーブルから、教室ID（classroom_id）が「101A」、「201C」、「301E」、「401G」以外の教室情報を取得するSQLを書いてください。

### 問題5-5

grades（成績）テーブルから、評価タイプ（grade_type）が「中間テスト」または「実技試験」で、かつ得点（score）が80点から90点の間にない成績を取得するSQLを書いてください。

### 問題5-6

course_schedule（授業カレンダー）テーブルから、教室ID（classroom_id）が「101A」、「202D」のいずれかで、かつ2025年5月15日から2025年5月30日の間に実施される授業スケジュールを取得するSQLを書いてください。

## 解答

### 解答5-1

```
SELECT * FROM grades
WHERE score BETWEEN 85 AND 95;
```

### 解答5-2

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

### 解答5-3

```
SELECT * FROM courses
WHERE teacher_id IN (104, 106, 108);
```

### 解答5-4

```
SELECT * FROM classrooms
WHERE classroom_id NOT IN ('101A', '201C', '301E', '401G');
```

### 解答5-5

```
SELECT * FROM grades
WHERE grade_type IN ('中間テスト', '実技試験')
```

```
  AND score NOT BETWEEN 80 AND 90;
```

解答5-6

```
SELECT * FROM course_schedule
WHERE classroom_id IN ('101A', '202D')
  AND schedule_date BETWEEN '2025-05-15' AND '2025-05-30';
```

## まとめ

この章では、範囲指定のための「BETWEEN演算子」と複数値指定のための「IN演算子」について学びました：

1. **BETWEEN演算子**：値が指定した範囲内（両端を含む）にあるかどうかをチェック
2. **NOT BETWEEN**：値が指定した範囲外にあるかどうかをチェック
3. **IN演算子**：値が指定したリストのいずれかに一致するかどうかをチェック
4. **NOT IN**：値が指定したリストのいずれにも一致しないかどうかをチェック
5. **サブクエリとIN**：動的に生成された値のリストを使用する方法
6. **複合条件**：BETWEEN、IN、論理演算子を組み合わせたより複雑な条件

これらの演算子を使うことで、複数の条件を指定する場合に、SQLをより簡潔に書くことができます。特に、多くの値を指定する場合や範囲条件を指定する場合に便利です。

次の章では、「NULL値の処理：IS NULL、IS NOT NULL」について学びます。

---

# 6. NULL値の処理：IS NULL、IS NOT NULL

## はじめに

データベースの世界では、データがない状態を表すために「NULL」という特別な値が使われます。NULLは「空」や「0」や「空白文字」とは異なる、「値が存在しない」または「不明」であることを表す特殊な概念です。

例えば、学校データベースでは次のようなシナリオがあります：

- まだ成績が付けられていない（NULL）
- コメントが入力されていない（NULL）
- 授業がキャンセルされたため教室が割り当てられていない（NULL）

この章では、NULL値を正しく処理するための「IS NULL」と「IS NOT NULL」演算子について学びます。

## NULLとは何か？

NULL値には、いくつかの特徴があります：

1. **値がない**：NULLは値がないことを表します。0でも空文字列（''）でもなく、値そのものが存在しないことを示します。
2. **不明**：データが不明であることを表す場合もあります。
3. **未設定**：まだ値が設定されていないことを表す場合もあります。
4. **比較できない**：NULLは通常の比較演算子（=, <, >など）で比較できません。

> **用語解説**：
>
> - **NULL**：データベースにおいて「値がない」または「不明」を表す特殊な値です。0や空文字とは異なります。

## NULL値と通常の比較演算子

通常の比較演算子（=, <>, >, <, >=, <=）では、NULL値を正しく検出できません。例えば：

```
-- この条件はNULL値に対して常にFALSEを返す
SELECT * FROM テーブル名 WHERE カラム名 = NULL;

-- この条件もNULL値に対して常にFALSEを返す
SELECT * FROM テーブル名 WHERE カラム名 <> NULL;
```

