==========

32. テーブル作成：CREATE TABLE文

はじめに

前章までで、データの操作と管理について詳しく学習しました。第6章からは、「データ定義言語（DDL：Data Definition Language）」について学習していきます。DDLは、データベースの構造そのものを定義・変更するためのSQL文群です。

この章では、データベースの基礎となる「テーブル」を作成するための「CREATE TABLE文」について学習します。CREATE TABLE文は、データベースに新しいテーブルを作成し、そのテーブルの構造（カラム名、データ型、制約など）を定義するためのSQL文です。

CREATE TABLE文が必要となる場面の例：

- 「新しい学校システムを構築するため、学生テーブルを作成したい」
- 「既存システムに新機能を追加するため、新しいテーブルが必要」
- 「データ移行のため、一時的なテーブルを作成したい」
- 「レポート用の集計テーブルを作成したい」
- 「ログデータを保存するためのテーブルを作成したい」
- 「テスト環境用のテーブルを作成したい」

この章では、CREATE TABLE文の基本構文から、データ型の選択、制約の設定、実践的な設計方法まで詳しく学んでいきます。

CREATE TABLE文とは

CREATE TABLE文は、データベースに新しいテーブルを作成するためのSQL文です。テーブルの構造（スキーマ）を定義し、どのような種類のデータをどのような形式で格納するかを決定します。

> **用語解説**：
>
> - **DDL（Data Definition Language）**：データベースの構造を定義するためのSQL文の分類です。CREATE、ALTER、DROPなどが含まれます。
> - **テーブル**：データベース内でデータを格納する基本的な構造で、行（レコード）と列（カラム）で構成されます。
> - **スキーマ**：データベースやテーブルの構造定義のことです。
> - **カラム（列）**：テーブル内の縦方向のデータ項目です。例：学生名、学生ID など。
> - **データ型**：各カラムに格納できるデータの種類を指定します。例：整数型、文字列型、日付型など。
> - **制約（Constraint）**：カラムに格納されるデータの条件や規則を定義します。例：NOT NULL、PRIMARY KEYなど。
> - **主キー（Primary Key）**：テーブル内の各行を一意に識別するためのカラムまたはカラムの組み合わせです。
> - **外部キー（Foreign Key）**：他のテーブルの主キーを参照するキーで、テーブル間の関連性を定義します。

CREATE TABLE文の基本構文

基本的な構文

```
CREATE TABLE テーブル名 (
    カラム名1 データ型 [制約],
    カラム名2 データ型 [制約],
    カラム名3 データ型 [制約],
    ...
    [テーブル制約]
);
```

構文の要素

- **テーブル名**：作成するテーブルの名前を指定します
- **カラム名**：テーブル内の各カラム（列）の名前を指定します
- **データ型**：そのカラムに格納できるデータの種類を指定します
- **制約**：データの整合性を保つための規則を指定します（オプション）

MySQLの主要データ型

CREATE TABLE文では、各カラムのデータ型を適切に選択することが重要です。

1. 数値型

データ型	説明	格納範囲	使用例
TINYINT	小さな整数	-128 ～ 127	フラグ、ステータス
INT/INTEGER	整数	-2,147,483,648 ～ 2,147,483,647	ID、数量
BIGINT	大きな整数	-9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807	大きなID
DECIMAL(M,D)	固定小数点数	M：全体桁数、D：小数点以下桁数	金額、正確な数値
FLOAT	単精度浮動小数点数	約7桁の精度	概算値
DOUBLE	倍精度浮動小数点数	約15桁の精度	科学計算

2. 文字列型

データ型	説明	最大長	使用例
CHAR(n)	固定長文字列	255文字	固定長コード
VARCHAR(n)	可変長文字列	65,535文字	名前、説明

データ型	説明	最大長	使用例
TEXT	長い文字列	65,535文字	長い説明文
MEDIUMTEXT	より長い文字列	16,777,215文字	記事本文
LONGTEXT	非常に長い文字列	4,294,967,295文字	大容量テキスト

3. 日付・時刻型

データ型	説明	形式	使用例
DATE	日付	YYYY-MM-DD	生年月日、開始日
TIME	時刻	HH:MM:SS	開始時刻、終了時刻
DATETIME	日付と時刻	YYYY-MM-DD HH:MM:SS	登録日時、更新日時
TIMESTAMP	タイムスタンプ	YYYY-MM-DD HH:MM:SS	自動更新される日時
YEAR	年	YYYY	年度、卒業年

4. その他の型

データ型	説明	使用例
BOOLEAN/BOOL	真偽値	フラグ（TRUE/FALSE）
ENUM('値1','値2',...)	列挙型	ステータス（'active','inactive'）
JSON	JSON形式データ	設定情報、メタデータ

基本的なテーブル作成例

例1：シンプルな学生テーブル

```
-- 基本的な学生テーブルの作成
CREATE TABLE simple_students (
    student_id INT,
    student_name VARCHAR(100),
    birth_date DATE,
    email VARCHAR(255)
);

-- テーブル構造の確認
DESCRIBE simple_students;

-- または
SHOW CREATE TABLE simple_students;
```

例2：制約を含む教師テーブル

```
-- 制約付きの教師テーブル
CREATE TABLE teachers_new (
    teacher_id BIGINT NOT NULL,
    teacher_name VARCHAR(64) NOT NULL,
    email VARCHAR(255) UNIQUE,
    hire_date DATE,
    department VARCHAR(50) DEFAULT '未配属',
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

例3：主キーと外部キーを含むテーブル

```
-- 講座テーブル（外部キー制約付き）
CREATE TABLE courses_new (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,
    credits INT DEFAULT 2,
    max_students INT DEFAULT 30,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    -- 外部キー制約
    FOREIGN KEY (teacher_id) REFERENCES teachers_new(teacher_id)
);
```

制約の詳細解説

1. カラム制約

NOT NULL制約

```
-- 必須項目の定義
CREATE TABLE required_data_example (
    id INT NOT NULL,                    -- 必須
    name VARCHAR(100) NOT NULL,         -- 必須
    description VARCHAR(500),           -- オプション（NULLを許可）
    created_at TIMESTAMP NOT NULL       -- 必須
);
```

DEFAULT制約

```
-- デフォルト値の設定
CREATE TABLE default_values_example (
```

```
    id INT NOT NULL,
    status VARCHAR(20) DEFAULT 'active',                    -- 文字列のデフォルト
    priority INT DEFAULT 1,                                 -- 数値のデフォルト
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,         -- 現在日時
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
               ON UPDATE CURRENT_TIMESTAMP,                 -- 更新時に自動更新
    is_enabled BOOLEAN DEFAULT TRUE                         -- 真偽値のデフォルト
);
```

UNIQUE制約

