==========

42. ER図：エンティティ関連モデル

はじめに

これまでの章では、既に存在するテーブルを使ってSQLの様々な機能を学習してきました。しかし、実際の開発現場では、**テーブルを作る前にデータベースの設計を行う**必要があります。

この章では、データベース設計の基礎となる「**ER図**（Entity-Relationship Diagram：エンティティ関連図）」について学習します。ER図は、データベースを作る前に「どんなテーブルが必要で、それらがどのような関係を持つのか」を図で表現する手法です。

> **用語解説**：
>
> - **エンティティ（Entity）**：データベースで管理したい「もの」や「概念」のことです。例：学生、教師、講座など
> - **関連（Relationship）**：エンティティ同士の関係のことです。例：学生は講座を受講する、教師は講座を担当する
> - **属性（Attribute）**：エンティティが持つ特徴や性質のことです。例：学生の名前、学生ID、入学日など
> - **ER図（Entity-Relationship Diagram）**：エンティティとその関連を図で表現したものです
> - **データベース設計**：データベースの構造を決める作業のことです
> - **概念設計**：データベースの全体的な構造を決める設計段階です
> - **論理設計**：概念設計を基に、具体的なテーブル構造を決める設計段階です
> - **物理設計**：実際のデータベース管理システムに合わせて、詳細な設定を決める設計段階です

学校システムを例に、ER図の読み方、書き方、そして実際のテーブル作成への応用方法を実践的に学んでいきます。

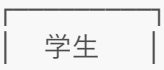
ER図の基本要素

1. エンティティ（Entity）

エンティティとは、データベースで管理したい「対象」のことです。学校システムでは以下のようなエンティティが考えられます：

- **学生（Students）**：学校に通う生徒
- **教師（Teachers）**：授業を行う先生
- **講座（Courses）**：開講されている授業
- **教室（Classrooms）**：授業が行われる部屋
- **時間割（Schedule）**：いつ、どこで、何の授業があるかの情報

ER図では、エンティティを**長方形**で表現します：



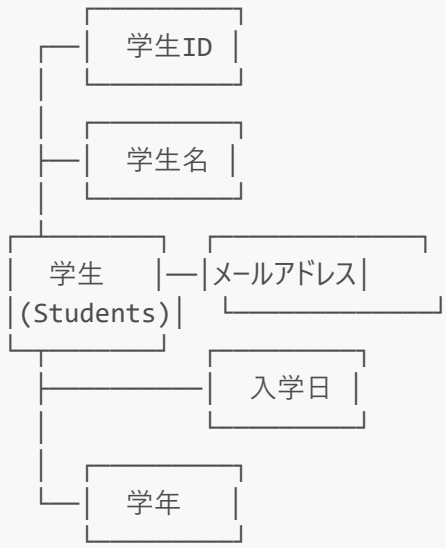
```
(Students)
```

2. 属性（Attribute）

属性とは、エンティティが持つ具体的な情報のことです。例えば「学生」エンティティの属性は以下のようになります：

- **学生ID**：学生を識別するための番号
- **学生名**：学生の名前
- **メールアドレス**：連絡先
- **入学日**：いつ入学したか
- **学年**：現在何年生か

ER図では、属性を**楕円形**で表現し、線でエンティティと結びます：

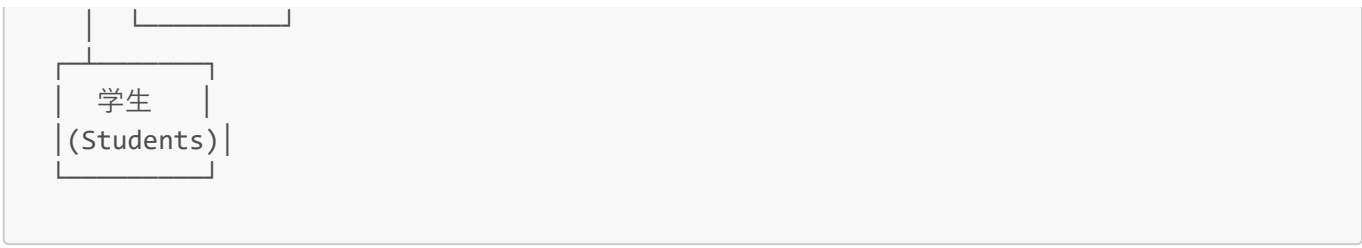


3. 主キー属性

主キーとは、そのエンティティの各レコードを一意に識別できる属性のことです。学生エンティティでは「学生ID」が主キーになります。

> **用語解説**：
>
> - **主キー（Primary Key）**：テーブル内の各行を一意に識別するための列（または列の組み合わせ）です
> - **一意（ユニーク）**：重複がない、同じ値が存在しないという意味です
> - **識別**：区別して認識することです

ER図では、主キー属性に**下線**を引いて表現します：



4. 関連（Relationship）

関連とは、エンティティ同士の関係を表現します。例えば：

- 学生は講座を**受講する**
- 教師は講座を**担当する**
- 講座は教室で**開講される**

ER図では、関連を**ひし形**で表現します：

関連の種類（カーディナリティ）

エンティティ間の関連には、**どのくらいの数のレコードが関係するか**を表す「**カーディナリティ**」という概念があります。

> **用語解説**：
>
> - **カーディナリティ（Cardinality）**：関連において、一方のエンティティの1つのレコードに対して、他方のエンティティの何個のレコードが関係するかを表す概念です

1. 一対一関係（1:1）

一方のエンティティの1つのレコードに対して、他方のエンティティの1つのレコードだけが関係する場合です。

例：教師と担任クラスの関係

- 1人の教師は1つのクラスだけを担任する
- 1つのクラスは1人の教師だけが担任する

2. 一対多関係（1:N）

一方のエンティティの1つのレコードに対して、他方のエンティティの複数のレコードが関係する場合です。

例：教師と講座の関係

- 1人の教師は複数の講座を担当できる
- 1つの講座は1人の教師だけが担当する

3. 多対多関係（M:N）

双方のエンティティで、1つのレコードが相手エンティティの複数のレコードと関係する場合です。

例：学生と講座の関係

- 1人の学生は複数の講座を受講できる
- 1つの講座は複数の学生が受講できる

> **重要なポイント**： 多対多関係は、実際のデータベースでは**中間テーブル**を作って、**2つの一対多関係**に分解して実装します。これについては後で詳しく説明します。

学校システムのER図作成

実際に学校システムの全体的なER図を作成してみましょう。

1. エンティティの抽出

まず、学校システムで管理したい「もの」を洗い出します：

1. **学生（Students）**
2. **教師（Teachers）**
3. **講座（Courses）**
4. **教室（Classrooms）**
5. **授業時間（Class_Periods）**
6. **学期（Terms）**
7. **授業スケジュール（Course_Schedule）**
8. **出席（Attendance）**
9. **成績（Grades）**

2. 各エンティティの属性定義

学生（Students）

- **学生ID**（主キー）：301, 302, 303...
- **学生名**：田中太郎、佐藤花子...
- **メールアドレス**：連絡先
- **入学日**：2023-04-01...

教師（Teachers）

- **教師ID**（主キー）：101, 102, 103...
- **教師名**：田中先生、佐藤先生...

講座（Courses）

- **講座ID**（主キー）：1, 2, 3...
- **講座名**：プログラミング基礎、データベース...
- **担当教師ID**（外部キー）：どの教師が担当するか

> **用語解説**：
>
> - **外部キー（Foreign Key）**：他のテーブルの主キーを参照する列のことです。テーブル間の関係を作るために使います

教室（Classrooms）

- **教室ID**（主キー）：A101, B201...
- **教室名**：コンピュータ室1、講義室A...
- **収容人数**：30人、50人...
- **建物**：A棟、B棟...
- **設備**：プロジェクター、PC...

授業時間（Class_Periods）

- **時限ID**（主キー）：1, 2, 3, 4, 5
- **開始時間**：09:00, 10:50, 13:00...
- **終了時間**：10:30, 12:20, 14:30...

3. 関連の定義

各エンティティ間の関係を整理します：

1. **教師** → **講座**（1:N）
    - 1人の教師は複数の講座を担当する
    - 1つの講座は1人の教師が担当する
2. **学生** ← → **講座**（M:N）
    - 1人の学生は複数の講座を受講する
    - 1つの講座は複数の学生が受講する

3. **講座** → **授業スケジュール**（1:N）
    - 1つの講座は複数回の授業がある
    - 1回の授業は1つの講座に属する
4. **教室** → **授業スケジュール**（1:N）
    - 1つの教室で複数の授業が行われる
    - 1回の授業は1つの教室で行われる
5. **授業時間** → **授業スケジュール**（1:N）
    - 1つの時限で複数の授業が行われる（異なる日に）
    - 1回の授業は1つの時限で行われる
6. **授業スケジュール** → **出席**（1:N）
    - 1回の授業に対して複数の学生の出席記録がある
    - 1つの出席記録は1回の授業に対応する
7. **学生** → **出席**（1:N）
    - 1人の学生は複数の出席記録を持つ
    - 1つの出席記録は1人の学生のもの
8. **学生** ← → **講座**（成績を通じてM:N）
    - 1人の学生は複数の講座で成績を持つ
    - 1つの講座で複数の学生が成績を持つ

4. 完成したER図の概念

学校システムの全体的なER図は以下のような構造になります：

```
学生（Students）
├ 学生ID（主キー）
├ 学生名
├ メールアドレス
└ 入学日
    │
    │ 受講（M:N）
    │
    ▼
講座（Courses）
├ 講座ID（主キー）
├ 講座名
└ 担当教師ID（外部キー）
    │
    │ 担当（1:N）
    │
    ▼
教師（Teachers）
├ 教師ID（主キー）
```

```
└ 教師名

講座（Courses）
    │
    │ 開講（1:N）
    │
    ▼
授業スケジュール（Course_Schedule）
├ スケジュールID（主キー）
├ 講座ID（外部キー）
├ 日付
├ 時限ID（外部キー）
├ 教室ID（外部キー）
└ 教師ID（外部キー）
    │
    ├ 使用（N:1）
    │   ▼
    │ 教室（Classrooms）
    │ ├ 教室ID（主キー）
    │ ├ 教室名
    │ ├ 収容人数
    │ ├ 建物
    │ └ 設備
    │
    ├ 時限（N:1）
    │   ▼
    │ 授業時間（Class_Periods）
    │ ├ 時限ID（主キー）
    │ ├ 開始時間
    │ └ 終了時間
    │
    └ 出席管理（1:N）
        ▼
      出席（Attendance）
      ├ スケジュールID（外部キー）
      ├ 学生ID（外部キー）
      ├ 出席状況
      └ コメント

学生（Students）
    │
    │ 成績管理（M:N）
    │
    ▼
成績（Grades）
├ 学生ID（外部キー）
├ 講座ID（外部キー）
├ 成績種別
├ 点数
├ 満点
└ 提出日
```

ER図からテーブル設計への変換

ER図ができたら、次は実際のテーブル設計に変換します。この過程を「**論理設計**」と呼びます。

1. エンティティからテーブルの作成

各エンティティは1つのテーブルになります：

```
-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64),
    student_email VARCHAR(100),
    admission_date DATE
);

-- 教師テーブル
CREATE TABLE teachers (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64)
);

-- 講座テーブル
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);
```

2. 多対多関係の解決

多対多関係は**中間テーブル**を作成して解決します。例えば「学生と講座」の多対多関係は：

```
-- 受講テーブル（学生と講座の中間テーブル）
CREATE TABLE student_courses (
    course_id VARCHAR(16),
    student_id BIGINT,
    enrollment_date DATE DEFAULT CURRENT_DATE,
    PRIMARY KEY (course_id, student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);
```

> **重要なポイント**：
>
> - 中間テーブルの主キーは、関連する両方のエンティティの主キーを組み合わせた**複合主キー**になります
> - 中間テーブルには、関連に関する追加情報（受講開始日など）も格納できます

3. 一対多関係の実装

一対多関係は、「多」の側のテーブルに「一」の側の主キーを**外部キー**として追加します：

```
-- 授業スケジュールテーブル
CREATE TABLE course_schedule (
    schedule_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_id VARCHAR(16) NOT NULL,
    schedule_date DATE NOT NULL,
    period_id INT NOT NULL,
    classroom_id VARCHAR(16) NOT NULL,
    teacher_id BIGINT NOT NULL,

    FOREIGN KEY (course_id) REFERENCES courses(course_id),
    FOREIGN KEY (period_id) REFERENCES class_periods(period_id),
    FOREIGN KEY (classroom_id) REFERENCES classrooms(classroom_id),
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);
```

実践的なER図作成手順

実際にER図を作成する際の手順を整理します：

ステップ1：要求分析

システムで「何を管理したいか」を明確にします：

- どんな情報を記録する必要があるか？
- どんな処理を行う必要があるか？
- どんなレポートを出力する必要があるか？

ステップ2：エンティティの抽出

管理対象となる「もの」を洗い出します：

- **名詞に注目**：要求仕様書に出てくる名詞がエンティティの候補
- **重要度の判定**：システムにとって本当に重要なものかを判断
- **具体性の確認**：抽象的すぎないか、具体的すぎないかを確認

ステップ3：属性の定義

各エンティティが持つべき情報を定義します：

- **識別子の選定**：主キーとなる属性を決める
- **必須属性の特定**：必ず値が入る属性を特定
- **データ型の決定**：文字列、数値、日付などの型を決める

ステップ4：関連の特定

エンティティ間の関係を分析します：

- **動詞に注目**：「受講する」「担当する」などの動詞が関連の候補
- **カーディナリティの決定**：1:1、1:N、M:Nのどれかを決める
- **制約の確認**：関連に制約があるかを確認

ステップ5：ER図の作成

図を描いて全体の構造を可視化します：

- **読みやすさを重視**：線の交差を避け、整理された配置にする
- **一貫性の確保**：記号の使い方を統一する
- **完全性の確認**：漏れがないかをチェックする

ステップ6：検証と改善

作成したER図が要求を満たしているかを確認します：

- **機能要求の確認**：必要な処理が実現できるか
- **整合性の確認**：矛盾がないか
- **拡張性の確認**：将来の変更に対応できるか

ER図作成の実習

実際に簡単なER図を作成してみましょう。

課題：図書館システムのER図作成

図書館システムに必要なエンティティと関連を考えてみてください：

管理したい情報：

- 本の情報（タイトル、著者、ISBN、出版社、出版年）
- 利用者の情報（利用者ID、名前、メールアドレス、登録日）
- 貸出記録（いつ、誰が、何を借りたか、返却期限、返却日）
- 著者の情報（著者ID、著者名、生年月日、国籍）