これは、NULL値との等価比較は「不明」と評価されるためです。つまり、NULL = NULLでさえFALSEではなく「不明」になります。

## IS NULL演算子

NULL値を持つレコードを検索するには、「IS NULL」演算子を使います。

> **用語解説**：
>
> - **IS NULL**：カラムの値がNULLかどうかを調べる演算子です。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE カラム名 IS NULL;
```

### 例：IS NULLの使用

例えば、コメントが入力されていない（NULL）出席レコードを検索するには：

```
SELECT * FROM attendance WHERE comment IS NULL;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 301 | present | NULL |
| 1 | 306 | present | NULL |
| 1 | 307 | present | NULL |
| ... | ... | ... | NULL |

## IS NOT NULL演算子

逆に、NULL値を持たないレコード（つまり、何らかの値を持つレコード）を検索するには、「IS NOT NULL」演算子を使います。

> **用語解説**：
>
> - **IS NOT NULL**：カラムの値がNULLでないかどうかを調べる演算子です。

### 基本構文

```
SELECT カラム名 FROM テーブル名 WHERE カラム名 IS NOT NULL;
```

### 例：IS NOT NULLの使用

例えば、コメントが入力されている（NOT NULL）出席レコードを検索するには：

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

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 302 | late | 15分遅刻 |
| 1 | 303 | absent | 事前連絡あり |
| 1 | 308 | late | 5分遅刻 |
| ... | ... | ... | ... |

## NULL値の論理的な扱い

NULL値は論理演算（AND、OR、NOT）でも特殊な扱いを受けます。

- **NULL AND TRUE** → NULL（不明）
- **NULL AND FALSE** → FALSE
- **NULL OR TRUE** → TRUE
- **NULL OR FALSE** → NULL（不明）

- **NOT NULL** → NULL（不明）

この特殊な振る舞いが、バグや誤った結果の原因になることがあります。

## NULL値と結合条件

テーブル結合（JOINなど、後の章で学習）の際も、NULL値は特殊な扱いを受けます。NULL値同士は「等しい」とは判定されないため、通常の結合条件ではNULL値を持つレコードは結合されません。

## IS NULLとIS NOT NULLを使った複合条件

IS NULLとIS NOT NULLも、他の条件と組み合わせて使用できます。

### 例：複合条件でのIS NULLの使用

例えば、「出席状態が "absent"（欠席）で、コメントがNULLでない（理由が入力されている）レコード」を検索するには：

```
SELECT * FROM attendance
WHERE status = 'absent' AND comment IS NOT NULL;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 303 | absent | 事前連絡あり |
| 1 | 317 | absent | 体調不良 |
| … | … | … | … |

## NVL/IFNULL/COALESCE関数：NULL値の置換

NULL値を別の値に置き換えるための関数が用意されています。データベースによって関数名が異なることがありますが、機能は似ています：

- MySQL/MariaDB: **IFNULL(expr, replace_value)**
- Oracle: **NVL(expr, replace_value)**
- SQL Server: **ISNULL(expr, replace_value)**
- 標準SQL: **COALESCE(expr1, expr2, ..., exprN)** - 最初のNULLでない式を返します

### 例：IFNULL関数の使用（MySQL）

例えば、コメントがNULLの場合は「特記事項なし」と表示するには：

```
SELECT schedule_id, student_id, status,
       IFNULL(comment, '特記事項なし') AS comment
FROM attendance;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 301 | present | 特記事項なし |
| 1 | 302 | late | 15分遅刻 |
| 1 | 303 | absent | 事前連絡あり |
| ... | ... | ... | ... |

## NULLを使う際の注意点

1. **除外の罠**：`WHERE カラム名 <> 値` だけでは、NULL値を持つレコードは含まれません。すべてのレコードを対象にするには：

   ```
   WHERE カラム名 <> 値 OR カラム名 IS NULL
   ```

2. **集計関数**：COUNT(*)はすべての行を数えますが、COUNT(カラム名)はそのカラムがNULLでない行だけを数えます。

3. **インデックス**：多くのデータベースでは、NULL値にもインデックスを適用できますが、データベースによって動作が異なる場合があります。

4. **一意性制約**：一般的に、UNIQUE制約ではNULL値は重複としてカウントされません（複数のNULL値が許可されます）。

## 練習問題

### 問題6-1

attendance（出席）テーブルから、コメント（comment）がNULLの出席レコードをすべて取得するSQLを書いてください。

### 問題6-2

course_schedule（授業カレンダー）テーブルから、状態（status）が「cancelled」で、かつ教室ID（classroom_id）がNULLでないレコードを取得するSQLを書いてください。