```
-- 一意性制約の設定
CREATE TABLE unique_constraint_example (
    id INT PRIMARY KEY,
    email VARCHAR(255) UNIQUE,                              -- 単一カラムの一意性
    username VARCHAR(50) UNIQUE,
    phone VARCHAR(20),

    -- 複数カラムの組み合わせ一意性
    UNIQUE KEY unique_phone_user (phone, username)
);
```

2. テーブル制約

PRIMARY KEY制約

```
-- 主キーの設定方法

-- 方法1：カラム定義時に指定
CREATE TABLE primary_key_example1 (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

-- 方法2：テーブル制約として指定
CREATE TABLE primary_key_example2 (
    id INT NOT NULL,
    name VARCHAR(100) NOT NULL,

    PRIMARY KEY (id)
);

-- 方法3：複合主キー
CREATE TABLE composite_key_example (
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    enrollment_date DATE NOT NULL,
```

```
    PRIMARY KEY (student_id, course_id)
);
```

FOREIGN KEY制約

```
-- 外部キー制約の設定
CREATE TABLE enrollment_example (
    enrollment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    enrollment_date DATE NOT NULL,
    grade DECIMAL(3,1),

    -- 外部キー制約の定義
    FOREIGN KEY (student_id) REFERENCES simple_students(student_id)
        ON DELETE CASCADE                    -- 参照先削除時に連動削除
        ON UPDATE CASCADE,                   -- 参照先更新時に連動更新

    FOREIGN KEY (course_id) REFERENCES courses_new(course_id)
        ON DELETE RESTRICT                   -- 参照先削除を禁止
        ON UPDATE RESTRICT                   -- 参照先更新を禁止
);
```

CHECK制約（MySQL 8.0以降）

```
-- CHECK制約の使用例
CREATE TABLE student_grades_example (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    subject VARCHAR(100) NOT NULL,
    score DECIMAL(5,2) NOT NULL,
    semester INT NOT NULL,
    academic_year INT NOT NULL,

    -- CHECK制約の定義
    CHECK (score >= 0 AND score <= 100),                    -- 点数の範囲制限
    CHECK (semester IN (1, 2)),                             -- 学期の値制限
    CHECK (academic_year >= 2000 AND academic_year <= 2100) -- 年度の範囲制限
);
```

AUTO_INCREMENTの使用

基本的な使用方法

```
-- 自動連番の設定
CREATE TABLE auto_increment_example (
```

```
    id BIGINT AUTO_INCREMENT PRIMARY KEY,       -- 自動増加する主キー
    name VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- データ挿入時（IDは自動設定される）
INSERT INTO auto_increment_example (name) VALUES ('テストデータ1');
INSERT INTO auto_increment_example (name) VALUES ('テストデータ2');

-- 結果確認
SELECT * FROM auto_increment_example;
```

開始値の設定

```
-- 連番の開始値を指定
CREATE TABLE custom_auto_increment (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    description VARCHAR(200)
) AUTO_INCREMENT = 1000;                        -- 1000から開始

-- 既存テーブルの開始値変更
ALTER TABLE auto_increment_example AUTO_INCREMENT = 100;
```

実践的なテーブル作成例

例4：学校管理システムの完全な学生テーブル

```
-- 実用的な学生テーブル
CREATE TABLE students_complete (
    -- 基本情報
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_number VARCHAR(20) UNIQUE NOT NULL,           -- 学籍番号
    first_name VARCHAR(50) NOT NULL,                      -- 名
    last_name VARCHAR(50) NOT NULL,                       -- 姓
    first_name_kana VARCHAR(100),                         -- 名（カナ）
    last_name_kana VARCHAR(100),                          -- 姓（カナ）

    -- 個人情報
    birth_date DATE,
    gender ENUM('male', 'female', 'other', 'prefer_not_to_say'),
    nationality VARCHAR(50) DEFAULT '日本',

    -- 連絡先情報
    email VARCHAR(255) UNIQUE,
    phone VARCHAR(20),
    emergency_contact VARCHAR(100),
    emergency_phone VARCHAR(20),

    -- 住所情報
```

```
    postal_code VARCHAR(10),
    prefecture VARCHAR(20),
    city VARCHAR(50),
    address_line VARCHAR(200),

    -- 学籍情報
    admission_date DATE NOT NULL,
    graduation_date DATE,
    status ENUM('enrolled', 'graduated', 'withdrawn', 'suspended') DEFAULT
'enrolled',
    class_year INT,
    major VARCHAR(100),

    -- システム情報
    is_active BOOLEAN DEFAULT TRUE,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    -- 制約
    CHECK (birth_date <= CURRENT_DATE),                    -- 生年月日は現在日以前
    CHECK (admission_date >= '2000-01-01'),               -- 入学日は2000年以降
    CHECK (class_year >= 1 AND class_year <= 6),          -- 学年は1-6年

    -- インデックス
    INDEX idx_student_number (student_number),
    INDEX idx_name (last_name, first_name),
    INDEX idx_admission_date (admission_date),
    INDEX idx_status (status)
);
```

例5：授業評価テーブル

```
-- 授業評価システムテーブル
CREATE TABLE course_evaluations (
    evaluation_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    teacher_id BIGINT NOT NULL,

    -- 評価項目（1-5のスコア）
    content_quality TINYINT,                               -- 内容の質
    teaching_method TINYINT,                               -- 教授法
    difficulty_level TINYINT,                              -- 難易度
    workload TINYINT,                                      -- 課題量
    overall_satisfaction TINYINT,                          -- 総合満足度

    -- 自由記述
    good_points TEXT,                                      -- 良かった点
    improvement_points TEXT,                               -- 改善点
```

```
    -- 推奨度
    would_recommend BOOLEAN,

    -- システム情報
    evaluation_date DATE NOT NULL,
    is_anonymous BOOLEAN DEFAULT TRUE,
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- 制約
    CHECK (content_quality >= 1 AND content_quality <= 5),
    CHECK (teaching_method >= 1 AND teaching_method <= 5),
    CHECK (difficulty_level >= 1 AND difficulty_level <= 5),
    CHECK (workload >= 1 AND workload <= 5),
    CHECK (overall_satisfaction >= 1 AND overall_satisfaction <= 5),

    -- 外部キー制約
    FOREIGN KEY (student_id) REFERENCES students_complete(student_id) ON DELETE
CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses_new(course_id) ON DELETE CASCADE,
    FOREIGN KEY (teacher_id) REFERENCES teachers_new(teacher_id) ON DELETE
CASCADE,

    -- 一意性制約（1学生1講座1回のみ評価可能）
    UNIQUE KEY unique_evaluation (student_id, course_id),