実現したい機能：

- 本の貸出・返却管理
- 利用者の貸出履歴確認
- 延滞者の管理
- 蔵書検索

解答例

エンティティの抽出

1. **本（Books）**
2. **利用者（Users）**
3. **著者（Authors）**
4. **貸出記録（Rentals）**

属性の定義

本（Books）

- 本ID（主キー）
- タイトル
- ISBN
- 出版社
- 出版年

利用者（Users）

- 利用者ID（主キー）
- 利用者名
- メールアドレス
- 登録日

著者（Authors）

- 著者ID（主キー）
- 著者名
- 生年月日
- 国籍

貸出記録（Rentals）

- 貸出ID（主キー）
- 本ID（外部キー）
- 利用者ID（外部キー）
- 貸出日
- 返却期限
- 返却日

関連の定義

1. **著者** ← → **本**（M:N）
    - 1人の著者は複数の本を書く
    - 1冊の本は複数の著者が書く場合がある
    - 中間テーブル「著者本関係（Author_Books）」が必要
2. **利用者** → **貸出記録**（1:N）
    - 1人の利用者は複数の貸出記録を持つ
    - 1つの貸出記録は1人の利用者のもの
3. **本** → **貸出記録**（1:N）
    - 1冊の本は複数回貸し出される（複数の貸出記録）
    - 1つの貸出記録は1冊の本に対応

テーブル設計例

```
-- 著者テーブル
CREATE TABLE authors (
    author_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    author_name VARCHAR(100) NOT NULL,
    birth_date DATE,
    nationality VARCHAR(50)
);

-- 本テーブル
CREATE TABLE books (
    book_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    isbn VARCHAR(20) UNIQUE,
    publisher VARCHAR(100),
    publication_year YEAR
);

-- 著者本関係テーブル（多対多の解決）
CREATE TABLE author_books (
    author_id BIGINT,
    book_id BIGINT,
    PRIMARY KEY (author_id, book_id),
    FOREIGN KEY (author_id) REFERENCES authors(author_id),
    FOREIGN KEY (book_id) REFERENCES books(book_id)
);

-- 利用者テーブル
CREATE TABLE users (
    user_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE,
    registration_date DATE DEFAULT CURRENT_DATE
);

-- 貸出記録テーブル
CREATE TABLE rentals (
    rental_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    book_id BIGINT NOT NULL,
    user_id BIGINT NOT NULL,
    rental_date DATE DEFAULT CURRENT_DATE,
    due_date DATE NOT NULL,
    return_date DATE NULL,

    FOREIGN KEY (book_id) REFERENCES books(book_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

ER図の表記法

ER図には、いくつかの表記法があります。ここでは代表的なものを紹介します。

1. Chen記法（チェン記法）

最も伝統的な記法で、教科書でよく使われます：

- **エンティティ**：長方形
- **属性**：楕円形
- **関連**：ひし形
- **主キー**：下線

2. IE記法（Information Engineering記法）

実務でよく使われる記法です：

- **エンティティ**：長方形（属性も内部に記載）
- **関連**：線で接続
- **カーディナリティ**：線の端に記号で表現

3. UML記法（Unified Modeling Language）

オブジェクト指向開発でよく使われます：

- **エンティティ**：クラス図として表現
- **関連**：線で接続し、多重度を記載

実際の開発現場では、**IE記法**や**UML記法**がよく使われています。ツールも豊富で、実用的だからです。

ER図作成ツール

ER図を作成するための代表的なツールを紹介します：

無料ツール

1. **draw.io（現：app.diagrams.net）**
    - ブラウザで使える無料ツール
    - ER図用のテンプレートが豊富
    - チーム共有機能もある
2. **MySQL Workbench**
    - MySQLの公式ツール
    - ER図から直接テーブルを生成可能
    - 既存のデータベースからER図を自動生成
3. **dbdiagram.io**
    - シンプルで使いやすいオンラインツール
    - テキストでER図を記述できる
    - SQLコードの自動生成機能

有料ツール

1. **ERwin Data Modeler**
    - 企業レベルのデータモデリングツール
    - 高度な機能が充実
2. **Lucidchart**
    - 汎用的な図表作成ツール
    - ER図テンプレートも豊富
3. **Microsoft Visio**
    - Microsoftの図表作成ツール
    - 企業環境でよく使われる

ER図設計のベストプラクティス

良いER図を作成するためのポイントをまとめます：

1. 命名規則の統一

エンティティ名：

- 複数形を使う（Students、Teachers、Courses）
- 分かりやすい英語名を使う
- 略語は避ける

属性名：

- 明確で具体的な名前を使う
- データ型が分かりやすい名前にする
- ID属性は「エンティティ名_id」の形式に統一

2. 適切な抽象化レベル

抽象化しすぎない：

- 「人」エンティティではなく、「学生」「教師」と具体的に
- システムの目的に合った粒度にする

具体化しすぎない：

- 「1年生」「2年生」を別エンティティにしない
- 属性で表現できるものはエンティティにしない

3. 関連の明確化

関連名を明記：

- 単に線で結ぶだけでなく、関連の意味を明記
- 「受講する」「担当する」「開講される」など

カーディナリティの根拠：

- なぜその関連度数になるのかを説明できるようにする
- 業務ルールとの整合性を確認

4. 将来の拡張性を考慮

柔軟性の確保：

- 将来の機能追加に対応できる構造にする
- 過度に制約をかけすぎない

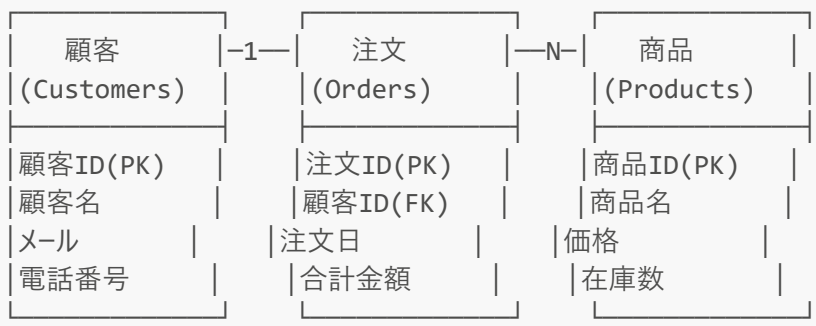
保守性の向上：

- 理解しやすい構造にする
- ドキュメントを充実させる

練習問題

問題42-1：基本的なER図の読み取り

以下のER図を見て、質問に答えてください：



質問：

1. このER図で表現されているエンティティは何個ありますか？
2. 「顧客」と「注文」の関係はどのような関連ですか？
3. 「注文」エンティティの主キーは何ですか？
4. 「注文」エンティティの外部キーは何ですか？
5. この構造では、1つの注文で複数の商品を購入することはできますか？理由も説明してください。

問題42-2：エンティティと属性の識別

以下の説明から、エンティティと属性を識別してください：

病院システムの要求：「病院では、患者の診療記録を管理したい。患者には患者番号、氏名、生年月日、住所、電話番号がある。医師には医師番号、氏名、専門科目がある。診療記録には、いつ、どの患者を、どの医師が診療したか、診療内容、処方薬の情報を記録する。薬には薬品番号、薬品名、単価がある。」

課題：

1. エンティティを4つ抽出してください
2. 各エンティティの属性を列挙してください
3. 各エンティティの主キーとなる属性を選んでください

問題42-3：関連とカーディナリティの分析

以下の業務ルールから、エンティティ間の関連とカーディナリティを分析してください：

レンタルビデオ店の業務ルール：

- 1人の会員は複数のDVDを借りることができる
- 1枚のDVDは同時に1人の会員だけが借りることができる
- 1枚のDVDは複数回貸し出される（返却後に別の会員が借りる）
- 1つのジャンル（アクション、コメディなど）には複数のDVDが属する
- 1枚のDVDは1つのジャンルに属する

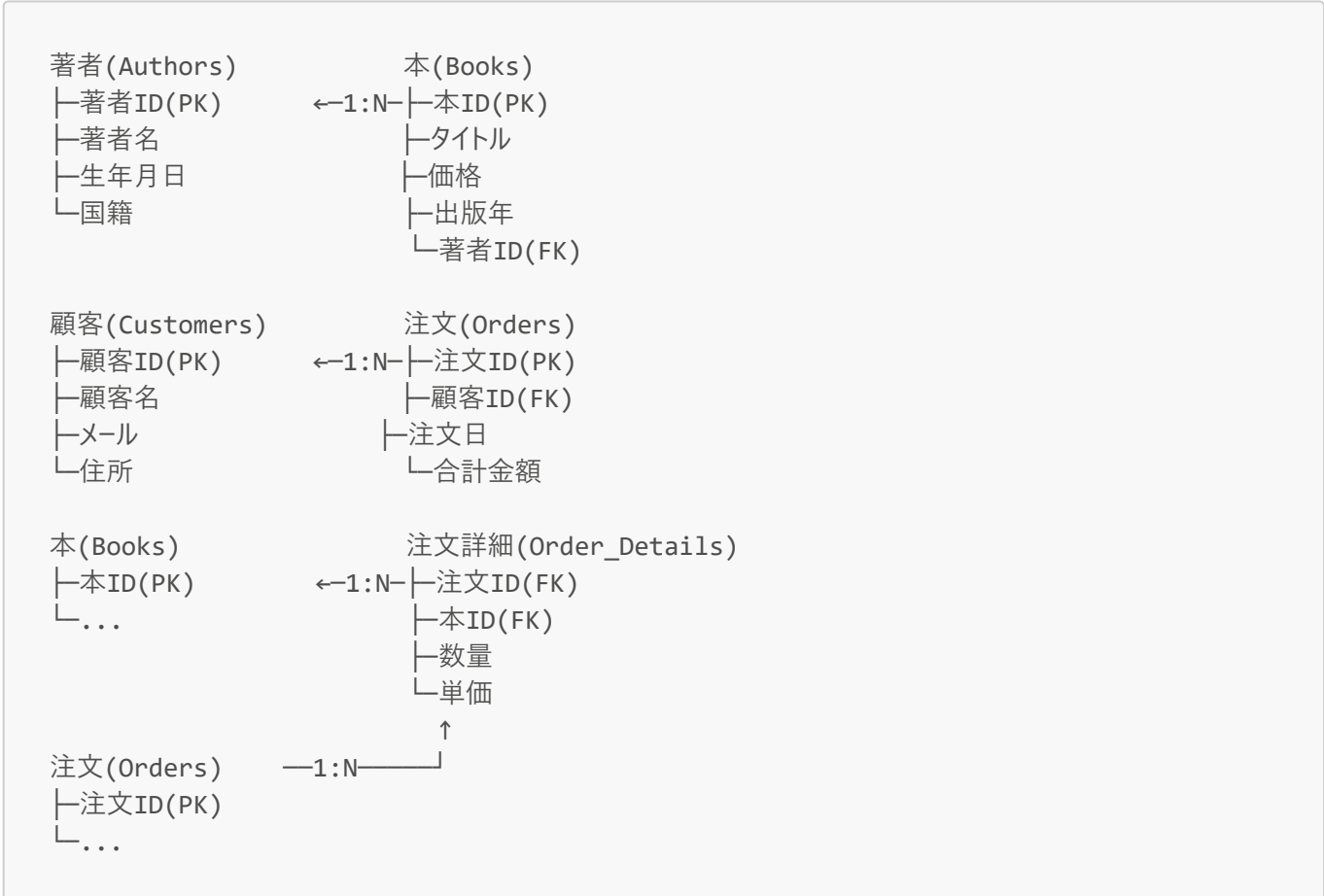
課題：

1. 必要なエンティティを抽出してください
2. エンティティ間の関連を特定してください
3. 各関連のカーディナリティ（1:1、1:N、M:N）を決定してください
4. 多対多関係がある場合は、中間テーブルの必要性を説明してください

問題42-4：ER図からテーブル設計

以下のER図を基に、適切なCREATE TABLE文を作成してください：

オンライン書店のER図：



課題：

1. 各エンティティに対応するテーブルのCREATE TABLE文を書いてください
2. 適切な制約（主キー、外部キー、NOT NULL等）を設定してください
3. データ型も適切に選択してください

問題42-5：複雑なER図の設計

以下の要求仕様を基に、完全なER図を設計してください：

大学の履修管理システム：

要求：

- 学生は複数の科目を履修できる
- 科目は複数の学生が履修できる
- 各科目には担当教授が1人いる
- 教授は複数の科目を担当できる
- 学生には学籍番号、氏名、学年、学部の情報がある
- 教授には教授番号、氏名、所属学部、職位の情報がある
- 科目には科目番号、科目名、単位数、開講学期の情報がある
- 履修には履修年度、成績（A、B、C、D、F）の情報がある
- 学部には学部番号、学部名の情報がある

課題：

1. 必要なエンティティをすべて抽出してください
2. 各エンティティの属性を定義してください
3. エンティティ間の関連とカーディナリティを決定してください
4. 完全なER図を設計してください
5. 多対多関係があれば中間テーブルを設計してください

問題42-6：既存システムの分析と改善

学校データベースの現在の構造を分析し、改善案を提案してください：

現在のテーブル構造（簡略版）：

```
-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64),
    teacher_name VARCHAR(64)  -- 担任の名前
);

-- 成績テーブル
CREATE TABLE grades (
    student_id BIGINT,
    subject_name VARCHAR(64),  -- 科目名
    teacher_name VARCHAR(64),  -- 担当教師名
```

```
    score DECIMAL(5,2)
);
```

課題：

1. 現在の設計の問題点を3つ以上指摘してください
2. 正規化の観点から改善すべき点を説明してください
3. 改善されたER図を設計してください
4. 改善後のテーブル構造をCREATE TABLE文で表現してください

解答

解答42-1

1. **エンティティの数**：3個（顧客、注文、商品）
2. **顧客と注文の関係**：一対多関係（1:N）
    - 1人の顧客は複数の注文をできる
    - 1つの注文は1人の顧客のもの
3. **注文エンティティの主キー**：注文ID
4. **注文エンティティの外部キー**：顧客ID
5. **複数商品の購入**：できません
    - 理由：注文と商品が直接結ばれており、中間テーブルがない
    - 現在の構造では1つの注文に1つの商品しか関連付けられない
    - 複数商品を扱うには「注文詳細」という中間テーブルが必要

解答42-2

1. **エンティティ**：
    - 患者（Patients）
    - 医師（Doctors）
    - 診療記録（Medical_Records）
    - 薬（Medicines）
2. **各エンティティの属性**：
    - **患者**：患者番号、氏名、生年月日、住所、電話番号
    - **医師**：医師番号、氏名、専門科目
    - **診療記録**：診療日、患者番号、医師番号、診療内容、処方薬番号
    - **薬**：薬品番号、薬品名、単価
3. **主キー**：
    - **患者**：患者番号
    - **医師**：医師番号
    - **診療記録**：診療記録番号（新たに追加）
    - **薬**：薬品番号

解答42-3

1. **エンティティ**：
    - 会員（Members）
    - DVD（DVDs）
    - ジャンル（Genres）
    - 貸出記録（Rentals）
2. **関連の特定**：
    - 会員 ← → DVD（貸出を通じて）
    - ジャンル → DVD
    - 会員 → 貸出記録
    - DVD → 貸出記録
3. **カーディナリティ**：
    - **ジャンル → DVD**：1:N（1つのジャンルに複数のDVD）
    - **会員 → 貸出記録**：1:N（1人の会員が複数回借りる）
    - **DVD → 貸出記録**：1:N（1枚のDVDが複数回貸し出される）
4. **中間テーブル**：
    - 「貸出記録（Rentals）」が実質的に会員とDVDの中間テーブルとして機能
    - 現在の貸出状況を管理するために必要

解答42-4