### 問題6-3

grades（成績）テーブルから、提出日（submission_date）がNULLの成績レコードを取得するSQLを書いてください。

### 問題6-4

attendance（出席）テーブルから、出席状態（status）が「present」か「late」で、かつコメント（comment）がNULLのレコードを取得するSQLを書いてください。

### 問題6-5

以下のSQLで教師（teachers）テーブルから「佐藤」という名前を持つ教師を検索する場合、NULL値を持つレコードも含めるにはどう修正すべきですか？

```
SELECT * FROM teachers WHERE teacher_name <> '佐藤花子';
```

### 問題6-6

attendance（出席）テーブルのすべてのレコードを取得し、コメント（comment）がNULLの場合は「記録なし」と表示するSQLを書いてください。

## 解答

### 解答6-1

```
SELECT * FROM attendance WHERE comment IS NULL;
```

### 解答6-2

```
SELECT * FROM course_schedule
WHERE status = 'cancelled' AND classroom_id IS NOT NULL;
```

### 解答6-3

```
SELECT * FROM grades WHERE submission_date IS NULL;
```

### 解答6-4

```
SELECT * FROM attendance
WHERE (status = 'present' OR status = 'late') AND comment IS NULL;
```

または

```
SELECT * FROM attendance
WHERE status IN ('present', 'late') AND comment IS NULL;
```

### 解答6-5

```
SELECT * FROM teachers
WHERE teacher_name <> '佐藤花子' OR teacher_name IS NULL;
```

解答6-6

```
SELECT schedule_id, student_id, status,
       IFNULL(comment, '記録なし') AS comment
FROM attendance;
```

## まとめ

この章では、データベースにおけるNULL値の概念と、NULL値を扱うための演算子や関数について学びました：

1. **NULL値の概念**：値がない、不明、未設定を表す特殊な値
2. **IS NULL演算子**：NULL値を持つレコードを検索する方法
3. **IS NOT NULL演算子**：NULL値を持たないレコードを検索する方法
4. **NULL値の論理的扱い**：論理演算（AND、OR、NOT）におけるNULLの振る舞い
5. **複合条件**：IS NULL/IS NOT NULLと他の条件の組み合わせ
6. **NULL値の置換**：IFNULL/NVL/COALESCE関数の使用方法
7. **注意点**：NULL値を扱う際の一般的な落とし穴

NULL値の正確な理解と適切な処理は、SQLプログラミングの重要な部分です。不適切なNULL処理は、予期しない結果やバグの原因になります。

次の章では、クエリ結果の並び替えを行うための「ORDER BY：結果の並び替え」について学びます。

---

# 7. ORDER BY：結果の並び替え

## はじめに

これまでの章では、データベースから条件に合ったレコードを取得する方法を学んできました。しかし実際の業務では、取得したデータを見やすく整理する必要があります。例えば：

- 成績を高い順に表示したい
- 学生を名前の五十音順に並べたい
- 日付の新しい順にスケジュールを確認したい

このようなデータの「並び替え」を行うためのSQLコマンドが「ORDER BY」です。この章では、クエリ結果を特定の順序で並べる方法を学びます。

## ORDER BYの基本

ORDER BY句は、SELECT文の結果を指定したカラムの値に基づいて並び替えるために使います。

> **用語解説：**
>
> - **ORDER BY**：「〜の順に並べる」という意味のSQLコマンドで、クエリ結果の並び順を指定します。

### 基本構文

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] ORDER BY 並び替えカラム;
```

ORDER BY句は通常、SELECT文の最後に記述します。

### 例1：単一カラムでの並び替え

例えば、学生（students）テーブルから、学生名（student_name）の五十音順（辞書順）でデータを取得するには：