    -- インデックス
    INDEX idx_course_evaluation (course_id, evaluation_date),
    INDEX idx_teacher_evaluation (teacher_id, evaluation_date)
);
```

テーブル作成時の注意点

1. 命名規則

```
-- 良い命名例
CREATE TABLE student_course_enrollments (     -- わかりやすいテーブル名
    enrollment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,               -- 一貫した命名
    course_id VARCHAR(16) NOT NULL,
    enrollment_date DATE NOT NULL
);

-- 避けるべき命名例
CREATE TABLE t1 (                             -- 意味不明なテーブル名
    id INT,                                   -- あいまいなカラム名
    data VARCHAR(100),                        -- 汎用的すぎるカラム名
    dt DATETIME                               -- 略語
);
```

2. データ型の適切な選択

```
-- 適切なデータ型の選択例
CREATE TABLE data_type_best_practices (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,                -- 将来の拡張を考慮
    status ENUM('active', 'inactive', 'pending'),        -- 固定値はENUM
    score DECIMAL(5,2),                                  -- 金額や正確な数値
    description VARCHAR(500),                            -- 適切な長さ設定
    large_text MEDIUMTEXT,                               -- 容量に応じた選択
    flag BOOLEAN,                                        -- 真偽値はBOOLEAN
    percentage TINYINT UNSIGNED,                         -- 0-255の範囲で十分

    CHECK (percentage <= 100)                            -- 範囲制限
);
```

3. 制約の適切な設定

```
-- 制約設定のベストプラクティス
CREATE TABLE constraint_best_practices (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    -- 必須項目には NOT NULL
    name VARCHAR(100) NOT NULL,

    -- 一意性が必要な項目には UNIQUE
    email VARCHAR(255) UNIQUE NOT NULL,

    -- デフォルト値の設定
    status VARCHAR(20) DEFAULT 'active',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- 範囲制限
    age TINYINT UNSIGNED,
    CHECK (age >= 0 AND age <= 150),

    -- 外部キー制約
    category_id INT,
    FOREIGN KEY (category_id) REFERENCES categories(id)
        ON DELETE SET NULL                               -- 参照先削除時はNULLに設定
        ON UPDATE CASCADE                                -- 参照先更新時は連動更新
);
```

エラーと対処法

1. 既存テーブル名の重複

```
-- エラー例
CREATE TABLE students (                          -- 既存のテーブル名
    id INT PRIMARY KEY,
```

```
    name VARCHAR(100)
);
-- エラー: Table 'students' already exists

-- 対処法1: IF NOT EXISTS を使用
CREATE TABLE IF NOT EXISTS students_backup (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);

-- 対処法2: 異なるテーブル名を使用
CREATE TABLE students_new_version (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);
```

2. データ型の不適切な使用

```
-- エラー例
CREATE TABLE data_type_error (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR,                           -- 長さ指定なし
    score DECIMAL,                          -- 精度指定なし
    date_value DATE(10)                     -- 不要な長さ指定
);

-- 正しい書き方
CREATE TABLE data_type_correct (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),                      -- 適切な長さ指定
    score DECIMAL(5,2),                     -- 精度指定
    date_value DATE                         -- 長さ指定不要
);
```

3. 外部キー制約エラー

```
-- エラー例：参照先テーブルが存在しない
CREATE TABLE enrollment_error (
    student_id INT,
    course_id VARCHAR(16),
    FOREIGN KEY (student_id) REFERENCES non_existent_table(id)  -- 存在しないテーブル
);

-- 対処法：参照先テーブルを先に作成
CREATE TABLE students_ref (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);
```

```
CREATE TABLE enrollment_correct (
    student_id INT,
    course_id VARCHAR(16),
    FOREIGN KEY (student_id) REFERENCES students_ref(id)
);
```

テーブル作成のベストプラクティス

1. 段階的なテーブル設計

```
-- Phase 1: 基本テーブル
CREATE TABLE products_basic (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(200) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Phase 2: カテゴリテーブル追加
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT
);

-- Phase 3: 関連付け
ALTER TABLE products_basic
ADD COLUMN category_id INT,
ADD FOREIGN KEY (category_id) REFERENCES categories(id);
```

2. 将来の拡張を考慮した設計

```
-- 拡張性を考慮したテーブル設計
CREATE TABLE future_proof_design (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,          -- 大きなIDに対応
    uuid CHAR(36) UNIQUE,                          -- UUID対応
    version INT DEFAULT 1,                         -- バージョン管理

    -- JSONカラムで柔軟な属性管理
    attributes JSON,

    -- 論理削除対応
    is_deleted BOOLEAN DEFAULT FALSE,
    deleted_at TIMESTAMP NULL,

    -- 監査ログ対応
    created_by BIGINT,
    updated_by BIGINT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

3. ドキュメント化

```
-- コメント付きテーブル作成
CREATE TABLE documented_table (
    id BIGINT AUTO_INCREMENT PRIMARY KEY COMMENT '主キー（自動増加）',
    user_id BIGINT NOT NULL COMMENT 'ユーザーID（usersテーブルの外部キー）',
    title VARCHAR(200) NOT NULL COMMENT 'タイトル（最大200文字）',
    content TEXT COMMENT '内容（無制限）',
    status ENUM('draft', 'published', 'archived') DEFAULT 'draft' COMMENT 'ステータス',
    view_count INT UNSIGNED DEFAULT 0 COMMENT '閲覧数',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '作成日時',
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新日時',

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
) COMMENT = 'ブログ記事テーブル';
```

練習問題

問題32-1：基本的なテーブル作成

教室管理システム用の「教室テーブル（classrooms_practice）」を作成してください。以下のカラムを含めてください：

- room_id: 教室ID（整数、主キー、自動増加）
- room_name: 教室名（文字列、最大50文字、必須）
- building: 建物名（文字列、最大30文字）
- floor: 階数（小さな整数）
- capacity: 収容人数（整数、デフォルト値30）
- has_projector: プロジェクター有無（真偽値、デフォルト値FALSE）

問題32-2：制約付きテーブル作成

図書管理システム用の「図書テーブル（books）」を作成してください。以下の要件を満たしてください：

- book_id: 図書ID（大きな整数、主キー、自動増加）
- isbn: ISBN番号（文字列、最大20文字、一意、必須）
- title: 書籍名（文字列、最大200文字、必須）
- author: 著者名（文字列、最大100文字、必須）
- publisher: 出版社（文字列、最大100文字）
- publication_year: 出版年（整数、1900年以降2100年以下の制約）
- price: 価格（小数点2桁まで）
- category: カテゴリ（'fiction', 'non-fiction', 'textbook', 'reference'のいずれか）
- is_available: 貸出可能フラグ（真偽値、デフォルト値TRUE）