```
-- 著者テーブル
CREATE TABLE authors (
    author_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    author_name VARCHAR(100) NOT NULL,
    birth_date DATE,
    nationality VARCHAR(50)
);

-- 本テーブル
CREATE TABLE books (
    book_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    publication_year YEAR,
    author_id BIGINT NOT NULL,

    FOREIGN KEY (author_id) REFERENCES authors(author_id)
);

-- 顧客テーブル
CREATE TABLE customers (
    customer_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE,
    address TEXT
```

```
);

-- 注文テーブル
CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    customer_id BIGINT NOT NULL,
    order_date DATE DEFAULT CURRENT_DATE,
    total_amount DECIMAL(12,2) NOT NULL DEFAULT 0,

    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- 注文詳細テーブル
CREATE TABLE order_details (
    order_id BIGINT,
    book_id BIGINT,
    quantity INT NOT NULL DEFAULT 1,
    unit_price DECIMAL(10,2) NOT NULL,

    PRIMARY KEY (order_id, book_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (book_id) REFERENCES books(book_id)
);
```

解答42-5

1. **エンティティ**：
    - 学生（Students）
    - 教授（Professors）
    - 科目（Subjects）
    - 学部（Departments）
    - 履修記録（Enrollments）
2. **属性定義**：
    - **学生**：学籍番号（PK）、氏名、学年、学部番号（FK）
    - **教授**：教授番号（PK）、氏名、所属学部番号（FK）、職位
    - **科目**：科目番号（PK）、科目名、単位数、開講学期、担当教授番号（FK）
    - **学部**：学部番号（PK）、学部名
    - **履修記録**：学籍番号（FK）、科目番号（FK）、履修年度、成績
3. **関連とカーディナリティ**：
    - **学部** → **学生**：1:N
    - **学部** → **教授**：1:N
    - **教授** → **科目**：1:N
    - **学生** ← → **科目**：M:N（履修記録を通じて）
4. **テーブル設計**：

```sql
-- 学部テーブル
CREATE TABLE departments (
    department_id INT AUTO_INCREMENT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL
);

-- 教授テーブル
CREATE TABLE professors (
    professor_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    professor_name VARCHAR(100) NOT NULL,
    department_id INT NOT NULL,
    position VARCHAR(50),

    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    grade_level INT NOT NULL,
    department_id INT NOT NULL,

    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- 科目テーブル
CREATE TABLE subjects (
    subject_id VARCHAR(16) PRIMARY KEY,
    subject_name VARCHAR(128) NOT NULL,
    credits INT NOT NULL,
    semester ENUM('spring', 'fall', 'summer') NOT NULL,
    professor_id BIGINT NOT NULL,

    FOREIGN KEY (professor_id) REFERENCES professors(professor_id)
);

-- 履修記録テーブル（多対多の解決）
CREATE TABLE enrollments (
    student_id BIGINT,
    subject_id VARCHAR(16),
    academic_year YEAR NOT NULL,
    grade ENUM('A', 'B', 'C', 'D', 'F', 'W') DEFAULT NULL,

    PRIMARY KEY (student_id, subject_id, academic_year),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id)
);
```

解答42-6

1. **現在の設計の問題点：**

- **非正規化**：teacher_name、subject_nameが重複格納されている
- **整合性の問題**：教師名や科目名の変更時に複数箇所を更新する必要がある
- **外部キー制約がない**：データの整合性が保証されない
- **拡張性の欠如**：教師や科目の詳細情報を追加できない

2. **正規化の観点からの改善点**：

- **第1正規形**：繰り返し項目の分離
- **第2正規形**：部分関数従属の除去
- **第3正規形**：推移関数従属の除去
- エンティティの独立性確保

3. **改善されたER図**：

```
学生(Students)          教師(Teachers)
├─学生ID(PK)      ←─1:N─├─教師ID(PK)
├─学生名                ├─教師名
└─担任教師ID(FK)        └─...

科目(Subjects)
├─科目ID(PK)
├─科目名
├─担当教師ID(FK)  ──N:1→ 教師(Teachers)
└─...

成績(Grades)
├─学生ID(FK)      ──N:1→ 学生(Students)
├─科目ID(FK)      ──N:1→ 科目(Subjects)
├─成績種別(PK)
├─点数
└─...
```

4. **改善後のテーブル構造**：

```sql
-- 教師テーブル
CREATE TABLE teachers (
    teacher_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    teacher_name VARCHAR(100) NOT NULL
);

-- 学生テーブル（改善版）
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64) NOT NULL,
    homeroom_teacher_id BIGINT,

    FOREIGN KEY (homeroom_teacher_id) REFERENCES teachers(teacher_id)
);

-- 科目テーブル
```

```sql
CREATE TABLE subjects (
    subject_id VARCHAR(16) PRIMARY KEY,
    subject_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);

-- 成績テーブル（改善版）
CREATE TABLE grades (
    student_id BIGINT,
    subject_id VARCHAR(16),
    grade_type VARCHAR(32),
    score DECIMAL(5,2),
    submission_date DATE,

    PRIMARY KEY (student_id, subject_id, grade_type),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id)
);
```

まとめ

この章では、データベース設計の基礎となるER図について詳しく学習しました：

1. **ER図の基本要素**：
    - エンティティ（長方形）：管理対象となる「もの」
    - 属性（楕円形）：エンティティの特徴や性質
    - 関連（ひし形）：エンティティ間の関係
    - 主キー（下線）：レコードを一意に識別する属性
2. **関連の種類**：
    - **一対一関係（1:1）**：両方のエンティティで1つのレコードずつが関連
    - **一対多関係（1:N）**：一方の1つに対して他方の複数が関連
    - **多対多関係（M:N）**：両方のエンティティで複数のレコードが関連
3. **ER図からテーブル設計への変換**：
    - エンティティは1つのテーブルになる
    - 一対多関係は外部キーで実装
    - 多対多関係は中間テーブルで解決
4. **設計のベストプラクティス**：
    - 命名規則の統一
    - 適切な抽象化レベル
    - 関連の明確化
    - 将来の拡張性を考慮

ER図は、データベース設計の「設計図」として非常に重要です。**要求を正確に分析し、適切なエンティティと関連を定義することで、保守性と拡張性の高いデータベースを設計**できます。

次の章では、「正規化：データの冗長性削減」について学び、データベース設計をさらに洗練させる手法を理解していきます。

---

43. 正規化：データの冗長性削減

はじめに

前章でER図によるデータベース設計を学びました。しかし、ER図から作成したテーブル設計がそのまま最適とは限りません。データベースをより効率的で保守しやすくするために、「**正規化**」という技法が必要です。

正規化とは、**データの重複（冗長性）を取り除き、データの整合性を保ちやすい構造に変換する**プロセスです。正規化を行うことで、データの更新時の問題を防ぎ、ストレージ容量の節約にもつながります。

> **用語解説**：
>
> - **正規化（Normalization）**：データベースの構造を改善し、データの重複を排除するプロセスです
> - **冗長性（Redundancy）**：同じデータが複数の場所に重複して格納されている状態です
> - **整合性（Consistency）**：データベース内のデータが矛盾なく一貫している状態です
> - **関数従属（Functional Dependency）**：ある列の値が決まると、他の列の値が一意に決まる関係です
> - **部分関数従属（Partial Functional Dependency）**：複合主キーの一部だけで、他の列の値が決まってしまう関係です
> - **推移関数従属（Transitive Functional Dependency）**：A→B、B→Cの関係があるとき、A→Cが成り立つ関係です
> - **候補キー（Candidate Key）**：主キーになりうる属性（または属性の組み合わせ）です
> - **非キー属性（Non-key Attribute）**：どの候補キーにも含まれない属性です

この章では、学校システムの具体例を使って、正規化の各段階を詳しく学習し、実践的なスキルを身につけます。

正規化が必要な理由

正規化前の問題のあるテーブル例

以下のような「学生情報統合テーブル」があるとします：

```
CREATE TABLE student_info_bad (
    student_id BIGINT,
    student_name VARCHAR(64),
    teacher_id BIGINT,
    teacher_name VARCHAR(64),
    teacher_department VARCHAR(64),
    course_id VARCHAR(16),
```

```
    course_name VARCHAR(128),
    grade_type VARCHAR(32),
    score DECIMAL(5,2),
    submission_date DATE
);

-- サンプルデータ
INSERT INTO student_info_bad VALUES
(301, '田中太郎', 101, '佐藤先生', '情報学部', '1', 'プログラミング基礎', 'テスト', 85.0,
'2025-05-15'),
(301, '田中太郎', 101, '佐藤先生', '情報学部', '1', 'プログラミング基礎', 'レポート', 90.0,
'2025-05-20'),
(301, '田中太郎', 102, '鈴木先生', '数学科', '2', 'データベース', 'テスト', 78.0, '2025-
05-18'),
(302, '山田花子', 101, '佐藤先生', '情報学部', '1', 'プログラミング基礎', 'テスト', 92.0,
'2025-05-15'),
(302, '山田花子', 102, '鈴木先生', '数学科', '2', 'データベース', 'レポート', 88.0, '2025-
05-22');
```

このテーブルの問題点

1. 更新異常（Update Anomaly）

佐藤先生の名前を「佐藤太郎先生」に変更したい場合、複数の行を更新する必要があります。1つでも更新し忘れると、データの不整合が発生します。

```
-- 間違った更新：一部の行だけ更新してしまった場合
UPDATE student_info_bad
SET teacher_name = '佐藤太郎先生'
WHERE student_id = 301 AND course_id = '1' AND grade_type = 'テスト';

-- 結果：同じ教師ID 101に対して異なる名前が存在してしまう
```

2. 挿入異常（Insert Anomaly）

新しい教師情報だけを登録したい場合、学生や成績の情報がないと登録できません。

```
-- 不可能：教師情報だけでは挿入できない
INSERT INTO student_info_bad (teacher_id, teacher_name, teacher_department)
VALUES (103, '高橋先生', '英語科');
-- エラー：student_idなど他の列にもNULLでない値が必要
```

3. 削除異常（Delete Anomaly）

学生301番がプログラミング基礎の成績をすべて削除すると、佐藤先生の情報も一緒に消えてしまう可能性があります。

4. ストレージの無駄

同じ教師情報、学生情報、講座情報が何度も重複して格納されます。

正規化の段階

正規化は段階的に行われ、それぞれの段階を「**正規形**」と呼びます。主要な正規形は以下の通りです：

1. **第1正規形（1NF）**：反復項目の除去
2. **第2正規形（2NF）**：部分関数従属の除去
3. **第3正規形（3NF）**：推移関数従属の除去
4. **ボイス・コッド正規形（BCNF）**：すべての関数従属の正規化
5. **第4正規形（4NF）**、**第5正規形（5NF）**：多値従属性の処理

実用的には、第3正規形まで行えば十分なケースがほとんどです。

第1正規形（1NF）

定義

第1正規形とは、テーブルのすべての列が**原子的な値**（これ以上分割できない単一の値）を持っている状態です。

> **用語解説**：
>
> - **原子的な値（Atomic Value）**：それ以上分割できない最小単位の値です
> - **反復項目（Repeating Group）**：同じ種類のデータが複数個、1つの行に格納されている状態です

第1正規形に違反している例

```
-- NG：第1正規形に違反
CREATE TABLE students_bad_1nf (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64),
    courses VARCHAR(200),  -- 複数の講座が1つの列に格納されている
    phone_numbers VARCHAR(100)  -- 複数の電話番号が1つの列に格納されている
);

INSERT INTO students_bad_1nf VALUES
(301, '田中太郎', 'プログラミング基礎,データベース,ネットワーク', '090-1234-5678,03-1234-5678'),
(302, '山田花子', 'プログラミング基礎,Webデザイン', '080-9876-5432');
```

問題点：

- 特定の講座を受講している学生を検索するのが困難
- 講座の追加・削除が複雑
- データの整合性チェックが困難

第1正規形への変換

方法1：行の複製

```
-- OK：第1正規形に準拠（ただし他の問題あり）
CREATE TABLE students_1nf_v1 (
    student_id BIGINT,
    student_name VARCHAR(64),
    course VARCHAR(64),
    phone_number VARCHAR(20)
);

INSERT INTO students_1nf_v1 VALUES
(301, '田中太郎', 'プログラミング基礎', '090-1234-5678'),
(301, '田中太郎', 'プログラミング基礎', '03-1234-5678'),
(301, '田中太郎', 'データベース', '090-1234-5678'),
(301, '田中太郎', 'データベース', '03-1234-5678'),
(301, '田中太郎', 'ネットワーク', '090-1234-5678'),
(301, '田中太郎', 'ネットワーク', '03-1234-5678'),
(302, '山田花子', 'プログラミング基礎', '080-9876-5432'),
(302, '山田花子', 'Webデザイン', '080-9876-5432');
```

方法2：テーブルの分割（推奨）

```
-- 学生基本情報テーブル
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64)
);

-- 学生電話番号テーブル
CREATE TABLE student_phones (
    student_id BIGINT,
    phone_number VARCHAR(20),
    phone_type ENUM('mobile', 'home', 'work') DEFAULT 'mobile',
    PRIMARY KEY (student_id, phone_number),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);

-- 学生受講テーブル
CREATE TABLE student_courses (
    student_id BIGINT,
    course_id VARCHAR(16),
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);
```

第2正規形（2NF）

定義

第2正規形とは、第1正規形を満たし、かつ**部分関数従属**がない状態です。

部分関数従属とは、複合主キーの一部だけで、非キー属性の値が決まってしまう関係のことです。

第2正規形に違反している例

```
-- NG：第2正規形に違反
CREATE TABLE enrollment_bad_2nf (
    student_id BIGINT,
    course_id VARCHAR(16),
    student_name VARCHAR(64),     -- student_idだけで決まる（部分関数従属）
    course_name VARCHAR(128),     -- course_idだけで決まる（部分関数従属）
    teacher_name VARCHAR(64),     -- course_idだけで決まる（部分関数従属）
    enrollment_date DATE,         -- 複合主キー全体で決まる
    grade CHAR(1),               -- 複合主キー全体で決まる

    PRIMARY KEY (student_id, course_id)
);

INSERT INTO enrollment_bad_2nf VALUES
(301, '1', '田中太郎', 'プログラミング基礎', '佐藤先生', '2025-04-01', 'A'),
(301, '2', '田中太郎', 'データベース', '鈴木先生', '2025-04-01', 'B'),
(302, '1', '山田花子', 'プログラミング基礎', '佐藤先生', '2025-04-01', 'A');
```

関数従属の分析：

- `student_id → student_name`（部分関数従属）
- `course_id → course_name, teacher_name`（部分関数従属）
- `(student_id, course_id) → enrollment_date, grade`（完全関数従属）

問題点：

- 学生名や講座名の変更時に複数行を更新する必要
- 同じ学生・講座情報の重複格納
- 新しい学生や講座だけを登録できない

第2正規形への変換

部分関数従属を除去するために、テーブルを分割します：