```
SELECT * FROM students ORDER BY student_name;
```

実行結果：

| student_id | student_name |
|---|---|
| 309 | 相沢吉夫 |
| 303 | 柴崎春花 |
| 306 | 河田咲奈 |
| 305 | 河口菜恵子 |
| … | … |

### デフォルトの並び順

ORDER BYを使わない場合、結果の順序は保証されません。多くの場合、データがデータベースに保存された順序で返されますが、これは信頼できるものではありません。

## 昇順と降順の指定

ORDER BY句では、並び順を「昇順」か「降順」のどちらかで指定できます。

> **用語解説：**
>
> - **昇順（ASC）**：小さい値から大きい値へ（A→Z、1→9）の順に並べます。
> - **降順（DESC）**：大きい値から小さい値へ（Z→A、9→1）の順に並べます。

### 構文

```
SELECT カラム名 FROM テーブル名 ORDER BY 並び替えカラム [ASC|DESC];
```

ASC（昇順）がデフォルトのため、省略可能です。

### 例2：降順での並び替え

例えば、成績（grades）テーブルから、得点（score）の高い順（降順）に成績を取得するには：

```
SELECT * FROM grades ORDER BY score DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |

## 複数カラムでの並び替え

複数のカラムを使って並び替えることもできます。最初に指定したカラムで並び替え、値が同じレコードがある場合は次のカラムで並び替えます。

### 構文

```
SELECT カラム名 FROM テーブル名
ORDER BY 並び替えカラム1 [ASC|DESC], 並び替えカラム2 [ASC|DESC], ...;
```

### 例3：複数カラムでの並び替え

例えば、成績（grades）テーブルから、課題タイプ（grade_type）の五十音順に並べ、同じ課題タイプ内では得点（score）の高い順に成績を取得するには：

```
SELECT * FROM grades
ORDER BY grade_type ASC, score DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 301 | 2 | 実技試験 | 88.0 | 100.0 | 2025-05-18 |

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 321 | 2 | 実技試験 | 85.5 | 100.0 | 2025-05-18 |
| … | … | … | … | … | … |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |
| 311 | 1 | レポート1 | 49.0 | 50.0 | 2025-05-08 |
| 302 | 1 | レポート1 | 48.0 | 50.0 | 2025-05-10 |
| … | … | … | … | … | … |

### 例4：昇順と降順の混合

各カラムごとに並び順を指定することもできます。例えば、講座ID（course_id）の昇順、評価タイプ（grade_type）の昇順、得点（score）の降順で並べるには：

```
SELECT * FROM grades
ORDER BY course_id ASC, grade_type ASC, score DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 311 | 1 | レポート1 | 49.0 | 50.0 | 2025-05-08 |
| … | … | … | … | … | … |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| … | … | … | … | … | … |
| 301 | 2 | 実技試験 | 88.0 | 100.0 | 2025-05-18 |
| … | … | … | … | … | … |

## NULLの扱い

ORDER BYでNULL値を並び替える場合、データベース製品によって動作が異なります。多くのデータベースでは、NULL値は最小値または最大値として扱われます。

- MySQL/MariaDBでは、NULL値は昇順（ASC）の場合は最小値として（最初に表示）、降順（DESC）の場合は最大値として（最後に表示）扱われます。

一部のデータベース（PostgreSQLなど）では、NULL値の位置を明示的に指定するための「NULLS FIRST」「NULLS LAST」構文がサポートされています。

### 例5：NULL値の扱い

例えば、出席（attendance）テーブルからコメント（comment）でソートすると、NULLが最初に来ます：

```
SELECT * FROM attendance ORDER BY comment;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 301 | present | NULL |
| 1 | 306 | present | NULL |
| ... | ... | ... | NULL |
| 1 | 308 | late | 5分遅刻 |
| 1 | 323 | late | 電車遅延 |
| ... | ... | ... | ... |

## カラム番号を使った並び替え

カラム名の代わりに、SELECT文の結果セットにおけるカラムの位置（番号）を使って並び替えることもできます。最初のカラムは1、2番目のカラムは2、という具合です。

### 構文

```
SELECT カラム名1, カラム名2, ... FROM テーブル名 ORDER BY カラム位置;
```

### 例6：カラム番号を使った並び替え

例えば、学生（students）テーブルから学生ID（student_id）と名前（student_name）を取得し、名前（2番目のカラム）で並べ替えるには：