- created_at: 登録日時（タイムスタンプ、現在日時がデフォルト）

問題32-3：外部キー制約付きテーブル作成

図書貸出管理システム用の「貸出テーブル（book_loans）」を作成してください。前問のbooksテーブルと、既存のstudentsテーブルを参照する外部キーを含めてください：

- loan_id: 貸出ID（大きな整数、主キー、自動増加）
- book_id: 図書ID（外部キー、booksテーブル参照、必須）
- student_id: 学生ID（外部キー、studentsテーブル参照、必須）
- loan_date: 貸出日（日付、必須）
- due_date: 返却予定日（日付、必須）
- return_date: 返却日（日付、NULL許可）
- status: ステータス（'borrowed', 'returned', 'overdue'のいずれか、デフォルト'borrowed'）
- 制約：due_dateはloan_dateより後の日付であること
- 制約：return_dateが設定されている場合、loan_date以降であること

問題32-4：複合的なテーブル設計

時間割管理システム用の「時間割テーブル（class_schedules）」を作成してください：

- schedule_id: スケジュールID（大きな整数、主キー、自動増加）
- course_id: 講座ID（文字列、既存のcoursesテーブル参照）
- classroom_id: 教室ID（前問で作成したclassrooms_practiceテーブル参照）
- day_of_week: 曜日（1=月曜日～7=日曜日、1-7の範囲制約）
- period: 時限（1-8の範囲制約）
- start_time: 開始時刻（時刻型）
- end_time: 終了時刻（時刻型）
- semester: 学期（'spring', 'summer', 'fall', 'winter'のいずれか）
- academic_year: 年度（整数、2000年以降の制約）
- is_active: 有効フラグ（真偽値、デフォルトTRUE）
- 制約：end_timeはstart_timeより後の時刻であること
- 制約：同じ教室、同じ曜日、同じ時限、同じ学期、同じ年度の組み合わせは一意であること

問題32-5：JSON型を使用したテーブル作成

学生の追加情報を管理する「学生プロフィールテーブル（student_profiles）」を作成してください：

- profile_id: プロフィールID（大きな整数、主キー、自動増加）
- student_id: 学生ID（外部キー、studentsテーブル参照、必須、一意）
- hobbies: 趣味（JSON型）
- skills: スキル（JSON型）
- emergency_contacts: 緊急連絡先（JSON型）
- preferences: 設定（JSON型、デフォルト値は空のJSONオブジェクト'{}'）
- last_updated: 最終更新日時（タイムスタンプ、更新時に自動更新）

問題32-6：包括的なテーブル設計

成績管理システムの拡張として「詳細成績テーブル（detailed_grades）」を作成してください。以下の要件をすべて満たしてください：

- 自動増加の主キー
- 既存のstudents、courses、teachersテーブルとの外部キー関係
- 複数の評価項目（筆記試験、実技試験、レポート、出席点など）
- 各評価の重み付け設定
- 成績の範囲制約（0-100点）
- 評価日時の記録
- 評価者（教師）の記録
- コメント機能
- 再評価フラグ
- 論理削除機能
- 作成・更新日時の自動管理
- 適切なインデックス設定

解答

解答32-1

```
CREATE TABLE classrooms_practice (
    room_id INT AUTO_INCREMENT PRIMARY KEY,
    room_name VARCHAR(50) NOT NULL,
    building VARCHAR(30),
    floor TINYINT,
    capacity INT DEFAULT 30,
    has_projector BOOLEAN DEFAULT FALSE
);
```

解答32-2

```
CREATE TABLE books (
    book_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    isbn VARCHAR(20) UNIQUE NOT NULL,
    title VARCHAR(200) NOT NULL,
    author VARCHAR(100) NOT NULL,
    publisher VARCHAR(100),
    publication_year INT,
    price DECIMAL(8,2),
    category ENUM('fiction', 'non-fiction', 'textbook', 'reference'),
    is_available BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CHECK (publication_year >= 1900 AND publication_year <= 2100)
);
```

解答32-3

```
CREATE TABLE book_loans (
    loan_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    book_id BIGINT NOT NULL,
    student_id BIGINT NOT NULL,
    loan_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,
    status ENUM('borrowed', 'returned', 'overdue') DEFAULT 'borrowed',

    FOREIGN KEY (book_id) REFERENCES books(book_id) ON DELETE RESTRICT,
    FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE RESTRICT,

    CHECK (due_date > loan_date),
    CHECK (return_date IS NULL OR return_date >= loan_date)
);
```

解答32-4

```
CREATE TABLE class_schedules (
    schedule_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_id VARCHAR(16) NOT NULL,
    classroom_id INT NOT NULL,
    day_of_week TINYINT NOT NULL,
    period TINYINT NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    semester ENUM('spring', 'summer', 'fall', 'winter') NOT NULL,
    academic_year INT NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,

    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE,
    FOREIGN KEY (classroom_id) REFERENCES classrooms_practice(room_id) ON DELETE
RESTRICT,

    CHECK (day_of_week >= 1 AND day_of_week <= 7),
    CHECK (period >= 1 AND period <= 8),
    CHECK (end_time > start_time),
    CHECK (academic_year >= 2000),

    UNIQUE KEY unique_schedule (classroom_id, day_of_week, period, semester,
academic_year)
);
```

解答32-5

```
CREATE TABLE student_profiles (
    profile_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL UNIQUE,
```

```
    hobbies JSON,
    skills JSON,
    emergency_contacts JSON,
    preferences JSON DEFAULT ('{}'),
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE
);
```

解答32-6

```
CREATE TABLE detailed_grades (
    grade_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    teacher_id BIGINT NOT NULL,

    -- 評価項目
    evaluation_type ENUM('written_exam', 'practical_exam', 'report', 'attendance',
'participation', 'project') NOT NULL,
    evaluation_name VARCHAR(100) NOT NULL,

    -- 成績関連
    score DECIMAL(5,2),
    max_score DECIMAL(5,2) DEFAULT 100,
    weight_percentage DECIMAL(5,2) DEFAULT 100,

    -- 日時関連
    evaluation_date DATE NOT NULL,
    submission_deadline DATETIME,
    submitted_at DATETIME,

    -- 追加情報
    comments TEXT,
    is_revaluation BOOLEAN DEFAULT FALSE,
    original_grade_id BIGINT,

    -- 論理削除
    is_deleted BOOLEAN DEFAULT FALSE,
    deleted_at TIMESTAMP NULL,
    deleted_by BIGINT,

    -- 監査情報
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    created_by BIGINT,
    updated_by BIGINT,

    -- 制約
    CHECK (score >= 0 AND score <= max_score),
    CHECK (max_score > 0),
```

```
    CHECK (weight_percentage >= 0 AND weight_percentage <= 100),
    CHECK (submitted_at IS NULL OR submitted_at <= CURRENT_TIMESTAMP),

    -- 外部キー制約
    FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE RESTRICT,
    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE RESTRICT,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id) ON DELETE RESTRICT,
    FOREIGN KEY (original_grade_id) REFERENCES detailed_grades(grade_id) ON DELETE
SET NULL,

    -- インデックス
    INDEX idx_student_course (student_id, course_id),
    INDEX idx_evaluation_date (evaluation_date),
    INDEX idx_teacher_course (teacher_id, course_id),
    INDEX idx_evaluation_type (evaluation_type),
    INDEX idx_active_grades (is_deleted, evaluation_date)
) COMMENT = '詳細成績管理テーブル';
```