```
-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64) NOT NULL
);

-- 講座テーブル
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
```

```
    teacher_name VARCHAR(64) NOT NULL
);

-- 受講テーブル（関連情報のみ）
CREATE TABLE enrollments (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE NOT NULL,
    grade CHAR(1),

    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);

-- データ投入
INSERT INTO students VALUES
(301, '田中太郎'),
(302, '山田花子');

INSERT INTO courses VALUES
('1', 'プログラミング基礎', '佐藤先生'),
('2', 'データベース', '鈴木先生');

INSERT INTO enrollments VALUES
(301, '1', '2025-04-01', 'A'),
(301, '2', '2025-04-01', 'B'),
(302, '1', '2025-04-01', 'A');
```

第3正規形（3NF）

定義

第3正規形とは、第2正規形を満たし、かつ**推移関数従属**がない状態です。

推移関数従属とは、「A → B」かつ「B → C」の関係があるとき、「A → C」が成り立つ関係のことです。

第3正規形に違反している例

第2正規形のcoursesテーブルを詳しく見てみましょう：

```
-- NG：第3正規形に違反
CREATE TABLE courses_bad_3nf (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,
    teacher_name VARCHAR(64) NOT NULL,     -- 推移関数従属
    teacher_department VARCHAR(64) NOT NULL,  -- 推移関数従属
    classroom_id VARCHAR(16) NOT NULL,
    classroom_name VARCHAR(64) NOT NULL,  -- 推移関数従属
```

```
    building VARCHAR(32) NOT NULL         -- 推移関数従属
);
```

関数従属の分析：

- `course_id → teacher_id`
- `teacher_id → teacher_name, teacher_department`（推移関数従属）
- `course_id → classroom_id`
- `classroom_id → classroom_name, building`（推移関数従属）

問題点：

- 教師名や教室名の変更時に複数の講座レコードを更新する必要
- 同じ教師・教室情報の重複格納
- 講座がない教師や教室の情報を単独で管理できない

第3正規形への変換

推移関数従属を除去するために、さらにテーブルを分割します：

```
-- 教師テーブル
CREATE TABLE teachers (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64) NOT NULL,
    teacher_department VARCHAR(64) NOT NULL
);

-- 教室テーブル
CREATE TABLE classrooms (
    classroom_id VARCHAR(16) PRIMARY KEY,
    classroom_name VARCHAR(64) NOT NULL,
    building VARCHAR(32) NOT NULL,
    capacity INT
);

-- 講座テーブル（第3正規形）
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,
    classroom_id VARCHAR(16) NOT NULL,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id),
    FOREIGN KEY (classroom_id) REFERENCES classrooms(classroom_id)
);

-- データ投入
INSERT INTO teachers VALUES
(101, '佐藤先生', '情報学部'),
(102, '鈴木先生', '数学科');
```

```
INSERT INTO classrooms VALUES
('A101', 'コンピュータ室1', 'A棟', 30),
('B201', '講義室B', 'B棟', 50);

INSERT INTO courses VALUES
('1', 'プログラミング基礎', 101, 'A101'),
('2', 'データベース', 102, 'A101');
```

ボイス・コッド正規形（BCNF）

定義

ボイス・コッド正規形とは、第3正規形を満たし、かつ**すべての関数従属の決定項が候補キーである**状態です。

> **用語解説**：
>
> - **決定項（Determinant）**：関数従属「A → B」において、Aの部分です
> - **従属項（Dependent）**：関数従属「A → B」において、Bの部分です

BCNFに違反している例

以下のような「講座時間割テーブル」を考えます：

```
-- NG：BCNFに違反
CREATE TABLE course_schedule_bad_bcnf (
    course_id VARCHAR(16),
    day_of_week ENUM('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'),
    time_slot INT,
    teacher_id BIGINT,
    classroom_id VARCHAR(16),

    PRIMARY KEY (course_id, day_of_week, time_slot)
);
```

想定する制約：

- 1つの講座は特定の教師が担当する：`course_id → teacher_id`
- 1つの教室の特定時間は1つの講座のみ：`(classroom_id, day_of_week, time_slot) → course_id`

問題：

- `course_id → teacher_id`の決定項`course_id`は候補キーではない（候補キーの一部）
- このため、同じ講座が複数の時間に開講される場合、教師情報が重複格納される

BCNFへの変換

```
-- 講座基本情報テーブル
CREATE TABLE courses (
```

```
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);

-- 講座スケジュールテーブル
CREATE TABLE course_schedules (
    course_id VARCHAR(16),
    day_of_week ENUM('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'),
    time_slot INT,
    classroom_id VARCHAR(16),

    PRIMARY KEY (course_id, day_of_week, time_slot),
    UNIQUE KEY unique_classroom_time (classroom_id, day_of_week, time_slot),
    FOREIGN KEY (course_id) REFERENCES courses(course_id),
    FOREIGN KEY (classroom_id) REFERENCES classrooms(classroom_id)
);
```

学校システムの完全正規化例

学校システム全体を第3正規形まで正規化した例を示します：

```
-- 1. 基本エンティティテーブル

-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64) NOT NULL,
    email VARCHAR(100) UNIQUE,
    admission_date DATE NOT NULL
);

-- 教師テーブル
CREATE TABLE teachers (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64) NOT NULL,
    department VARCHAR(64),
    email VARCHAR(100) UNIQUE
);

-- 教室テーブル
CREATE TABLE classrooms (
    classroom_id VARCHAR(16) PRIMARY KEY,
    classroom_name VARCHAR(64) NOT NULL,
    building VARCHAR(32) NOT NULL,
    capacity INT NOT NULL,
    facilities TEXT
);
```

```sql
-- 学期テーブル
CREATE TABLE terms (
    term_id VARCHAR(16) PRIMARY KEY,
    term_name VARCHAR(64) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL
);

-- 授業時間テーブル
CREATE TABLE class_periods (
    period_id INT PRIMARY KEY,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL
);

-- 2. 関連テーブル

-- 講座テーブル
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,
    term_id VARCHAR(16) NOT NULL,
    credits INT DEFAULT 1,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id),
    FOREIGN KEY (term_id) REFERENCES terms(term_id)
);

-- 学生受講テーブル（多対多の解決）
CREATE TABLE student_courses (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);

-- 授業スケジュールテーブル
CREATE TABLE course_schedule (
    schedule_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_id VARCHAR(16) NOT NULL,
    classroom_id VARCHAR(16) NOT NULL,
    period_id INT NOT NULL,
    schedule_date DATE NOT NULL,
    status ENUM('scheduled', 'completed', 'cancelled') DEFAULT 'scheduled',

    FOREIGN KEY (course_id) REFERENCES courses(course_id),
    FOREIGN KEY (classroom_id) REFERENCES classrooms(classroom_id),
    FOREIGN KEY (period_id) REFERENCES class_periods(period_id),

    UNIQUE KEY unique_classroom_time (classroom_id, period_id, schedule_date)
```

```
);

-- 出席テーブル
CREATE TABLE attendance (
    schedule_id BIGINT,
    student_id BIGINT,
    status ENUM('present', 'absent', 'late') NOT NULL,
    comment TEXT,

    PRIMARY KEY (schedule_id, student_id),
    FOREIGN KEY (schedule_id) REFERENCES course_schedule(schedule_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);

-- 成績テーブル
CREATE TABLE grades (
    student_id BIGINT,
    course_id VARCHAR(16),
    grade_type VARCHAR(32),
    score DECIMAL(5,2) NOT NULL,
    max_score DECIMAL(5,2) NOT NULL,
    submission_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (student_id, course_id, grade_type),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

正規化の利点と欠点

利点

1. データ整合性の向上

```
-- 教師名の変更が1箇所だけで済む
UPDATE teachers SET teacher_name = '佐藤太郎先生' WHERE teacher_id = 101;

-- 正規化前なら複数箇所の更新が必要で、更新漏れのリスクがあった
```

2. ストレージ容量の節約

```
-- 正規化前：同じ教師情報が各講座レコードに重複
-- 正規化後：教師情報は1回だけ格納、講座テーブルでは教師IDのみ参照
```

3. 挿入・削除・更新の柔軟性

```
-- 新しい教師を講座と関係なく登録可能
INSERT INTO teachers VALUES (103, '高橋先生', '英語科', 'takahashi@school.ac.jp');

-- 講座を削除しても教師情報は残る
DELETE FROM courses WHERE course_id = '1';
```

欠点

1. 複雑なクエリが必要

```
-- 正規化前：1つのテーブルで完結
SELECT * FROM student_info_bad WHERE student_name = '田中太郎';

-- 正規化後：複数テーブルのJOINが必要
SELECT s.student_name, c.course_name, t.teacher_name, g.score
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
WHERE s.student_name = '田中太郎';
```

2. パフォーマンスの低下

- 多数のJOINが必要でクエリが重くなる場合がある
- 特に大量データの場合は影響が大きい

3. 設計・開発の複雑化

- テーブル数が増加し、関連が複雑になる
- 開発者の理解が必要

逆正規化（非正規化）

実際の運用では、パフォーマンスを重視して意図的に正規化を緩める「**逆正規化**」を行う場合があります。

> **用語解説**：
>
> - **逆正規化（Denormalization）**：パフォーマンス向上のために、意図的にデータの重複を許可する設計手法です

逆正規化の例

```
-- 頻繁にアクセスされる情報を重複格納
CREATE TABLE student_course_summary (
    student_id BIGINT,
    student_name VARCHAR(64),  -- 重複格納
```

```
    course_id VARCHAR(16),
    course_name VARCHAR(128),  -- 重複格納
    teacher_name VARCHAR(64),  -- 重複格納
    current_grade DECIMAL(5,2),
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    PRIMARY KEY (student_id, course_id),
    -- 元テーブルとの整合性は別途管理
    INDEX idx_student_name (student_name),
    INDEX idx_course_name (course_name)
);
```

逆正規化の注意点

1. **データ同期の管理**：元テーブルとの整合性を保つ仕組みが必要
2. **更新コストの増加**：複数箇所の更新が必要
3. **明確な目的**：パフォーマンス改善の明確な根拠が必要

正規化の実践的な指針

基本方針

1. **第3正規形を目標**：実用上、第3正規形で十分なケースがほとんど
2. **段階的な適用**：一度にすべて正規化せず、段階的に進める
3. **要件に応じた調整**：システムの要件に応じて柔軟に対応

正規化の判断基準

```
-- 正規化が有効なケース
-- 1. データの更新が頻繁
-- 2. データの整合性が重要
-- 3. ストレージ容量が限られている

-- 逆正規化を検討するケース
-- 1. 読み取り専用のデータ
-- 2. パフォーマンスが最重要
-- 3. データの更新が稀
```

正規化の検証方法

正規化が適切に行われているかを検証する手順：

1. 関数従属の確認

```
-- 各テーブルの関数従属を文書化
-- 例：students テーブル
-- student_id → student_name, email, admission_date
-- email → student_id, student_name, admission_date（候補キー）
```

2. 正規形の確認

```
-- チェックポイント
-- □ 第1正規形：すべての値が原子的か？
-- □ 第2正規形：部分関数従属がないか？
-- □ 第3正規形：推移関数従属がないか？
```

3. 制約の実装確認

```
-- 外部キー制約の確認
SELECT
    TABLE_NAME,
    COLUMN_NAME,
    CONSTRAINT_NAME,
    REFERENCED_TABLE_NAME,
    REFERENCED_COLUMN_NAME
FROM information_schema.KEY_COLUMN_USAGE
WHERE REFERENCED_TABLE_SCHEMA = 'school_db'
AND REFERENCED_TABLE_NAME IS NOT NULL;
```

練習問題

問題43-1：正規化レベルの判定

以下のテーブルがどの正規形まで満たしているかを判定し、問題点を指摘してください：

```
CREATE TABLE library_books (
    book_id INT PRIMARY KEY,
    title VARCHAR(200),
    authors VARCHAR(300),  -- 複数著者をカンマ区切りで格納
    publisher_name VARCHAR(100),
    publisher_address VARCHAR(200),
    isbn VARCHAR(20),
    price DECIMAL(8,2),
    category VARCHAR(50)
);
```

質問：

1. このテーブルは第1正規形を満たしていますか？
2. 問題がある場合、どのように修正すべきですか？
3. 正規化後のテーブル構造を設計してください。

問題43-2：部分関数従属の除去

以下のテーブルを第2正規形に変換してください：

```
CREATE TABLE order_details (
    order_id INT,
    product_id INT,
    customer_name VARCHAR(100),     -- order_idだけで決まる
    customer_address VARCHAR(200), -- order_idだけで決まる
    product_name VARCHAR(100),      -- product_idだけで決まる
    product_price DECIMAL(8,2),     -- product_idだけで決まる
    quantity INT,                   -- 複合主キー全体で決まる
    discount_rate DECIMAL(3,2),     -- 複合主キー全体で決まる

    PRIMARY KEY (order_id, product_id)
);
```

課題：

1. 部分関数従属を特定してください
2. 第2正規形に変換したテーブル構造を設計してください
3. 変換後のCREATE TABLE文を書いてください
4. 変換のメリットを3つ説明してください

問題43-3：推移関数従属の除去

以下のテーブルを第3正規形に変換してください：

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(100),
    department_id INT,
    department_name VARCHAR(100),  -- department_idで決まる
    department_location VARCHAR(100), -- department_idで決まる
    manager_id INT,
    manager_name VARCHAR(100),      -- manager_idで決まる
    salary DECIMAL(10,2),
    hire_date DATE
);
```

課題：

1. 推移関数従属を特定してください
2. 第3正規形に変換したテーブル構造を設計してください
3. データの整合性を保つための制約を追加してください
4. 変換前後でのデータ更新の違いを具体例で説明してください

問題43-4：複雑な正規化問題

病院システムの以下のテーブルを第3正規形まで正規化してください：

```
CREATE TABLE patient_treatments (
    patient_id INT,
    treatment_date DATE,
    patient_name VARCHAR(100),
    patient_address VARCHAR(200),
    patient_phone VARCHAR(20),
    doctor_id INT,
    doctor_name VARCHAR(100),
    doctor_specialty VARCHAR(50),
    treatment_code VARCHAR(10),
    treatment_name VARCHAR(100),
    treatment_cost DECIMAL(8,2),
    medicine_codes VARCHAR(200),  -- 複数の薬をカンマ区切り
    medicine_names VARCHAR(500),  -- 複数の薬名をカンマ区切り
    total_cost DECIMAL(10,2),

    PRIMARY KEY (patient_id, treatment_date, treatment_code)
);
```

課題：

1. 第1正規形への変換から段階的に正規化してください
2. 各段階での問題点と解決方法を説明してください
3. 最終的なテーブル構造をER図で表現してください
4. 正規化後のテーブルで「特定の患者の治療履歴を取得する」クエリを書いてください

問題43-5：逆正規化の判断

オンラインショップのシステムで、以下の正規化されたテーブル構造があります：