```
SELECT student_id, student_name FROM students ORDER BY 2;
```

この場合、「ORDER BY 2」は「ORDER BY student_name」と同じ意味になります。

> **注意**：カラム番号を使う方法は、カラムの順序を変更すると問題が起きるため、実際の業務では使用を避けた方が良いとされています。

## 式や関数を使った並び替え

ORDER BY句で式や関数を使うことにより、計算結果に基づいて並び替えることもできます。

### 例7：式を使った並び替え

例えば、成績（grades）テーブルから、得点の達成率（score/max_score）の高い順に並べるには：

```
SELECT student_id, course_id, grade_type, score, max_score,
       (score/max_score)*100 AS 達成率
FROM grades
ORDER BY (score/max_score) DESC;
```

または

```
SELECT student_id, course_id, grade_type, score, max_score,
       (score/max_score)*100 AS 達成率
FROM grades
ORDER BY 達成率 DESC;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | 達成率 |
|---|---|---|---|---|---|
| 311 | 1 | レポート1 | 49.0 | 50.0 | 98.0 |
| 320 | 1 | レポート1 | 48.5 | 50.0 | 97.0 |
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 95.0 |
| … | … | … | … | … | … |

### 例8：関数を使った並び替え

文字列関数を使って並び替えることもできます。例えば、月名で並べることを考えましょう：

```
SELECT schedule_date, MONTH(schedule_date) AS month
FROM course_schedule
ORDER BY MONTH(schedule_date);
```

実行結果：

| schedule_date | month |
|---|---|
| 2025-04-07 | 4 |
| 2025-04-08 | 4 |
| … | … |
| 2025-05-01 | 5 |

| schedule_date | month |
|---|---|
| 2025-05-02 | 5 |
| … | … |
| 2025-06-01 | 6 |
| … | … |

## CASE式を使った条件付き並び替え

さらに高度な並び替えとして、CASE式を使って条件に応じた並び順を定義することもできます。

### 例9：CASE式を使った並び替え

例えば、出席（attendance）テーブルから、出席状況（status）を「欠席→遅刻→出席」の順に優先して表示するには：

```
SELECT * FROM attendance
ORDER BY CASE
           WHEN status = 'absent' THEN 1
           WHEN status = 'late' THEN 2
           WHEN status = 'present' THEN 3
           ELSE 4
         END;
```

実行結果：

| schedule_id | student_id | status | comment |
|---|---|---|---|
| 1 | 303 | absent | 事前連絡あり |
| 1 | 317 | absent | 体調不良 |
| … | … | … | … |
| 1 | 302 | late | 15分遅刻 |
| 1 | 308 | late | 5分遅刻 |
| … | … | … | … |
| 1 | 301 | present | NULL |
| 1 | 306 | present | NULL |
| … | … | … | … |

## 練習問題

### 問題7-1

students（学生）テーブルから、すべての学生情報を学生名（student_name）の降順（逆五十音順）で取得するSQLを書いてください。

### 問題7-2

grades（成績）テーブルから、得点（score）が85点以上の成績を得点の高い順に取得するSQLを書いてください。

### 問題7-3

course_schedule（授業カレンダー）テーブルから、2025年5月の授業スケジュールを日付（schedule_date）の昇順で取得するSQLを書いてください。

### 問題7-4

teachers（教師）テーブルから、教師IDと名前を取得し、名前（teacher_name）の五十音順で並べるSQLを書いてください。

### 問題7-5

grades（成績）テーブルから、講座ID（course_id）ごとに、成績を評価タイプ（grade_type）の五十音順に、同じ評価タイプ内では得点（score）の高い順に並べて取得するSQLを書いてください。

### 問題7-6

attendance（出席）テーブルから、すべての出席情報を出席状況（status）が「absent」「late」「present」の順番で、同じ状態内ではコメント（comment）の有無（NULLが後）で並べて取得するSQLを書いてください。