まとめ

この章では、CREATE TABLE文について詳しく学びました：

1. **CREATE TABLE文の基本概念**：
   - DDL（データ定義言語）の一部としての役割
   - テーブル構造（スキーマ）の定義
   - カラム、データ型、制約の基本理解

2. **データ型の選択**：
   - 数値型（INT、BIGINT、DECIMAL等）の適切な使用
   - 文字列型（VARCHAR、TEXT等）の容量考慮
   - 日付・時刻型の使い分け
   - 特殊型（ENUM、JSON等）の活用

3. **制約の活用**：
   - NOT NULL制約による必須項目の定義
   - PRIMARY KEY制約による主キー設定
   - FOREIGN KEY制約による関連性の定義
   - CHECK制約による値の範囲制限
   - UNIQUE制約による一意性の保証

4. **実践的な設計手法**：
   - AUTO_INCREMENTによる自動連番
   - DEFAULT値による初期値設定
   - 複合制約の使用
   - 将来の拡張性を考慮した設計

5. **ベストプラクティス**：

- 適切な命名規則の採用
- 段階的なテーブル設計
- ドキュメント化の重要性
- パフォーマンスを考慮したインデックス設定

6. **エラー回避と対処法**：

- よくあるエラーパターンの理解
- IF NOT EXISTSによる安全な作成
- 制約エラーの回避方法

CREATE TABLE文は、データベース設計の基礎となる重要なSQL文です。適切にテーブルを設計することで、データの整合性を保ち、効率的なデータベースシステムを構築できます。

次の章では、「ALTER TABLE：テーブル構造の変更」について学び、既存テーブルの構造を安全に変更する方法を理解していきます。

---

33. テーブル構造の変更：ALTER TABLE文

はじめに

前章では、CREATE TABLE文を使用してテーブルを作成する方法を学びました。しかし、実際のシステム運用では、要件の変更や機能追加により、既存のテーブル構造を変更する必要が頻繁に発生します。この章では、既存テーブルの構造を安全に変更するための「ALTER TABLE文」について学習します。

ALTER TABLE文は、既存のテーブルに対してカラムの追加・削除・変更、制約の追加・削除、インデックスの管理など、テーブル構造のあらゆる変更を行うことができる非常に強力なSQL文です。

ALTER TABLE文が必要となる場面の例：

- 「学生テーブルに新しく『出身地』カラムを追加したい」
- 「成績テーブルの点数カラムのデータ型を変更したい」
- 「講座テーブルに受講者数制限の制約を追加したい」
- 「不要になったカラムを削除してテーブルを整理したい」
- 「システム改修でテーブル名を変更したい」
- 「パフォーマンス向上のためにインデックスを追加したい」
- 「セキュリティ強化のため、新しい制約を追加したい」

この章では、ALTER TABLE文の基本構文から、実践的な使用方法、注意点まで詳しく学んでいきます。

ALTER TABLE文とは

ALTER TABLE文は、既存のテーブル構造（スキーマ）を変更するためのSQL文です。テーブルを削除・再作成することなく、カラムや制約、インデックスなどを動的に変更できます。

> **用語解説**：
>
> - **ALTER TABLE文**：既存テーブルの構造を変更するDDL文です。
> - **カラムの追加（ADD COLUMN）**：テーブルに新しいカラムを追加することです。

> - **カラムの削除（DROP COLUMN）**：テーブルから不要なカラムを削除することです。
> - **カラムの変更（MODIFY/CHANGE）**：既存カラムのデータ型や制約を変更することです。
> - **制約の追加（ADD CONSTRAINT）**：テーブルに新しい制約を追加することです。
> - **制約の削除（DROP CONSTRAINT）**：既存の制約を削除することです。
> - **インデックスの追加（ADD INDEX）**：検索性能向上のためのインデックスを追加することです。
> - **テーブル名変更（RENAME TO）**：テーブル名を変更することです。
> - **スキーマ変更**：テーブルの構造定義を変更することです。
> - **DDLロック**：ALTER TABLE実行中にテーブルがロックされることです。

ALTER TABLE文の基本構文

基本的な構文パターン

```
-- カラムの追加
ALTER TABLE テーブル名 ADD COLUMN カラム名 データ型 [制約];

-- カラムの削除
ALTER TABLE テーブル名 DROP COLUMN カラム名;

-- カラムの変更
ALTER TABLE テーブル名 MODIFY COLUMN カラム名 新しいデータ型 [制約];
ALTER TABLE テーブル名 CHANGE COLUMN 古いカラム名 新しいカラム名 データ型 [制約];

-- 制約の追加
ALTER TABLE テーブル名 ADD CONSTRAINT 制約名 制約内容;

-- 制約の削除
ALTER TABLE テーブル名 DROP CONSTRAINT 制約名;

-- インデックスの追加
ALTER TABLE テーブル名 ADD INDEX インデックス名 (カラム名);

-- インデックスの削除
ALTER TABLE テーブル名 DROP INDEX インデックス名;

-- テーブル名の変更
ALTER TABLE 古いテーブル名 RENAME TO 新しいテーブル名;
```

カラムの操作

1. カラムの追加（ADD COLUMN）

基本的なカラム追加

```
-- studentsテーブルに出身地カラムを追加
ALTER TABLE students ADD COLUMN hometown VARCHAR(100);
```

```
-- 追加されたカラムの確認
DESCRIBE students;

-- デフォルト値付きでカラムを追加
ALTER TABLE students ADD COLUMN grade_level INT DEFAULT 1;

-- 必須カラムの追加（既存データに影響するため注意）
ALTER TABLE students ADD COLUMN student_status VARCHAR(20) NOT NULL DEFAULT
'active';
```

特定の位置にカラムを追加

```
-- 最初の位置にカラムを追加
ALTER TABLE students ADD COLUMN student_code VARCHAR(20) FIRST;

-- 特定のカラムの後に追加
ALTER TABLE students ADD COLUMN middle_name VARCHAR(50) AFTER student_name;

-- 確認
DESCRIBE students;
```