```
-- 顧客テーブル
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    email VARCHAR(100),
    address VARCHAR(200)
);

-- 注文テーブル
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    status VARCHAR(20),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- 商品テーブル
CREATE TABLE products (
    product_id INT PRIMARY KEY,
```

```
    product_name VARCHAR(100),
    price DECIMAL(8,2),
    category VARCHAR(50)
);

-- 注文詳細テーブル
CREATE TABLE order_items (
    order_id INT,
    product_id INT,
    quantity INT,
    unit_price DECIMAL(8,2),
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

業務要件：

- 顧客の注文履歴表示が最も頻繁な処理（全体の70%）
- 在庫管理システムでリアルタイムに商品情報を参照
- 月次売上レポートで集計処理を実行

課題：

1. どのテーブルに逆正規化を適用すべきか判断してください
2. 逆正規化後のテーブル構造を設計してください
3. 逆正規化による利点と欠点を整理してください
4. データ整合性を保つための仕組みを提案してください

問題43-6：実践的な正規化設計

大学の時間割管理システムを設計してください。以下の要件を満たすテーブル構造を第3正規形で設計してください：

要件：

- 学生は複数の科目を履修できる
- 科目は複数の時間帯で開講される場合もある
- 各時間帯には教室と担当教員が割り当てられる
- 同じ科目でも時間帯によって担当教員が異なる場合がある
- 学生の履修登録と成績管理を行う
- 教室には収容人数の制限がある
- 時間割の重複チェックが必要（学生・教員・教室）

管理したい情報：

- 学生：学籍番号、氏名、学年、学部
- 教員：教員番号、氏名、所属学部、職位
- 科目：科目番号、科目名、単位数、必修/選択
- 教室：教室番号、建物、収容人数、設備
- 時間帯：曜日、時限、開始時間、終了時間

- 履修：履修年度、成績
- 時間割：開講年度、学期

課題：

1. 必要なエンティティを抽出してください
2. エンティティ間の関連を分析してください
3. 第3正規形のテーブル構造を設計してください
4. 時間割重複チェックのためのCHECK制約やトリガーを考案してください
5. よく使われるクエリ（学生の時間割表示、教室利用状況確認など）を作成してください

解答

解答43-1

1. **第1正規形の判定**： このテーブルは第1正規形を**満たしていません**。

   理由：`authors`列に複数の著者がカンマ区切りで格納されており、原子的な値ではありません。

2. **修正方法**：

   - 著者情報を別テーブルに分離
   - 本と著者の多対多関係を中間テーブルで解決

3. **正規化後のテーブル構造**：

```
-- 出版社テーブル
CREATE TABLE publishers (
    publisher_id INT AUTO_INCREMENT PRIMARY KEY,
    publisher_name VARCHAR(100) NOT NULL,
    publisher_address VARCHAR(200)
);

-- 著者テーブル
CREATE TABLE authors (
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    author_name VARCHAR(100) NOT NULL
);

-- 本テーブル
CREATE TABLE books (
    book_id INT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    publisher_id INT NOT NULL,
    isbn VARCHAR(20) UNIQUE,
    price DECIMAL(8,2),
    category VARCHAR(50),

    FOREIGN KEY (publisher_id) REFERENCES publishers(publisher_id)
);

-- 本著者関係テーブル（多対多の解決）
```

```
CREATE TABLE book_authors (
    book_id INT,
    author_id INT,
    author_order INT DEFAULT 1,  -- 著者の順序

    PRIMARY KEY (book_id, author_id),
    FOREIGN KEY (book_id) REFERENCES books(book_id),
    FOREIGN KEY (author_id) REFERENCES authors(author_id)
);
```

解答43-2

1. **部分関数従属の特定：**
    - order_id → customer_name, customer_address
    - product_id → product_name, product_price
2. **第2正規形への変換：**

```
-- 顧客テーブル
CREATE TABLE customers (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(100) NOT NULL,
    customer_address VARCHAR(200)
);

-- 注文テーブル
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT NOT NULL,
    order_date DATE DEFAULT CURRENT_DATE,

    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- 商品テーブル
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100) NOT NULL,
    product_price DECIMAL(8,2) NOT NULL
);

-- 注文詳細テーブル
CREATE TABLE order_details (
    order_id INT,
    product_id INT,
    quantity INT NOT NULL,
    discount_rate DECIMAL(3,2) DEFAULT 0.00,

    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
```

```
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

3. **変換のメリット**：
    - **データ整合性向上**：顧客情報の変更が1箇所で済む
    - **ストレージ効率**：重複データの削減
    - **保守性向上**：商品価格の変更が簡単

解答43-3

1. **推移関数従属の特定**：

    - `employee_id → department_id → department_name, department_location`
    - `employee_id → manager_id → manager_name`

2. **第3正規形への変換**：

```
-- 部署テーブル
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL,
    department_location VARCHAR(100) NOT NULL
);

-- 従業員テーブル（第3正規形）
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(100) NOT NULL,
    department_id INT NOT NULL,
    manager_id INT,
    salary DECIMAL(10,2),
    hire_date DATE,

    FOREIGN KEY (department_id) REFERENCES departments(department_id),
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
);
```

3. **整合性制約**：

```
-- セルフ参照制約（従業員は自分を管理者にできない）
ALTER TABLE employees
ADD CONSTRAINT chk_manager_not_self
CHECK (employee_id != manager_id);

-- 管理者は同じ部署または上位部署の従業員である制約
--  （複雑な制約はトリガーで実装）
```

4. **データ更新の違い**：

```sql
-- 変換前：部署名変更時（複数行更新が必要）
UPDATE employees
SET department_name = '新情報システム部'
WHERE department_id = 1;

-- 変換後：部署名変更時（1行更新で済む）
UPDATE departments
SET department_name = '新情報システム部'
WHERE department_id = 1;
```

解答43-4

1. **段階的正規化**：

第1正規形への変換

```sql
-- 薬情報の分離
CREATE TABLE patient_treatment_medicines (
    patient_id INT,
    treatment_date DATE,
    treatment_code VARCHAR(10),
    medicine_code VARCHAR(10),
    medicine_name VARCHAR(100),

    PRIMARY KEY (patient_id, treatment_date, treatment_code, medicine_code)
);
```

第2正規形への変換

```sql
-- 患者テーブル
CREATE TABLE patients (
    patient_id INT PRIMARY KEY,
    patient_name VARCHAR(100) NOT NULL,
    patient_address VARCHAR(200),
    patient_phone VARCHAR(20)
);

-- 医師テーブル
CREATE TABLE doctors (
    doctor_id INT PRIMARY KEY,
    doctor_name VARCHAR(100) NOT NULL,
    doctor_specialty VARCHAR(50)
);

-- 治療テーブル
CREATE TABLE treatments (
    treatment_code VARCHAR(10) PRIMARY KEY,
```

```
    treatment_name VARCHAR(100) NOT NULL,
    treatment_cost DECIMAL(8,2)
);

-- 薬テーブル
CREATE TABLE medicines (
    medicine_code VARCHAR(10) PRIMARY KEY,
    medicine_name VARCHAR(100) NOT NULL
);
```

第3正規形（最終形）

```
-- 診療記録テーブル
CREATE TABLE patient_treatments (
    treatment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    patient_id INT NOT NULL,
    doctor_id INT NOT NULL,
    treatment_date DATE NOT NULL,
    treatment_code VARCHAR(10) NOT NULL,
    total_cost DECIMAL(10,2),

    FOREIGN KEY (patient_id) REFERENCES patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id),
    FOREIGN KEY (treatment_code) REFERENCES treatments(treatment_code)
);

-- 診療薬剤テーブル
CREATE TABLE treatment_medicines (
    treatment_id BIGINT,
    medicine_code VARCHAR(10),

    PRIMARY KEY (treatment_id, medicine_code),
    FOREIGN KEY (treatment_id) REFERENCES patient_treatments(treatment_id),
    FOREIGN KEY (medicine_code) REFERENCES medicines(medicine_code)
);
```

2. **特定患者の治療履歴取得クエリ**：

```
SELECT
    pt.treatment_date,
    d.doctor_name,
    d.doctor_specialty,
    t.treatment_name,
    t.treatment_cost,
    GROUP_CONCAT(m.medicine_name) as medicines,
    pt.total_cost
FROM patient_treatments pt
JOIN doctors d ON pt.doctor_id = d.doctor_id
JOIN treatments t ON pt.treatment_code = t.treatment_code
```

```
LEFT JOIN treatment_medicines tm ON pt.treatment_id = tm.treatment_id
LEFT JOIN medicines m ON tm.medicine_code = m.medicine_code
WHERE pt.patient_id = 12345
GROUP BY pt.treatment_id
ORDER BY pt.treatment_date DESC;
```

解答43-5

1. **逆正規化の適用判断**： 最も頻繁な処理（注文履歴表示）を最適化するため、以下に逆正規化を適用：

```
-- 注文履歴サマリーテーブル（逆正規化）
CREATE TABLE order_history_summary (
    order_id INT PRIMARY KEY,
    customer_id INT,
    customer_name VARCHAR(100),     -- 重複格納
    customer_email VARCHAR(100),    -- 重複格納
    order_date DATE,
    total_amount DECIMAL(10,2),
    item_count INT,
    status VARCHAR(20),
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    INDEX idx_customer_date (customer_id, order_date)
);
```

2. **利点と欠点**：

利点：

- 注文履歴表示が高速化（JOINが不要）
- 顧客情報への負荷軽減
- シンプルなクエリで済む

欠点：

- 顧客情報変更時の同期が必要
- ストレージ使用量の増加
- データ不整合のリスク

3. **整合性保持の仕組み**：

```
-- トリガーによる自動同期
DELIMITER //
CREATE TRIGGER update_order_summary_on_customer_change
AFTER UPDATE ON customers
FOR EACH ROW
BEGIN
    UPDATE order_history_summary
```

```
    SET customer_name = NEW.customer_name,
        customer_email = NEW.email,
        last_updated = CURRENT_TIMESTAMP
    WHERE customer_id = NEW.customer_id;
END //
DELIMITER ;

-- 定期的な整合性チェック用クエリ
SELECT o.order_id
FROM order_history_summary o
JOIN customers c ON o.customer_id = c.customer_id
WHERE o.customer_name != c.customer_name
   OR o.customer_email != c.email;
```

解答43-6

1. **エンティティの抽出**：
    - 学生（Students）
    - 教員（Faculty）
    - 科目（Subjects）
    - 教室（Classrooms）
    - 時間帯（Time_Slots）
    - 学部（Departments）
    - 履修記録（Enrollments）
    - 時間割（Schedule）
2. **第3正規形のテーブル構造**：

```
-- 学部テーブル
CREATE TABLE departments (
    department_id INT AUTO_INCREMENT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL
);

-- 学生テーブル
CREATE TABLE students (
    student_id VARCHAR(20) PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    grade_level INT NOT NULL,
    department_id INT NOT NULL,

    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- 教員テーブル
CREATE TABLE faculty (
    faculty_id INT AUTO_INCREMENT PRIMARY KEY,
    faculty_name VARCHAR(100) NOT NULL,
    department_id INT NOT NULL,
```

```
    position ENUM('assistant', 'associate', 'professor') NOT NULL,

    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);

-- 教室テーブル
CREATE TABLE classrooms (
    classroom_id VARCHAR(20) PRIMARY KEY,
    building VARCHAR(50) NOT NULL,
    capacity INT NOT NULL,
    equipment TEXT
);

-- 時間帯テーブル
CREATE TABLE time_slots (
    slot_id INT AUTO_INCREMENT PRIMARY KEY,
    day_of_week ENUM('Monday','Tuesday','Wednesday','Thursday','Friday') NOT NULL,
    period INT NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,

    UNIQUE KEY unique_day_period (day_of_week, period)
);

-- 科目テーブル
CREATE TABLE subjects (
    subject_id VARCHAR(20) PRIMARY KEY,
    subject_name VARCHAR(128) NOT NULL,
    credits INT NOT NULL,
    subject_type ENUM('required', 'elective') NOT NULL
);

-- 時間割テーブル
CREATE TABLE schedules (
    schedule_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    subject_id VARCHAR(20) NOT NULL,
    faculty_id INT NOT NULL,
    classroom_id VARCHAR(20) NOT NULL,
    slot_id INT NOT NULL,
    academic_year YEAR NOT NULL,
    semester ENUM('spring', 'fall') NOT NULL,

    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id),
    FOREIGN KEY (faculty_id) REFERENCES faculty(faculty_id),
    FOREIGN KEY (classroom_id) REFERENCES classrooms(classroom_id),
    FOREIGN KEY (slot_id) REFERENCES time_slots(slot_id),

    -- 重複防止制約
    UNIQUE KEY unique_classroom_time (classroom_id, slot_id, academic_year,
semester),
    UNIQUE KEY unique_faculty_time (faculty_id, slot_id, academic_year, semester)
);

-- 履修記録テーブル
```

```
CREATE TABLE enrollments (
    student_id VARCHAR(20),
    schedule_id BIGINT,
    academic_year YEAR NOT NULL,
    grade ENUM('A+','A','B+','B','C+','C','D+','D','F') DEFAULT NULL,

    PRIMARY KEY (student_id, schedule_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (schedule_id) REFERENCES schedules(schedule_id)
);
```

3. **重複チェック制約**：

```
-- 学生の時間割重複チェック用ビュー
CREATE VIEW student_schedule_conflicts AS
SELECT
    e1.student_id,
    e1.schedule_id as schedule1_id,
    e2.schedule_id as schedule2_id,
    s1.academic_year,
    s1.semester
FROM enrollments e1
JOIN enrollments e2 ON e1.student_id = e2.student_id AND e1.schedule_id <
e2.schedule_id
JOIN schedules s1 ON e1.schedule_id = s1.schedule_id
JOIN schedules s2 ON e2.schedule_id = s2.schedule_id
WHERE s1.slot_id = s2.slot_id
AND s1.academic_year = s2.academic_year
AND s1.semester = s2.semester;
```

4. **よく使われるクエリ**：

```
-- 学生の時間割表示
SELECT
    ts.day_of_week,
    ts.period,
    ts.start_time,
    ts.end_time,
    sub.subject_name,
    f.faculty_name,
    c.classroom_id,
    c.building
FROM enrollments e
JOIN schedules s ON e.schedule_id = s.schedule_id
JOIN subjects sub ON s.subject_id = sub.subject_id
JOIN faculty f ON s.faculty_id = f.faculty_id
JOIN classrooms c ON s.classroom_id = c.classroom_id
JOIN time_slots ts ON s.slot_id = ts.slot_id
WHERE e.student_id = 'S2025001'
```

```sql
AND s.academic_year = 2025
AND s.semester = 'spring'
ORDER BY ts.day_of_week, ts.period;