## 解答

### 解答7-1

```sql
SELECT * FROM students ORDER BY student_name DESC;
```

### 解答7-2

```sql
SELECT * FROM grades WHERE score >= 85 ORDER BY score DESC;
```

### 解答7-3

```sql
SELECT * FROM course_schedule
WHERE schedule_date BETWEEN '2025-05-01' AND '2025-05-31'
ORDER BY schedule_date;
```

解答7-4

```
SELECT teacher_id, teacher_name FROM teachers ORDER BY teacher_name;
```

解答7-5

```
SELECT * FROM grades
ORDER BY course_id, grade_type, score DESC;
```

解答7-6

```
SELECT * FROM attendance
ORDER BY
  CASE
    WHEN status = 'absent' THEN 1
    WHEN status = 'late' THEN 2
    WHEN status = 'present' THEN 3
    ELSE 4
  END,
  CASE
    WHEN comment IS NULL THEN 2
    ELSE 1
  END;
```

## まとめ

この章では、クエリ結果を特定の順序で並べるための「ORDER BY」句について学びました：

1. **基本的な並び替え**：指定したカラムの値に基づいて結果を並べる方法
2. **昇順と降順**：ASC（昇順）とDESC（降順）の指定方法
3. **複数カラムでの並び替え**：優先順位の高いカラムから順に指定する方法
4. **NULL値の扱い**：NULL値が並び替えでどのように扱われるか
5. **カラム番号**：カラム名の代わりに位置で指定する方法（あまり推奨されない）
6. **式や関数**：計算結果に基づいて並べる方法
7. **CASE式**：条件付きの複雑な並び替え

ORDER BY句は、データを見やすく整理するために非常に重要です。特に大量のデータを扱う場合、適切な並び順はデータの理解を大きく助けます。

次の章では、取得する結果の件数を制限する「LIMIT句：結果件数の制限とページネーション」について学びます。

---

# 8. LIMIT句：結果件数の制限とページネーション

# はじめに

これまでの章では、条件に合うデータを取得し、それを特定の順序で並べる方法を学びました。しかし実際のアプリケーションでは、大量のデータがあるときに、その一部だけを表示したいことがよくあります。例えば：

- 成績上位10件だけを表示したい
- Webページで一度に20件ずつ表示したい（ページネーション）
- 最新の5件のお知らせだけを取得したい

このような「結果の件数を制限する」ためのSQLコマンドが「LIMIT句」です。この章では、クエリ結果の件数を制限する方法と、ページネーションの実装方法を学びます。

# LIMIT句の基本

LIMIT句は、SELECT文の結果から指定した件数だけを取得するために使います。

> **用語解説**：
>
> - **LIMIT**：「制限する」という意味のSQLコマンドで、取得する行数を制限します。

## 基本構文（MySQL/MariaDB）

MySQLやMariaDBでのLIMIT句の基本構文は次のとおりです：

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] [ORDER BY 並び順] LIMIT 件数;
```

LIMIT句は通常、SELECT文の最後に記述します（ORDER BYの後）。

## 例1：単純なLIMIT

例えば、学生（students）テーブルから最初の5人だけを取得するには：

```
SELECT * FROM students LIMIT 5;
```

実行結果：

| student_id | student_name |
|---|---|
| 301 | 黒沢春馬 |
| 302 | 新垣愛留 |
| 303 | 柴崎春花 |
| 304 | 森下風凛 |
| 305 | 河口菜恵子 |

### ORDER BYとLIMITの組み合わせ

通常、LIMIT句はORDER BY句と組み合わせて使用します。これにより、「上位N件」「最新N件」などの操作が可能になります。

### 例2：ORDER BYとLIMITの組み合わせ

例えば、成績（grades）テーブルから得点（score）の高い順に上位3件を取得するには：

```
SELECT * FROM grades ORDER BY score DESC LIMIT 3;
```

実行結果：

| student_id | course_id | grade_type | score | max_score | submission_date |
|---|---|---|---|---|---|
| 311 | 1 | 中間テスト | 95.0 | 100.0 | 2025-05-20 |
| 320 | 1 | 中間テスト | 93.5 | 100.0 | 2025-05-20 |
| 302 | 1 | 中間テスト | 92.0 | 100.0 | 2025-05-20 |

### 例3：最新のレコードを取得

日付でソートして最新のデータを取得することもよくあります。例えば、最新の3つの授業スケジュールを取得するには：