複数カラムの同時追加

```
-- 複数のカラムを一度に追加
ALTER TABLE students
ADD COLUMN birth_place VARCHAR(100),
ADD COLUMN guardian_name VARCHAR(100),
ADD COLUMN guardian_phone VARCHAR(20);
```

2. カラムの削除（DROP COLUMN）

```
-- 不要なカラムの削除
ALTER TABLE students DROP COLUMN middle_name;

-- 複数カラムの同時削除
ALTER TABLE students
DROP COLUMN student_code,
DROP COLUMN birth_place;

-- 削除の確認
DESCRIBE students;
```

3. カラムの変更

データ型の変更（MODIFY）

```
-- gradesテーブルのscoreカラムのデータ型を変更
-- 変更前の確認
DESCRIBE grades;

-- DECIMAL(5,2)からDECIMAL(6,2)に変更（より大きな値に対応）
ALTER TABLE grades MODIFY COLUMN score DECIMAL(6,2);

-- 文字列カラムの長さを変更
ALTER TABLE courses MODIFY COLUMN course_name VARCHAR(200);

-- 変更後の確認
DESCRIBE grades;
DESCRIBE courses;
```

カラム名とデータ型の同時変更（CHANGE）

```
-- カラム名を変更しながらデータ型も変更
ALTER TABLE students
CHANGE COLUMN hometown home_prefecture VARCHAR(50);

-- 確認
DESCRIBE students;
```

制約の追加・削除

```
-- カラムにNOT NULL制約を追加
-- まず既存のNULL値を適切な値に更新
UPDATE students SET guardian_name = '未登録' WHERE guardian_name IS NULL;

-- NOT NULL制約を追加
ALTER TABLE students MODIFY COLUMN guardian_name VARCHAR(100) NOT NULL;

-- NOT NULL制約を削除（NULL許可に変更）
ALTER TABLE students MODIFY COLUMN guardian_phone VARCHAR(20) NULL;
```

制約の操作

1. 主キー制約の操作

主キーの追加

```
-- 新しいテーブルを作成（主キーなし）
CREATE TABLE temp_table (
    id INT,
```

```
    name VARCHAR(100)
);

-- 主キーを追加
ALTER TABLE temp_table ADD PRIMARY KEY (id);

-- 確認
DESCRIBE temp_table;
```

主キーの削除と変更

```
-- 主キーの削除
ALTER TABLE temp_table DROP PRIMARY KEY;

-- 複合主キーの追加
ALTER TABLE temp_table ADD PRIMARY KEY (id, name);

-- 再度主キーを削除
ALTER TABLE temp_table DROP PRIMARY KEY;

-- AUTO_INCREMENT付き主キーの追加
ALTER TABLE temp_table
MODIFY COLUMN id INT AUTO_INCREMENT,
ADD PRIMARY KEY (id);
```

2. 外部キー制約の操作

外部キー制約の追加

```
-- 既存のstudent_coursesテーブルに外部キー制約を追加
-- まず制約名を指定して追加
ALTER TABLE student_courses
ADD CONSTRAINT fk_student_courses_student
FOREIGN KEY (student_id) REFERENCES students(student_id)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE student_courses
ADD CONSTRAINT fk_student_courses_course
FOREIGN KEY (course_id) REFERENCES courses(course_id)
ON DELETE CASCADE ON UPDATE CASCADE;

-- 制約の確認
SHOW CREATE TABLE student_courses;
```

外部キー制約の削除

```
-- 外部キー制約の削除
ALTER TABLE student_courses DROP FOREIGN KEY fk_student_courses_student;
ALTER TABLE student_courses DROP FOREIGN KEY fk_student_courses_course;
```

3. UNIQUE制約の操作

```
-- 学生テーブルにメールアドレスカラムを追加
ALTER TABLE students ADD COLUMN email VARCHAR(255);

-- UNIQUE制約を追加
ALTER TABLE students ADD CONSTRAINT uk_students_email UNIQUE (email);

-- または短縮形で
ALTER TABLE students ADD UNIQUE (guardian_phone);

-- UNIQUE制約の削除
ALTER TABLE students DROP INDEX uk_students_email;
```

4. CHECK制約の操作（MySQL 8.0以降）

```
-- CHECK制約の追加
ALTER TABLE grades
ADD CONSTRAINT chk_score_range
CHECK (score >= 0 AND score <= 100);

-- CHECK制約の削除
ALTER TABLE grades DROP CHECK chk_score_range;
```

インデックスの操作

1. インデックスの追加

```
-- 単一カラムインデックスの追加
ALTER TABLE students ADD INDEX idx_student_name (student_name);

-- 複合インデックスの追加
ALTER TABLE grades ADD INDEX idx_student_course (student_id, course_id);

-- 一意インデックスの追加
ALTER TABLE teachers ADD UNIQUE INDEX idx_teacher_email (teacher_name);

-- インデックスの確認
SHOW INDEX FROM students;
SHOW INDEX FROM grades;
```

2. インデックスの削除

```
-- インデックスの削除
ALTER TABLE students DROP INDEX idx_student_name;
ALTER TABLE grades DROP INDEX idx_student_course;
```

テーブル名の変更

```
-- テーブル名の変更
ALTER TABLE temp_table RENAME TO sample_table;

-- 確認
SHOW TABLES LIKE 'sample%';

-- 元に戻す
ALTER TABLE sample_table RENAME TO temp_table;
```

実践的なALTER TABLE例

例1：学生テーブルの段階的拡張

```
-- 元のstudentsテーブル構造を確認
DESCRIBE students;

-- Phase 1: 基本的な個人情報の追加
ALTER TABLE students
ADD COLUMN email VARCHAR(255),
ADD COLUMN phone VARCHAR(20),
ADD COLUMN birth_date DATE,
ADD COLUMN gender ENUM('male', 'female', 'other', 'prefer_not_to_say');

-- Phase 2: 学籍情報の追加
ALTER TABLE students
ADD COLUMN student_number VARCHAR(20) UNIQUE,
ADD COLUMN admission_date DATE,
ADD COLUMN graduation_date DATE,
ADD COLUMN status ENUM('enrolled', 'graduated', 'withdrawn', 'suspended') DEFAULT
'enrolled';

-- Phase 3: 制約の追加
ALTER TABLE students
ADD CONSTRAINT uk_student_email UNIQUE (email),
ADD CONSTRAINT chk_birth_date CHECK (birth_date <= CURRENT_DATE),
ADD CONSTRAINT chk_admission_date CHECK (admission_date >= '2000-01-01');

-- Phase 4: インデックスの追加
ALTER TABLE students
```

```
ADD INDEX idx_student_number (student_number),
ADD INDEX idx_admission_date (admission_date),
ADD INDEX idx_status (status);

-- 最終的な構造確認
DESCRIBE students;
```

