==========

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

```

```

```sql
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

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
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

student_id	student_name	course_id	score
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
…	…	…	…
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

student_id	student_name	course_id	course_name
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

```sql
-- 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

```sql
-- 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

```sql
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

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）

```
-- 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
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
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。