```
SELECT * FROM course_schedule
ORDER BY schedule_date DESC LIMIT 3;
```

実行結果：

| schedule_id | course_id | schedule_date | period_id | classroom_id | teacher_id | status |
|---|---|---|---|---|---|---|
| 95 | 28 | 2026-12-21 | 3 | 202D | 119 | scheduled |
| 94 | 1 | 2026-12-21 | 1 | 102B | 101 | scheduled |
| 93 | 14 | 2026-12-15 | 4 | 202D | 110 | scheduled |

## OFFSETとページネーション

Webアプリケーションなどでは、大量のデータを「ページ」に分けて表示することがよくあります（ページネーション）。この機能を実現するためには、「OFFSET」（オフセット）という機能が必要です。

> **用語解説**：
>
> - **OFFSET**：「ずらす」という意味で、結果セットの先頭から指定した数だけ行をスキップします。
> - **ページネーション**：大量のデータを複数のページに分割して表示する技術です。

### 基本構文（MySQL/MariaDB）

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] [ORDER BY 並び順] LIMIT 件数 OFFSET スキップ数;
```

または、短縮形として：

```
SELECT カラム名 FROM テーブル名 [WHERE 条件] [ORDER BY 並び順] LIMIT スキップ数, 件数;
```

### 例4：OFFSETを使ったスキップ

例えば、学生（students）テーブルから6番目から10番目までの学生を取得するには：

```
SELECT * FROM students LIMIT 5 OFFSET 5;
```

または：

```
SELECT * FROM students LIMIT 5, 5;
```

実行結果：

| student_id | student_name |
|---|---|
| 306 | 河田咲奈 |
| 307 | 織田柚夏 |
| 308 | 永田悦子 |
| 309 | 相沢吉夫 |
| 310 | 吉川伽羅 |

### 例5：ページネーションの実装

ページネーションを実装する場合、通常は以下の式を使ってOFFSETを計算します：

```
OFFSET = (ページ番号 - 1) × ページあたりの件数
```

例えば、1ページあたり10件表示で、3ページ目のデータを取得するには：

```
SELECT * FROM students ORDER BY student_id LIMIT 10 OFFSET 20;
```

または：

```
SELECT * FROM students ORDER BY student_id LIMIT 20, 10;
```

実行結果：

| student_id | student_name |
|---|---|
| 321 | 井上竜也 |
| 322 | 木村結衣 |
| 323 | 林正義 |
| 324 | 清水香織 |
| 325 | 山田翔太 |
| 326 | 葉山陽太 |
| 327 | 青山凛 |
| 328 | 沢村大和 |
| 329 | 白石優月 |
| 330 | 月岡星奈 |

## LIMIT句を使用する際の注意点

### 1. ORDER BYの重要性

LIMIT句を使用する場合、通常はORDER BY句も一緒に使うべきです。ORDER BYがなければ、どのレコードが取得されるかは保証されません。

```
-- 良い例：結果が予測可能
SELECT * FROM students ORDER BY student_id LIMIT 5;

-- 悪い例：結果が不確定
SELECT * FROM students LIMIT 5;
```

### 2. パフォーマンスへの影響

大規模なテーブルで大きなOFFSET値を使用すると、パフォーマンスが低下する可能性があります。これは、データベースがOFFSET分のレコードを読み込んでから破棄する必要があるためです。

### 3. データベース製品による構文の違い

LIMIT句の構文はデータベース製品によって異なります：

- **MySQL/MariaDB/SQLite**：`LIMIT 件数 OFFSET スキップ数` または `LIMIT スキップ数, 件数`
- **PostgreSQL**：`LIMIT 件数 OFFSET スキップ数`
- **Oracle**：`OFFSET スキップ数 ROWS FETCH NEXT 件数 ROWS ONLY`
- **SQL Server**：`OFFSET スキップ数 ROWS FETCH NEXT 件数 ROWS ONLY` または旧バージョンでは `TOP`句