-- 教室利用状況確認
SELECT
    c.classroom_id,
    c.building,
    ts.day_of_week,
    ts.period,
    sub.subject_name,
    f.faculty_name,
    COUNT(e.student_id) as enrolled_students,
    c.capacity,
    ROUND(COUNT(e.student_id) * 100.0 / c.capacity, 1) as occupancy_rate
FROM schedules s
JOIN classrooms c ON s.classroom_id = c.classroom_id
JOIN time_slots ts ON s.slot_id = ts.slot_id
JOIN subjects sub ON s.subject_id = sub.subject_id
JOIN faculty f ON s.faculty_id = f.faculty_id
LEFT JOIN enrollments e ON s.schedule_id = e.schedule_id
WHERE s.academic_year = 2025
AND s.semester = 'spring'
GROUP BY s.schedule_id
ORDER BY c.classroom_id, ts.day_of_week, ts.period;
```

まとめ

この章では、データベース正規化について詳しく学習しました：

1. **正規化の目的**：
    - データの冗長性削除
    - 更新・挿入・削除異常の防止
    - データ整合性の向上
    - ストレージ効率の改善
2. **正規化の段階**：
    - **第1正規形（1NF）**：原子的な値、反復項目の除去
    - **第2正規形（2NF）**：部分関数従属の除去
    - **第3正規形（3NF）**：推移関数従属の除去
    - **ボイス・コッド正規形（BCNF）**：決定項が候補キーの制約
3. **実践的な考慮事項**：
    - パフォーマンスとのバランス
    - 逆正規化の適用場面
    - 業務要件に応じた調整
4. **設計指針**：

- 第3正規形を基本目標とする
- 段階的な適用
- 明確な目的を持った逆正規化

正規化は、**保守性の高いデータベース設計の基礎**となる重要な技法です。ただし、盲目的に適用するのではなく、**システムの要件とパフォーマンスを考慮したバランスの取れた設計**が重要です。

次の章では、「キー：主キー、外部キー、候補キー」について学び、データベースの整合性を保つための制約について理解を深めていきます。

---

44. キー：主キー、外部キー、候補キー

はじめに

前章では正規化について学習し、データの冗長性を削除する方法を理解しました。正規化の過程で頻繁に登場したのが「**キー**」という概念です。

キーは、データベースにおいて**データの一意性を保証し、テーブル間の関係を定義する**重要な役割を果たします。適切なキー設計により、データの整合性が保たれ、効率的なデータ操作が可能になります。

> **用語解説：**
>
> - **キー（Key）**：テーブル内の行を識別したり、テーブル間の関係を定義したりするための属性（列）または属性の組み合わせです
> - **一意性（Uniqueness）**：同じ値が重複して存在しないことです
> - **識別子（Identifier）**：レコードを他と区別するために使用される値です
> - **制約（Constraint）**：データベースに格納されるデータに対するルールや条件です
> - **参照整合性（Referential Integrity）**：外部キーによって関連付けられたテーブル間でデータの整合性が保たれている状態です
> - **複合キー（Composite Key）**：複数の列を組み合わせて作られるキーです
> - **代理キー（Surrogate Key）**：業務的な意味を持たない、システムが自動生成するキーです
> - **自然キー（Natural Key）**：業務的な意味を持つ、実際のデータから構成されるキーです

この章では、学校システムを例に、キーの種類と特徴、設計における注意点を実践的に学習します。

キーの種類と基本概念

1. スーパーキー（Super Key）

スーパーキーとは、テーブル内の行を一意に識別できる属性（または属性の組み合わせ）です。

学校システムの学生テーブルを例に考えてみましょう：

```
CREATE TABLE students (
    student_id BIGINT,
    student_name VARCHAR(64),
    email VARCHAR(100),
    phone VARCHAR(20),
```

```
    admission_date DATE
);

-- サンプルデータ
INSERT INTO students VALUES
(301, '田中太郎', 'tanaka@example.com', '090-1234-5678', '2023-04-01'),
(302, '山田花子', 'yamada@example.com', '080-9876-5432', '2023-04-01'),
(303, '佐藤次郎', 'sato@example.com', '070-1111-2222', '2024-04-01');
```

このテーブルでのスーパーキーの例：

- `student_id`（学生IDは一意）
- `email`（メールアドレスは一意）
- `(student_id, student_name)`（学生IDが一意なので、組み合わせも一意）
- `(student_id, email)`（どちらも一意なので、組み合わせも一意）
- `(student_id, student_name, email)`（すべての組み合わせ）

2. 候補キー（Candidate Key）

候補キーとは、スーパーキーの中で**最小限の属性で構成される**ものです。つまり、どの属性を除いても一意性が保てなくなるキーです。

上記の学生テーブルでの候補キー：

- `student_id`（これだけで一意、除けるものがない）
- `email`（これだけで一意、除けるものがない）

候補キーではないもの：

- `(student_id, student_name)`（student_nameを除いてもstudent_idだけで一意）
- `(student_id, email)`（どちらかを除いても一意性が保てる）

3. 主キー（Primary Key）

主キーとは、候補キーの中から**1つだけ選ばれた**、そのテーブルの代表的なキーです。

```
-- 主キーの設定例
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,  -- 主キーとして選択
    student_name VARCHAR(64),
    email VARCHAR(100) UNIQUE,      -- 候補キーだが、主キーではない
    phone VARCHAR(20),
    admission_date DATE
);
```

主キーの特徴：

- **NOT NULL**：主キーにはNULL値を格納できません
- **UNIQUE**：主キーの値は一意でなければなりません

- **不変性**：主キーの値は変更しないことが推奨されます
- **テーブルごとに1つ**：1つのテーブルには1つの主キーのみ設定できます

4. 代替キー（Alternate Key）

代替キーとは、主キーに選ばれなかった候補キーのことです。

上記の例では：

- **主キー**：`student_id`
- **代替キー**：`email`

```
-- 代替キーにはUNIQUE制約を設定
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64),
    email VARCHAR(100) UNIQUE,  -- 代替キー（UNIQUE制約）
    phone VARCHAR(20),
    admission_date DATE
);
```

5. 外部キー（Foreign Key）

外部キーとは、他のテーブルの主キーを参照する属性です。テーブル間の関係を表現し、参照整合性を保証します。

```
-- 講座テーブル
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL  -- 外部キー（teachersテーブルを参照予定）
);

-- 教師テーブル
CREATE TABLE teachers (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64) NOT NULL
);

-- 外部キー制約の追加
ALTER TABLE courses
ADD CONSTRAINT fk_courses_teacher
FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
```

主キーの詳細

主キーの設計原則

1. 単純性（Simplicity）

可能な限り単一の列で構成する：

```
-- Good: 単一列の主キー
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(64)
);

-- 避けるべき: 不必要に複雑な複合主キー
CREATE TABLE students_bad (
    student_name VARCHAR(64),
    admission_date DATE,
    phone VARCHAR(20),
    PRIMARY KEY (student_name, admission_date, phone)  -- 複雑すぎる
);
```

2. 安定性（Stability）

主キーの値は変更されないことが理想：

```
-- Good: 学生IDは変更されない
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,  -- 安定
    student_name VARCHAR(64)        -- 名前は変更される可能性がある
);

-- 問題あり: 名前を主キーにすると変更時に問題
CREATE TABLE students_bad (
    student_name VARCHAR(64) PRIMARY KEY,  -- 結婚等で変更される可能性
    email VARCHAR(100)
);
```

3. 意味の有無

自然キー（Natural Key）

業務的な意味を持つキー：

```
-- 自然キーの例：学籍番号
CREATE TABLE students (
    student_number VARCHAR(20) PRIMARY KEY,  -- 2025001, 2025002...
    student_name VARCHAR(64)
);
```

利点：

- 業務上の意味があり、理解しやすい
- レポート等で直接使用できる

欠点：

- 番号体系の変更時に影響が大きい
- 複雑な番号体系の場合、パフォーマンスが劣る場合がある

代理キー（Surrogate Key）

業務的な意味を持たない、システム生成のキー：

```
-- 代理キーの例：自動増分ID
CREATE TABLE students (
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,  -- システム生成
    student_number VARCHAR(20) UNIQUE,             -- 業務上のキーは別途管理
    student_name VARCHAR(64)
);
```

利点：

- 安定性が高い（変更されない）
- パフォーマンスが良い（整数型）
- システム間の連携が容易

欠点：

- 業務的な意味がない
- 別途業務キーの管理が必要

複合主キー（Composite Primary Key）

複数の列を組み合わせて主キーとする場合：

```
-- 学生受講テーブル：学生と講座の組み合わせが主キー
CREATE TABLE student_courses (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (student_id, course_id),  -- 複合主キー
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

複合主キーの注意点：

- 各列は単独ではNULLを許可する場合でも、複合主キーとしてはNULLは不可
- インデックスのサイズが大きくなる傾向
- 外部キーとして参照される場合、参照側も複合外部キーが必要

主キーの実装パターン

パターン1: AUTO_INCREMENT を使用

```
CREATE TABLE students (
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_name VARCHAR(64) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- データ挿入時はIDを指定しない
INSERT INTO students (student_name) VALUES ('田中太郎');
INSERT INTO students (student_name) VALUES ('山田花子');

-- 結果確認
SELECT * FROM students;
-- student_id | student_name | created_at
-- 1          | 田中太郎     | 2025-05-27 10:00:00
-- 2          | 山田花子     | 2025-05-27 10:00:01
```

パターン2: UUID を使用

```
CREATE TABLE students (
    student_id CHAR(36) PRIMARY KEY,  -- UUID用
    student_name VARCHAR(64) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- UUID の生成（MySQL 8.0以降）
INSERT INTO students (student_id, student_name)
VALUES (UUID(), '田中太郎');

INSERT INTO students (student_id, student_name)
VALUES (UUID(), '山田花子');

-- 結果確認
SELECT * FROM students;
-- student_id                           | student_name | created_at
-- 12345678-1234-1234-1234-123456789abc | 田中太郎     | 2025-05-27 10:00:00
-- 87654321-4321-4321-4321-cba987654321 | 山田花子     | 2025-05-27 10:00:01
```

UUIDの利点：

- グローバルに一意

- 分散システムで有効
- セキュリティ面で推測されにくい

UUIDの欠点：

- ストレージサイズが大きい
- 人間には読みにくい
- インデックスの断片化が起きやすい

外部キーの詳細

外部キー制約の基本

```
-- 親テーブル（参照される側）
CREATE TABLE teachers (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64) NOT NULL,
    department VARCHAR(64)
);

-- 子テーブル（参照する側）
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT,  -- 外部キー

    CONSTRAINT fk_courses_teacher
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);
```

参照整合性の動作

1. INSERT 時の制約

```
-- 教師データの挿入
INSERT INTO teachers VALUES (101, '佐藤先生', '情報学部');

-- OK: 存在する教師を参照
INSERT INTO courses VALUES ('1', 'プログラミング基礎', 101);

-- ERROR: 存在しない教師を参照
INSERT INTO courses VALUES ('2', 'データベース', 999);
-- ERROR 1452: Cannot add or update a child row: a foreign key constraint fails
```

2. UPDATE 時の制約

```
-- ERROR: 存在しない教師IDに変更しようとする
UPDATE courses SET teacher_id = 999 WHERE course_id = '1';
-- ERROR 1452: Cannot add or update a child row: a foreign key constraint fails

-- OK: 存在する教師IDに変更
INSERT INTO teachers VALUES (102, '鈴木先生', '数学科');
UPDATE courses SET teacher_id = 102 WHERE course_id = '1';
```

3. DELETE 時の制約

```
-- ERROR: 参照されている教師を削除しようとする
DELETE FROM teachers WHERE teacher_id = 102;
-- ERROR 1451: Cannot delete or update a parent row: a foreign key constraint
fails
```

外部キー制約のオプション

ON DELETE / ON UPDATE オプション

```
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT,

    CONSTRAINT fk_courses_teacher
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
    ON DELETE CASCADE     -- 教師削除時、講座も削除
    ON UPDATE CASCADE     -- 教師ID変更時、講座も更新
);
```

オプションの種類：

CASCADE

親の削除/更新時に、子も削除/更新：

```
-- CASCADE の例
CREATE TABLE student_courses (
    student_id BIGINT,
    course_id VARCHAR(16),
    PRIMARY KEY (student_id, course_id),

    FOREIGN KEY (student_id) REFERENCES students(student_id)
    ON DELETE CASCADE,  -- 学生削除時、受講記録も削除
```

```
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
    ON DELETE CASCADE   -- 講座削除時、受講記録も削除
);

-- 学生を削除すると、その学生の受講記録もすべて削除される
DELETE FROM students WHERE student_id = 301;
```

SET NULL

親の削除/更新時に、子の外部キーをNULLに設定：

```
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT,  -- NULLを許可

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
    ON DELETE SET NULL  -- 教師削除時、teacher_idをNULLに
);

-- 教師を削除すると、その教師の講座のteacher_idがNULLになる
DELETE FROM teachers WHERE teacher_id = 101;
```

RESTRICT / NO ACTION

親の削除/更新を拒否（デフォルト）：

```
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
    ON DELETE RESTRICT  -- 明示的にRESTRICTを指定
);
```

SET DEFAULT

親の削除/更新時に、子の外部キーをデフォルト値に設定：

```
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT DEFAULT 0,  -- デフォルト値を設定

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
```

```
    ON DELETE SET DEFAULT
);
```

複合外部キー

```
-- 親テーブル：複合主キー
CREATE TABLE course_schedules (
    course_id VARCHAR(16),
    schedule_date DATE,
    period INT,
    classroom_id VARCHAR(16),

    PRIMARY KEY (course_id, schedule_date, period)
);

-- 子テーブル：複合外部キー
CREATE TABLE attendance (
    student_id BIGINT,
    course_id VARCHAR(16),
    schedule_date DATE,
    period INT,
    status ENUM('present', 'absent', 'late'),

    PRIMARY KEY (student_id, course_id, schedule_date, period),

    -- 複合外部キー
    FOREIGN KEY (course_id, schedule_date, period)
    REFERENCES course_schedules(course_id, schedule_date, period)
);
```

キー設計のベストプラクティス

1. 主キー設計の指針

業務要件による選択

```
-- ケース1: 学生管理（安定性重視）
CREATE TABLE students (
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,  -- 代理キー
    student_number VARCHAR(20) UNIQUE NOT NULL,    -- 自然キー（別途管理）
    student_name VARCHAR(64) NOT NULL
);

-- ケース2: ログテーブル（パフォーマンス重視）
CREATE TABLE access_logs (
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,  -- 代理キー（高速）
    user_id BIGINT NOT NULL,
    access_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    url VARCHAR(500)
);

-- ケース3: 関連テーブル（関係性明示）
CREATE TABLE student_courses (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (student_id, course_id),  -- 複合主キー（関係性を表現）
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

2. 外部キー設計の指針

適切な参照動作の選択

```
-- マスターデータ：RESTRICT（厳格な整合性）
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
    ON DELETE RESTRICT  -- 教師は簡単に削除させない
);

-- トランザクションデータ：CASCADE（自動削除）
CREATE TABLE enrollments (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE,

    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
    ON DELETE CASCADE,  -- 学生削除時、履修記録も削除
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
    ON DELETE CASCADE   -- 講座削除時、履修記録も削除
);