例2：成績テーブルの改良

```
-- 現在のgradesテーブル構造を確認
DESCRIBE grades;

-- 成績システムの改良
-- 1. 新しい評価項目の追加
ALTER TABLE grades
ADD COLUMN evaluation_method ENUM('exam', 'assignment', 'project',
'participation') DEFAULT 'exam',
ADD COLUMN weight DECIMAL(5,2) DEFAULT 100.00,
ADD COLUMN comments TEXT,
ADD COLUMN evaluated_by BIGINT,
ADD COLUMN evaluated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP;

-- 2. 外部キー制約の追加
ALTER TABLE grades
ADD CONSTRAINT fk_grades_evaluated_by
FOREIGN KEY (evaluated_by) REFERENCES teachers(teacher_id);

-- 3. CHECK制約の追加
ALTER TABLE grades
ADD CONSTRAINT chk_weight_range CHECK (weight >= 0 AND weight <= 100),
ADD CONSTRAINT chk_max_score_positive CHECK (max_score > 0);

-- 4. インデックスの追加
ALTER TABLE grades
ADD INDEX idx_evaluation_method (evaluation_method),
ADD INDEX idx_evaluated_at (evaluated_at),
ADD INDEX idx_student_course_method (student_id, course_id, evaluation_method);

-- 結果確認
DESCRIBE grades;
SHOW INDEX FROM grades;
```

例3：新機能追加のためのテーブル拡張

```
-- 出席管理システムの拡張
DESCRIBE attendance;

-- 出席管理の詳細化
ALTER TABLE attendance
```

```
ADD COLUMN check_in_time TIME,
ADD COLUMN check_out_time TIME,
ADD COLUMN late_minutes INT DEFAULT 0,
ADD COLUMN excuse_reason TEXT,
ADD COLUMN approved_by BIGINT,
ADD COLUMN approved_at TIMESTAMP NULL;

-- 制約の追加
ALTER TABLE attendance
ADD CONSTRAINT chk_late_minutes CHECK (late_minutes >= 0),
ADD CONSTRAINT chk_check_times CHECK (check_out_time IS NULL OR check_out_time >=
check_in_time);

-- 外部キー制約の追加
ALTER TABLE attendance
ADD CONSTRAINT fk_attendance_approved_by
FOREIGN KEY (approved_by) REFERENCES teachers(teacher_id);

-- インデックスの追加
ALTER TABLE attendance
ADD INDEX idx_check_in_time (check_in_time),
ADD INDEX idx_status_date (status, schedule_id);
```

データの互換性を考慮した変更

1. データ型変更時の注意点

```
-- 安全なデータ型変更の例

-- Step 1: 現在のデータ範囲を確認
SELECT
    MIN(score) as min_score,
    MAX(score) as max_score,
    COUNT(*) as total_records
FROM grades;

-- Step 2: 新しいデータ型が既存データを収容できることを確認
-- DECIMAL(5,2) → DECIMAL(6,2) は安全（より大きな範囲）
ALTER TABLE grades MODIFY COLUMN score DECIMAL(6,2);

-- Step 3: 変更後のデータ確認
SELECT
    MIN(score) as min_score,
    MAX(score) as max_score,
    COUNT(*) as total_records
FROM grades;
```

2. NOT NULL制約追加時の対処

```
-- NOT NULL制約を安全に追加する手順

-- Step 1: NULL値の存在確認
SELECT COUNT(*) as null_count
FROM students
WHERE email IS NULL;

-- Step 2: NULL値がある場合は適切な値で更新
UPDATE students
SET email = CONCAT('student_', student_id, '@school.example.com')
WHERE email IS NULL;

-- Step 3: NOT NULL制約の追加
ALTER TABLE students MODIFY COLUMN email VARCHAR(255) NOT NULL;

-- Step 4: 確認
SELECT COUNT(*) as null_count
FROM students
WHERE email IS NULL;
```

3. カラム削除前のデータ確認

```
-- カラム削除前の安全確認

-- Step 1: 削除予定カラムの使用状況確認
SELECT
    COUNT(*) as total_records,
    COUNT(guardian_name) as non_null_records,
    COUNT(*) - COUNT(guardian_name) as null_records
FROM students;

-- Step 2: 重要なデータがある場合はバックアップ
CREATE TABLE students_guardian_backup AS
SELECT student_id, guardian_name, guardian_phone
FROM students
WHERE guardian_name IS NOT NULL;

-- Step 3: カラムの削除
ALTER TABLE students
DROP COLUMN guardian_name,
DROP COLUMN guardian_phone;
```

パフォーマンスへの影響と対策

1. 大きなテーブルでの変更

```
-- 大量データテーブルでの安全な変更手順
```

```
-- 現在のテーブルサイズ確認
SELECT
    COUNT(*) as record_count,
    ROUND(AVG(LENGTH(CONCAT_WS('', student_id, student_name)))) as avg_row_size
FROM students;

-- 変更中のロック時間を最小化するため、段階的に実行
-- Phase 1: インデックス追加（比較的高速）
ALTER TABLE students ADD INDEX idx_student_name (student_name);

-- Phase 2: カラム追加（高速）
ALTER TABLE students ADD COLUMN created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP;

-- Phase 3: データ型変更（時間がかかる可能性）
-- 本番環境では事前にテストが必要
ALTER TABLE students MODIFY COLUMN student_name VARCHAR(150);
```

2. ALTER TABLEの進捗監視

```
-- MySQL 5.7以降での進捗確認
SELECT
    THREAD_ID,
    PROCESSLIST_ID,
    PROCESSLIST_TIME,
    PROCESSLIST_STATE,
    PROCESSLIST_INFO
FROM performance_schema.threads
WHERE PROCESSLIST_INFO LIKE 'ALTER TABLE%';

-- 実行中のクエリ確認
SHOW PROCESSLIST;
```

ALTER TABLEのエラーと対処法

1. データ型変換エラー

```
-- エラー例：互換性のないデータ型変更
-- ALTER TABLE grades MODIFY COLUMN score VARCHAR(10);
-- エラー: 数値データが文字列に変換できない場合

-- 対処法1: 事前にデータ確認
SELECT score, COUNT(*)
FROM grades
GROUP BY score
ORDER BY score;

-- 対処法2: 段階的変更
-- まず新しいカラムを追加
ALTER TABLE grades ADD COLUMN score_text VARCHAR(10);
```

```
-- データを変換して新カラムに設定
UPDATE grades SET score_text = CAST(score AS CHAR);

-- 元のカラムを削除し、新カラムの名前を変更
ALTER TABLE grades DROP COLUMN score;
ALTER TABLE grades CHANGE COLUMN score_text score VARCHAR(10);
```