この章では主にMySQL/MariaDBの構文を使用します。

## 実践的なページネーションの実装

実際のアプリケーションでページネーションを実装する場合、以下のようなコードになります（疑似コード）：

```
ページ番号 = URLから取得またはデフォルト値（例：1）
1ページあたりの件数 = 設定値（例：10）
総レコード数 = SELECTで取得（COUNT(*)を使用）
総ページ数 = CEILING(総レコード数 ÷ 1ページあたりの件数)
OFFSET = (ページ番号 - 1) × 1ページあたりの件数

SQLクエリ = "SELECT * FROM テーブル ORDER BY カラム LIMIT " + 1ページあたりの件数 + " OFFSET
" + OFFSET
```

### 例6：総レコード数と総ページ数の取得

総レコード数を取得するには：

```sql
SELECT COUNT(*) AS total_records FROM students;
```

実行結果：

| total_records |
| --- |
| 100 |

この場合、1ページあたり10件表示なら、総ページ数は10ページ（CEILING(100 ÷ 10)）になります。

## 練習問題

### 問題8-1

grades（成績）テーブルから、得点（score）の高い順に上位5件の成績レコードを取得するSQLを書いてください。

### 問題8-2

course_schedule（授業カレンダー）テーブルから、日付（schedule_date）の新しい順に3件のスケジュールを取得するSQLを書いてください。

### 問題8-3

students（学生）テーブルを学生ID（student_id）の昇順で並べ、11番目から15番目までの学生（5件）を取得するSQLを書いてください。

### 問題8-4

teachers（教師）テーブルから、教師名（teacher_name）の五十音順で6番目から10番目までの教師情報を取得するSQLを書いてください。

### 問題8-5

1ページあたり20件表示で、grades（成績）テーブルの3ページ目のデータを得点（score）の高い順に取得するSQLを書いてください。

### 問題8-6

course_schedule（授業カレンダー）テーブルから、状態（status）が「scheduled」のスケジュールを日付（schedule_date）の昇順で並べ、先頭から10件スキップして次の5件を取得するSQLを書いてください。

## 解答

### 解答8-1

```
SELECT * FROM grades ORDER BY score DESC LIMIT 5;
```

### 解答8-2

```
SELECT * FROM course_schedule ORDER BY schedule_date DESC LIMIT 3;
```

### 解答8-3

```
SELECT * FROM students ORDER BY student_id LIMIT 5 OFFSET 10;
```

または

```
SELECT * FROM students ORDER BY student_id LIMIT 10, 5;
```

### 解答8-4

```
SELECT * FROM teachers ORDER BY teacher_name LIMIT 5 OFFSET 5;
```

または

```
SELECT * FROM teachers ORDER BY teacher_name LIMIT 5, 5;
```

解答8-5

```
SELECT * FROM grades ORDER BY score DESC LIMIT 20 OFFSET 40;
```

または

```
SELECT * FROM grades ORDER BY score DESC LIMIT 40, 20;
```

解答8-6

```
SELECT * FROM course_schedule
WHERE status = 'scheduled'
ORDER BY schedule_date
LIMIT 5 OFFSET 10;
```

または

```
SELECT * FROM course_schedule
WHERE status = 'scheduled'
ORDER BY schedule_date
LIMIT 10, 5;
```

# まとめ

この章では、クエリ結果の件数を制限するためのLIMIT句と、ページネーションの実装方法について学びました：

1. **LIMIT句の基本**：指定した件数だけのレコードを取得する方法
2. **ORDER BYとの組み合わせ**：順序付けされた結果から一部だけを取得する方法
3. **OFFSET**：結果の先頭から指定した数だけレコードをスキップする方法
4. **ページネーション**：大量のデータを複数のページに分けて表示する実装方法
5. **注意点**：LIMIT句を使用する際の留意事項
6. **データベース製品による違い**：異なるデータベースでの構文の違い

LIMIT句は特にWebアプリケーションの開発で重要な機能です。大量のデータを効率よく表示するためのページネーション機能を実装するために欠かせません。また、トップN（上位N件）やボトムN（下位N件）のデータを取得する際にも使われます。

次の章では、データの集計分析を行うための「集計関数：COUNT、SUM、AVG、MAX、MIN」について学びます。