-- 履歴データ：SET NULL（関連保持）
CREATE TABLE grade_history (
    history_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT,
    course_id VARCHAR(16),
    teacher_id BIGINT,  -- 履歴なので教師削除後もNULLで保持
    grade CHAR(1),
    graded_date DATE,

    FOREIGN KEY (student_id) REFERENCES students(student_id)
```

```
    ON DELETE SET NULL,  -- 学生削除後も履歴は残す
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
    ON DELETE SET NULL   -- 教師削除後も履歴は残す
);
```

3. パフォーマンス考慮

インデックスの自動作成

```
-- 主キーには自動的にインデックスが作成される
CREATE TABLE students (
    student_id BIGINT PRIMARY KEY,  -- 自動的にインデックス作成
    student_name VARCHAR(64)
);

-- 外部キーにも明示的にインデックスを作成することを推奨
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128),
    teacher_id BIGINT,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id),
    INDEX idx_teacher_id (teacher_id)  -- 外部キー用インデックス
);
```

大量データでの考慮事項

```
-- 大量データのテーブルでは、主キーの型に注意
CREATE TABLE access_logs (
    -- BIGINT を使用（INTは約21億まで）
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    access_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_user_time (user_id, access_time)  -- 複合インデックス
);
```

学校システムでの実践例

完全なキー設計例

```
-- 1. 基本マスターテーブル

-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,      -- 代理キー
```

```
    student_number VARCHAR(20) UNIQUE NOT NULL,         -- 自然キー
    student_name VARCHAR(64) NOT NULL,
    email VARCHAR(100) UNIQUE,                          -- 代替キー
    phone VARCHAR(20),
    admission_date DATE NOT NULL,

    INDEX idx_student_number (student_number),
    INDEX idx_email (email)
);

-- 教師テーブル
CREATE TABLE teachers (
    teacher_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    teacher_code VARCHAR(20) UNIQUE NOT NULL,
    teacher_name VARCHAR(64) NOT NULL,
    department VARCHAR(64),
    email VARCHAR(100) UNIQUE,

    INDEX idx_teacher_code (teacher_code)
);

-- 講座テーブル
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,                  -- 自然キー
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,
    credits INT DEFAULT 1,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
    ON DELETE RESTRICT,                                 -- 教師削除防止

    INDEX idx_teacher_id (teacher_id)
);

-- 2. 関連テーブル

-- 学生受講テーブル（多対多の解決）
CREATE TABLE enrollments (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE DEFAULT CURRENT_DATE,
    status ENUM('active', 'dropped', 'completed') DEFAULT 'active',

    PRIMARY KEY (student_id, course_id),                -- 複合主キー

    FOREIGN KEY (student_id) REFERENCES students(student_id)
    ON DELETE CASCADE,                                  -- 学生削除時、履修も削除

    FOREIGN KEY (course_id) REFERENCES courses(course_id)
    ON DELETE CASCADE,                                  -- 講座削除時、履修も削除

    INDEX idx_enrollment_date (enrollment_date)
);
```

```
-- 成績テーブル
CREATE TABLE grades (
    grade_id BIGINT AUTO_INCREMENT PRIMARY KEY,        -- 代理キー
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    grade_type VARCHAR(32) NOT NULL,
    score DECIMAL(5,2) NOT NULL,
    max_score DECIMAL(5,2) NOT NULL,
    submission_date DATE DEFAULT CURRENT_DATE,

    -- 複合ユニーク制約（同じ学生の同じ講座の同じ種別は1つだけ）
    UNIQUE KEY uk_student_course_type (student_id, course_id, grade_type),

    FOREIGN KEY (student_id) REFERENCES students(student_id)
    ON DELETE CASCADE,

    FOREIGN KEY (course_id) REFERENCES courses(course_id)
    ON DELETE CASCADE,

    -- 外部キーと複合キーの確認制約
    FOREIGN KEY (student_id, course_id) REFERENCES enrollments(student_id,
course_id)
    ON DELETE CASCADE,

    INDEX idx_submission_date (submission_date),
    INDEX idx_student_course (student_id, course_id)
);

-- 3. ログ/履歴テーブル

-- アクセスログテーブル
CREATE TABLE user_access_logs (
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_type ENUM('student', 'teacher', 'admin') NOT NULL,
    user_id BIGINT,                                    -- NULLを許可（削除された場合）
    access_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    action VARCHAR(100),
    ip_address VARCHAR(45),

    INDEX idx_user_time (user_type, user_id, access_time),
    INDEX idx_access_time (access_time)
);

-- 成績変更履歴テーブル
CREATE TABLE grade_change_history (
    history_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    grade_id BIGINT,                                   -- NULLを許可（削除された場合）
    student_id BIGINT,
    course_id VARCHAR(16),
    old_score DECIMAL(5,2),
    new_score DECIMAL(5,2),
    changed_by BIGINT,
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    reason TEXT,
```

```
    FOREIGN KEY (grade_id) REFERENCES grades(grade_id)
    ON DELETE SET NULL,                                    -- 成績削除後も履歴は保持

    FOREIGN KEY (changed_by) REFERENCES teachers(teacher_id)
    ON DELETE SET NULL,                                    -- 教師削除後も履歴は保持

    INDEX idx_grade_changed_at (grade_id, changed_at),
    INDEX idx_student_changed_at (student_id, changed_at)
);
```

キー制約の確認方法

```
-- 主キー制約の確認
SELECT
    TABLE_NAME,
    COLUMN_NAME,
    CONSTRAINT_NAME
FROM information_schema.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'school_db'
AND CONSTRAINT_NAME = 'PRIMARY'
ORDER BY TABLE_NAME;

-- 外部キー制約の確認
SELECT
    TABLE_NAME,
    COLUMN_NAME,
    CONSTRAINT_NAME,
    REFERENCED_TABLE_NAME,
    REFERENCED_COLUMN_NAME,
    DELETE_RULE,
    UPDATE_RULE
FROM information_schema.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'school_db'
AND REFERENCED_TABLE_NAME IS NOT NULL
ORDER BY TABLE_NAME, CONSTRAINT_NAME;

-- UNIQUE制約の確認
SELECT
    TABLE_NAME,
    COLUMN_NAME,
    CONSTRAINT_NAME
FROM information_schema.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'school_db'
AND CONSTRAINT_NAME LIKE 'UK_%'
ORDER BY TABLE_NAME;
```

練習問題

問題44-1：キーの識別と分類

以下のテーブルを分析して、各種キーを識別してください：

```
CREATE TABLE employees (
    emp_id INT AUTO_INCREMENT,
    emp_code VARCHAR(20),
    emp_name VARCHAR(100),
    email VARCHAR(150),
    phone VARCHAR(20),
    department_id INT,
    hire_date DATE,

    PRIMARY KEY (emp_id),
    UNIQUE KEY uk_emp_code (emp_code),
    UNIQUE KEY uk_email (email),
    KEY idx_department (department_id)
);
```

質問：

1. 主キーは何ですか？
2. 候補キーをすべて挙げてください
3. 代替キーは何ですか？
4. このテーブルに外部キーはありますか？あるとすれば何ですか？
5. スーパーキーの例を3つ挙げてください

問題44-2：主キー設計の選択

オンラインショップの商品テーブルを設計しています。以下の2つの設計案のメリット・デメリットを分析してください：

案1：自然キー使用

```
CREATE TABLE products_v1 (
    product_code VARCHAR(20) PRIMARY KEY,  -- PRD-2025-001 形式
    product_name VARCHAR(200) NOT NULL,
    category_id INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    stock_quantity INT DEFAULT 0
);
```

案2：代理キー使用

```
CREATE TABLE products_v2 (
    product_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    product_code VARCHAR(20) UNIQUE NOT NULL,
    product_name VARCHAR(200) NOT NULL,
    category_id INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
```

```
    stock_quantity INT DEFAULT 0
);
```

課題：

1. 各案のメリット・デメリットを3つずつ挙げてください
2. どちらの設計を推奨しますか？理由も説明してください
3. 注文詳細テーブルで商品を参照する場合、どちらが良いか考察してください

問題44-3：外部キー制約の設計

図書館システムで以下の要件があります。適切な外部キー制約を設計してください：

要件：

- 利用者が削除された場合、その利用者の貸出履歴は残したい（統計のため）
- 本が削除された場合、その本の貸出履歴も削除したい
- 著者が削除された場合、その著者の本は削除したくない（著者情報をNULLにしたい）
- カテゴリが削除された場合、そのカテゴリの本の削除は拒否したい

現在のテーブル構造：

```
CREATE TABLE users (
    user_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_name VARCHAR(100) NOT NULL
);

CREATE TABLE authors (
    author_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    author_name VARCHAR(100) NOT NULL
);

CREATE TABLE categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(50) NOT NULL
);

CREATE TABLE books (
    book_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    author_id BIGINT,
    category_id INT NOT NULL
);

CREATE TABLE rentals (
    rental_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT,
    book_id BIGINT NOT NULL,
    rental_date DATE NOT NULL,
    return_date DATE
);
```

課題：

1. 各テーブルに適切な外部キー制約を追加してください
2. ON DELETE/ON UPDATE オプションを適切に設定してください
3. 設計の妥当性を検証するテストケースを作成してください

問題44-4：複合キーの設計

大学の時間割システムで、以下の要件を満たすテーブル設計を行ってください：

要件：

- 同じ教室の同じ時間帯には1つの授業のみ
- 同じ教員の同じ時間帯には1つの授業のみ
- 授業は「科目」「教員」「教室」「時間帯」の組み合わせで一意
- 学生は複数の授業を履修できる
- 同じ学生が同じ時間帯に複数の授業を履修することはできない

課題：

1. 必要なテーブルを設計してください
2. 適切な主キーと外部キーを設定してください
3. 複合キーが必要な場所を特定し、実装してください
4. 制約違反を防ぐためのUNIQUE制約を設計してください
5. テストデータを作成し、制約が正しく動作することを確認してください

問題44-5：キー制約の改善

以下の既存システムには設計上の問題があります。問題点を特定し、改善案を提示してください：

```
-- 既存の問題のあるテーブル設計
CREATE TABLE student_grades (
    id INT AUTO_INCREMENT PRIMARY KEY,
    student_name VARCHAR(100),
    student_email VARCHAR(150),
    course_name VARCHAR(200),
    teacher_name VARCHAR(100),
    grade_type VARCHAR(50),
    score DECIMAL(5,2),
    submission_date DATE
);

-- サンプルデータ
INSERT INTO student_grades VALUES
(1, '田中太郎', 'tanaka@example.com', 'プログラミング基礎', '佐藤先生', 'テスト', 85.0,
'2025-05-15'),
(2, '田中太郎', 'tanaka@example.com', 'プログラミング基礎', '佐藤先生', 'レポート', 90.0,
'2025-05-20'),
(3, '山田花子', 'yamada@example.com', 'データベース', '鈴木先生', 'テスト', 92.0, '2025-
05-15'),
```

```
(4, '田中太郎', 'tanaka@example.com', 'データベース', '鈴木先生', 'テスト', 78.0, '2025-
05-18');
```

課題：

1. 現在の設計の問題点を5つ以上指摘してください
2. 正規化とキー設計の観点から改善されたテーブル構造を提案してください
3. データ移行用のSQLを作成してください
4. 改善後の設計で同じデータを取得するクエリを書いてください

問題44-6：実践的なキー設計

ECサイトのシステム設計を行います。以下の要件を満たすキー設計を行ってください：

業務要件：

- 顧客は複数の住所を登録できる（配送先として）
- 商品には SKU（Stock Keeping Unit）コードがある
- 注文は複数の商品を含むことができる
- 注文には配送先住所が必要
- 支払い方法は複数登録できる
- 在庫は商品ごとに管理する
- 注文履歴は永続的に保持する
- 商品レビューシステムがある

技術要件：

- 高いパフォーマンスが求められる
- 将来的にシャーディング（分散）を検討
- データ整合性は重要
- 外部システムとの連携がある

課題：

1. 必要なテーブルをすべて洗い出してください
2. 各テーブルの主キー設計（自然キー vs 代理キー）を検討してください
3. テーブル間の関係と外部キー制約を設計してください
4. パフォーマンスを考慮したインデックス設計を行ってください
5. 将来のシャーディングを考慮したキー設計のポイントを説明してください

解答

解答44-1

1. **主キー**：`emp_id`
2. **候補キー**：
    - `emp_id`（主キー）
    - `emp_code`（UNIQUE制約）
    - `email`（UNIQUE制約）
3. **代替キー**：

    - `emp_code`
    - `email`
4. **外部キー**：`department_id`（参照先テーブルが明示されていないが、インデックスが設定されていることから外部キーと推測）
5. **スーパーキーの例**：
    - `emp_id`
    - `emp_code`
    - `email`
    - `(emp_id, emp_name)`
    - `(emp_code, hire_date)`
    - `(email, phone)`

解答44-2

1. **各案のメリット・デメリット**：

案1（自然キー）のメリット：

- 業務的な意味があり、理解しやすい
- レポートや画面で直接表示できる
- 外部システムとの連携時に意味がある

案1（自然キー）のデメリット：

- 商品コード体系変更時の影響が大きい
- 文字列型のため、整数型よりパフォーマンスが劣る
- 長いキーのため、外部キーとして参照される際のオーバーヘッド

案2（代理キー）のメリット：

- 高いパフォーマンス（整数型、AUTO_INCREMENT）
- 安定性（変更されない）
- 外部キー参照時の効率性

案2（代理キー）のデメリット：

- 業務的な意味がない
- 商品コードでの結合時に追加の処理が必要
- システム間の連携時に商品コードでの照合が必要

2. **推奨設計**：案2（代理キー使用）を推奨

理由：

- ECサイトでは大量の商品データとトランザクションが発生するため、パフォーマンスが重要
- 商品コードは業務上重要だが、UNIQUE制約で一意性を保証し、検索インデックスで効率性を確保
- 将来的な商品コード体系の変更に柔軟に対応可能

3. **注文詳細テーブルでの参照**：

```
-- 案2の方が効率的
CREATE TABLE order_items (
    order_id BIGINT,
    product_id BIGINT,  -- 整数型で高速
    quantity INT,
    unit_price DECIMAL(10,2),

    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (product_id) REFERENCES products_v2(product_id)
);

-- 商品コードでの検索も可能
SELECT oi.*, p.product_code, p.product_name
FROM order_items oi
JOIN products_v2 p ON oi.product_id = p.product_id
WHERE p.product_code = 'PRD-2025-001';
```

解答44-3

1. **適切な外部キー制約**：

```
-- 本テーブル
ALTER TABLE books
ADD CONSTRAINT fk_books_author
FOREIGN KEY (author_id) REFERENCES authors(author_id)
ON DELETE SET NULL      -- 著者削除時はNULLに
ON UPDATE CASCADE,

ADD CONSTRAINT fk_books_category
FOREIGN KEY (category_id) REFERENCES categories(category_id)
ON DELETE RESTRICT      -- カテゴリ削除は拒否
ON UPDATE CASCADE;

-- 貸出テーブル
ALTER TABLE rentals
ADD CONSTRAINT fk_rentals_user
FOREIGN KEY (user_id) REFERENCES users(user_id)
ON DELETE SET NULL      -- 利用者削除時はNULLに（履歴保持）
ON UPDATE CASCADE,

ADD CONSTRAINT fk_rentals_book
FOREIGN KEY (book_id) REFERENCES books(book_id)
ON DELETE CASCADE       -- 本削除時は貸出履歴も削除
ON UPDATE CASCADE;
```

2. **テストケース**：

```
-- テストデータの準備
INSERT INTO categories VALUES (1, 'プログラミング'), (2, '文学');
INSERT INTO authors VALUES (1, '山田太郎'), (2, '佐藤花子');
INSERT INTO users VALUES (1, '田中一郎'), (2, '鈴木次郎');
INSERT INTO books VALUES (1, 'SQL入門', 1, 1), (2, '小説集', 2, 2);
INSERT INTO rentals VALUES (1, 1, 1, '2025-05-01', '2025-05-15');

-- テスト1: 著者削除（SET NULL の確認）
DELETE FROM authors WHERE author_id = 1;
SELECT * FROM books WHERE book_id = 1;  -- author_id が NULL になる

-- テスト2: カテゴリ削除の拒否（RESTRICT の確認）
DELETE FROM categories WHERE category_id = 1;  -- エラーが発生すべき

-- テスト3: 利用者削除（履歴保持の確認）
DELETE FROM users WHERE user_id = 1;
SELECT * FROM rentals WHERE rental_id = 1;  -- user_id が NULL になるが履歴は残る

-- テスト4: 本削除（CASCADE の確認）
DELETE FROM books WHERE book_id = 1;
SELECT * FROM rentals WHERE book_id = 1;  -- 貸出履歴も削除される
```

解答44-4

1. **テーブル設計**：

```
-- 科目テーブル
CREATE TABLE subjects (
    subject_id VARCHAR(16) PRIMARY KEY,
    subject_name VARCHAR(128) NOT NULL,
    credits INT DEFAULT 1
);

-- 教員テーブル
CREATE TABLE instructors (
    instructor_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    instructor_name VARCHAR(100) NOT NULL,
    department VARCHAR(64)
);

-- 教室テーブル
CREATE TABLE classrooms (
    classroom_id VARCHAR(16) PRIMARY KEY,
    classroom_name VARCHAR(64) NOT NULL,
    capacity INT NOT NULL
);

-- 時間帯テーブル
CREATE TABLE time_slots (
    slot_id INT AUTO_INCREMENT PRIMARY KEY,
    day_of_week ENUM('Monday','Tuesday','Wednesday','Thursday','Friday') NOT NULL,
```

```
    period INT NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,

    UNIQUE KEY uk_day_period (day_of_week, period)
);

-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_number VARCHAR(20) UNIQUE NOT NULL
);
```

2. **授業テーブル（複合キーと制約）**：

```
-- 授業テーブル
CREATE TABLE classes (
    class_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    subject_id VARCHAR(16) NOT NULL,
    instructor_id BIGINT NOT NULL,
    classroom_id VARCHAR(16) NOT NULL,
    slot_id INT NOT NULL,
    academic_year YEAR NOT NULL,
    semester ENUM('spring', 'fall') NOT NULL,

    -- 外部キー制約
    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id),
    FOREIGN KEY (instructor_id) REFERENCES instructors(instructor_id),
    FOREIGN KEY (classroom_id) REFERENCES classrooms(classroom_id),
    FOREIGN KEY (slot_id) REFERENCES time_slots(slot_id),

    -- 重複防止のための制約
    UNIQUE KEY uk_classroom_time (classroom_id, slot_id, academic_year, semester),
    UNIQUE KEY uk_instructor_time (instructor_id, slot_id, academic_year,
semester),
    UNIQUE KEY uk_subject_instructor_time (subject_id, instructor_id,
classroom_id, slot_id, academic_year, semester)
);

-- 履修テーブル
CREATE TABLE enrollments (
    student_id BIGINT,
    class_id BIGINT,
    enrollment_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (student_id, class_id),  -- 複合主キー
    FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE,
    FOREIGN KEY (class_id) REFERENCES classes(class_id) ON DELETE CASCADE
);
```

3. **学生の時間重複防止**：

```
-- 学生の時間割重複チェックビュー
CREATE VIEW student_time_conflicts AS
SELECT
    e1.student_id,
    c1.slot_id,
    c1.academic_year,
    c1.semester,
    COUNT(*) as conflict_count
FROM enrollments e1
JOIN classes c1 ON e1.class_id = c1.class_id
GROUP BY e1.student_id, c1.slot_id, c1.academic_year, c1.semester
HAVING COUNT(*) > 1;

-- 履修登録時のチェック用プロシージャ
DELIMITER //
CREATE PROCEDURE check_enrollment_conflict(
    IN p_student_id BIGINT,
    IN p_class_id BIGINT
)
BEGIN
    DECLARE v_conflict_count INT DEFAULT 0;

    SELECT COUNT(*) INTO v_conflict_count
    FROM enrollments e
    JOIN classes c1 ON e.class_id = c1.class_id
    JOIN classes c2 ON p_class_id = c2.class_id
    WHERE e.student_id = p_student_id
    AND c1.slot_id = c2.slot_id
    AND c1.academic_year = c2.academic_year
    AND c1.semester = c2.semester;

    IF v_conflict_count > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Time slot conflict detected';
    END IF;
END //
DELIMITER ;
```

4. **テストデータと検証**：

```
-- テストデータの投入
INSERT INTO subjects VALUES ('CS101', 'プログラミング基礎', 2);
INSERT INTO instructors VALUES (1, '佐藤先生', 'コンピュータ科学');
INSERT INTO classrooms VALUES ('A101', 'コンピュータ室1', 30);
INSERT INTO time_slots VALUES (1, 'Monday', 1, '09:00', '10:30');
INSERT INTO time_slots VALUES (2, 'Monday', 2, '10:40', '12:10');
INSERT INTO students VALUES (1, '田中太郎', 'S2025001');

-- 授業の作成
```

```
INSERT INTO classes VALUES (1, 'CS101', 1, 'A101', 1, 2025, 'spring');

-- 制約違反のテスト
-- 同じ教室・時間に別の授業を作ろうとする（エラーが発生すべき）
INSERT INTO classes VALUES (2, 'CS101', 1, 'A101', 1, 2025, 'spring');

-- 履修登録
INSERT INTO enrollments VALUES (1, 1);
```

解答44-5

1. **現在の設計の問題点**：
    - **非正規化**：学生情報、講座情報、教師情報が重複格納
    - **主キーの不適切性**：id は業務的意味がない単純な連番
    - **一意性制約の欠如**：同じ学生の同じ講座の同じ種別で複数レコード可能
    - **外部キー制約なし**：データ整合性が保証されない
    - **更新・削除異常**：学生名や講座名変更時の不整合リスク
    - **検索効率の悪さ**：適切なインデックスがない
    - **データ型の不整合**：学生IDや講座IDが文字列として格納
2. **改善されたテーブル構造**：

```
-- 学生テーブル
CREATE TABLE students (
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(150) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 教師テーブル
CREATE TABLE teachers (
    teacher_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    teacher_name VARCHAR(100) NOT NULL,
    department VARCHAR(64),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 講座テーブル
CREATE TABLE courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(200) NOT NULL,
    teacher_id BIGINT NOT NULL,
    credits INT DEFAULT 1,

    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);

-- 履修テーブル
CREATE TABLE enrollments (
```

```
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);

-- 成績テーブル
CREATE TABLE grades (
    student_id BIGINT,
    course_id VARCHAR(16),
    grade_type VARCHAR(50),
    score DECIMAL(5,2) NOT NULL,
    submission_date DATE DEFAULT CURRENT_DATE,

    PRIMARY KEY (student_id, course_id, grade_type),
    FOREIGN KEY (student_id, course_id) REFERENCES enrollments(student_id,
course_id) ON DELETE CASCADE,

    INDEX idx_submission_date (submission_date)
);
```

3. **データ移行SQL**：

```
-- 一時テーブルでデータを整理
CREATE TEMPORARY TABLE temp_migration AS
SELECT DISTINCT
    student_name,
    student_email,
    course_name,
    teacher_name
FROM student_grades;

-- 教師データの移行
INSERT INTO teachers (teacher_name)
SELECT DISTINCT teacher_name
FROM temp_migration;

-- 学生データの移行
INSERT INTO students (student_name, student_email)
SELECT DISTINCT student_name, student_email
FROM temp_migration;

-- 講座データの移行
INSERT INTO courses (course_id, course_name, teacher_id)
SELECT
    CONCAT('C', LPAD(ROW_NUMBER() OVER (ORDER BY tm.course_name), 3, '0')) as
course_id,
    tm.course_name,
    t.teacher_id
```

```
FROM (SELECT DISTINCT course_name, teacher_name FROM temp_migration) tm
JOIN teachers t ON tm.teacher_name = t.teacher_name;

-- 履修データの移行
INSERT INTO enrollments (student_id, course_id)
SELECT DISTINCT s.student_id, c.course_id
FROM student_grades sg
JOIN students s ON sg.student_name = s.student_name AND sg.student_email =
s.student_email
JOIN courses c ON sg.course_name = c.course_name
JOIN teachers t ON sg.teacher_name = t.teacher_name AND c.teacher_id =
t.teacher_id;

-- 成績データの移行
INSERT INTO grades (student_id, course_id, grade_type, score, submission_date)
SELECT
    s.student_id,
    c.course_id,
    sg.grade_type,
    sg.score,
    sg.submission_date
FROM student_grades sg
JOIN students s ON sg.student_name = s.student_name AND sg.student_email =
s.student_email
JOIN courses c ON sg.course_name = c.course_name
JOIN teachers t ON sg.teacher_name = t.teacher_name AND c.teacher_id =
t.teacher_id;
```

4. **改善後のクエリ**：

```
-- 改善後：学生の成績を取得
SELECT
    s.student_name,
    c.course_name,
    t.teacher_name,
    g.grade_type,
    g.score,
    g.submission_date
FROM students s
JOIN grades g ON s.student_id = g.student_id
JOIN courses c ON g.course_id = c.course_id
JOIN teachers t ON c.teacher_id = t.teacher_id
WHERE s.student_name = '田中太郎'
ORDER BY g.submission_date DESC;

-- 特定講座の全学生成績
SELECT
    s.student_name,
    g.grade_type,
    g.score,
    g.submission_date
FROM grades g
```

```
JOIN students s ON g.student_id = s.student_id
JOIN courses c ON g.course_id = c.course_id
WHERE c.course_name = 'プログラミング基礎'
ORDER BY s.student_name, g.grade_type;
```

解答44-6

1. **必要なテーブル**：

```
-- 顧客テーブル
CREATE TABLE customers (
    customer_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    customer_uuid CHAR(36) UNIQUE NOT NULL,  -- 外部連携用
    email VARCHAR(150) UNIQUE NOT NULL,
    customer_name VARCHAR(100) NOT NULL,
    phone VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_email (email),
    INDEX idx_uuid (customer_uuid)
);

-- 顧客住所テーブル
CREATE TABLE customer_addresses (
    address_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    customer_id BIGINT NOT NULL,
    address_type ENUM('billing', 'shipping', 'both') DEFAULT 'shipping',
    address_name VARCHAR(50),  -- 「自宅」「会社」など
    postal_code VARCHAR(10),
    address_line1 VARCHAR(200) NOT NULL,
    address_line2 VARCHAR(200),
    city VARCHAR(50) NOT NULL,
    state VARCHAR(50),
    country VARCHAR(50) DEFAULT 'Japan',
    is_default BOOLEAN DEFAULT FALSE,

    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE,
    INDEX idx_customer_type (customer_id, address_type)
);

-- カテゴリテーブル
CREATE TABLE categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(100) NOT NULL,
    parent_category_id INT,

    FOREIGN KEY (parent_category_id) REFERENCES categories(category_id),
    INDEX idx_parent (parent_category_id)
);

-- 商品テーブル
```

```sql
CREATE TABLE products (
    product_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    sku_code VARCHAR(50) UNIQUE NOT NULL,  -- 外部システム連携用
    product_name VARCHAR(200) NOT NULL,
    category_id INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    description TEXT,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (category_id) REFERENCES categories(category_id),
    INDEX idx_sku (sku_code),
    INDEX idx_category_active (category_id, is_active),
    INDEX idx_price (price)
);

-- 在庫テーブル
CREATE TABLE inventory (
    product_id BIGINT PRIMARY KEY,
    quantity INT DEFAULT 0,
    reserved_quantity INT DEFAULT 0,  -- 注文済み未出荷分
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    FOREIGN KEY (product_id) REFERENCES products(product_id) ON DELETE CASCADE
);

-- 支払い方法テーブル
CREATE TABLE payment_methods (
    payment_method_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    customer_id BIGINT NOT NULL,
    method_type ENUM('credit_card', 'bank_transfer', 'paypal', 'cash_on_delivery')
NOT NULL,
    method_name VARCHAR(50),  -- 「メインのカード」など
    is_default BOOLEAN DEFAULT FALSE,
    external_id VARCHAR(100),  -- 外部決済システムのID
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE,
    INDEX idx_customer_default (customer_id, is_default)
);

-- 注文テーブル
CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    order_number VARCHAR(32) UNIQUE NOT NULL,  -- 注文番号（顧客向け）
    customer_id BIGINT NOT NULL,
    shipping_address_id BIGINT NOT NULL,
    payment_method_id BIGINT NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status ENUM('pending', 'confirmed', 'shipped', 'delivered', 'cancelled')
DEFAULT 'pending',
    subtotal DECIMAL(12,2) NOT NULL,
    tax_amount DECIMAL(10,2) DEFAULT 0,
    shipping_fee DECIMAL(8,2) DEFAULT 0,
```

```
    total_amount DECIMAL(12,2) NOT NULL,

    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (shipping_address_id) REFERENCES customer_addresses(address_id),
    FOREIGN KEY (payment_method_id) REFERENCES payment_methods(payment_method_id),

    INDEX idx_customer_date (customer_id, order_date),
    INDEX idx_order_number (order_number),
    INDEX idx_status_date (status, order_date)
);

-- 注文詳細テーブル
CREATE TABLE order_items (
    order_id BIGINT,
    product_id BIGINT,
    quantity INT NOT NULL,
    unit_price DECIMAL(10,2) NOT NULL,  -- 注文時の価格を保存
    total_price DECIMAL(12,2) NOT NULL,

    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

-- レビューテーブル
CREATE TABLE product_reviews (
    review_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    product_id BIGINT NOT NULL,
    customer_id BIGINT NOT NULL,
    order_id BIGINT,  -- 購入履歴との紐付け
    rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),
    title VARCHAR(100),
    comment TEXT,
    is_verified_purchase BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),

    UNIQUE KEY uk_customer_product_order (customer_id, product_id, order_id),
    INDEX idx_product_rating (product_id, rating),
    INDEX idx_created_at (created_at)
);
```