2. 外部キー制約エラー

```
-- エラー例：参照整合性違反
-- ALTER TABLE courses ADD CONSTRAINT fk_teacher
-- FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
-- エラー: 存在しないteacher_idがある場合

-- 対処法: 事前にデータ整合性を確認・修正
SELECT c.course_id, c.teacher_id
FROM courses c
LEFT JOIN teachers t ON c.teacher_id = t.teacher_id
WHERE t.teacher_id IS NULL;

-- 不整合データの修正
UPDATE courses
SET teacher_id = 101  -- 存在する教師ID
WHERE teacher_id NOT IN (SELECT teacher_id FROM teachers);

-- 制約追加
ALTER TABLE courses
ADD CONSTRAINT fk_courses_teacher
FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);
```

3. 容量不足エラー

```
-- 対処法: 一時的な容量確保
-- 不要なデータの削除
DELETE FROM log_table WHERE created_at < DATE_SUB(NOW(), INTERVAL 1 YEAR);

-- テーブルの最適化
OPTIMIZE TABLE large_table;

-- 変更実行
ALTER TABLE large_table ADD COLUMN new_column TEXT;
```

ALTER TABLEのベストプラクティス

1. 事前計画と準備

```
-- 変更前チェックリスト

-- 1. 現在のテーブル構造を記録
SHOW CREATE TABLE students;

-- 2. データ量とインデックスサイズを確認
SELECT
    table_name,
    table_rows,
    data_length,
    index_length,
    (data_length + index_length) as total_size
FROM information_schema.tables
WHERE table_schema = DATABASE()
AND table_name = 'students';

-- 3. 依存関係の確認
SELECT
    constraint_name,
    table_name,
    referenced_table_name,
    referenced_column_name
FROM information_schema.referential_constraints
WHERE constraint_schema = DATABASE();

-- 4. 実行時間の見積もり（小規模テストテーブルで）
CREATE TABLE students_test LIKE students;
INSERT INTO students_test SELECT * FROM students LIMIT 1000;
-- テスト実行
ALTER TABLE students_test ADD COLUMN test_column VARCHAR(100);
DROP TABLE students_test;
```

2. 安全な実行手順

```
-- 本番環境での安全な変更手順

-- Step 1: バックアップの作成
CREATE TABLE students_backup AS SELECT * FROM students;

-- Step 2: トランザクション開始（可能な場合）
START TRANSACTION;

-- Step 3: 変更実行
ALTER TABLE students ADD COLUMN emergency_contact VARCHAR(100);

-- Step 4: 確認
DESCRIBE students;
SELECT COUNT(*) FROM students;

-- Step 5: 問題なければコミット
```

```
COMMIT;

-- Step 6: バックアップテーブルの削除（十分な確認後）
-- DROP TABLE students_backup;
```

3. 段階的なリリース

```
-- 大規模な変更の段階的実装

-- Release 1: カラム追加（NULL許可）
ALTER TABLE students ADD COLUMN new_feature_flag BOOLEAN DEFAULT NULL;

-- Release 2: デフォルト値設定
ALTER TABLE students MODIFY COLUMN new_feature_flag BOOLEAN DEFAULT FALSE;

-- Release 3: 既存データの更新
UPDATE students SET new_feature_flag = FALSE WHERE new_feature_flag IS NULL;

-- Release 4: NOT NULL制約追加
ALTER TABLE students MODIFY COLUMN new_feature_flag BOOLEAN NOT NULL DEFAULT
FALSE;

-- Release 5: インデックス追加（必要に応じて）
ALTER TABLE students ADD INDEX idx_new_feature_flag (new_feature_flag);
```

練習問題

問題33-1：基本的なカラム操作

studentsテーブルに以下の変更を行ってください：

1. `nationality`カラム（VARCHAR(50)、デフォルト値'日本'）を追加
2. `birth_date`カラム（DATE型）を追加
3. `student_name`カラムのデータ型をVARCHAR(150)に変更
4. 変更後のテーブル構造を確認

問題33-2：制約の追加

teachersテーブルに以下の変更を行ってください：

1. `email`カラム（VARCHAR(255)）を追加
2. `hire_date`カラム（DATE型）を追加
3. `email`カラムにUNIQUE制約を追加
4. `hire_date`に対して2000年1月1日以降という CHECK制約を追加（MySQL 8.0以降）
5. `teacher_name`にNOT NULL制約が設定されていることを確認

問題33-3：インデックスの管理

gradesテーブルに以下のインデックスを追加してください：

1. student_idの単一インデックス（名前：idx_grades_student）
2. course_idとgrade_typeの複合インデックス（名前：idx_grades_course_type）
3. submission_dateの単一インデックス（名前：idx_grades_submission_date）
4. 追加したインデックスを確認し、その後すべて削除

問題33-4：外部キー制約の追加

以下の手順で外部キー制約を追加してください：

1. attendanceテーブルとstudentsテーブルの関係を確認
2. attendance.student_idに対してstudentsテーブルへの外部キー制約を追加
3. CASCADE削除オプションを設定
4. 制約が正しく追加されたことを確認

問題33-5：テーブルの大幅な拡張

coursesテーブルを以下の要件で拡張してください：

1. descriptionカラム（TEXT型）を追加
2. creditsカラム（INT型、デフォルト値2）を追加
3. max_studentsカラム（INT型、デフォルト値30）を追加
4. statusカラム（ENUM型：'active', 'inactive', 'planning'、デフォルト値'planning'）を追加
5. created_atカラム（TIMESTAMP型、現在日時がデフォルト）を追加
6. updated_atカラム（TIMESTAMP型、現在日時がデフォルト、更新時に自動更新）を追加
7. creditsに対して1以上8以下のCHECK制約を追加
8. max_studentsに対して1以上200以下のCHECK制約を追加
9. 適切なインデックスを追加

問題33-6：複雑なデータ移行を伴う変更

以下の複雑な変更を安全に実行してください：

1. studentsテーブルのstudent_nameをfirst_nameとlast_nameに分割
2. 既存のstudent_nameデータを適切に分割して新しいカラムに移行
3. 元のstudent_nameカラムを削除
4. 新しいカラムに適切な制約とインデックスを追加
5. 変更前後でデータの整合性を確認

解答

解答33-1

```
-- 1. nationalityカラムの追加
ALTER TABLE students ADD COLUMN nationality VARCHAR(50) DEFAULT '日本';

-- 2. birth_dateカラムの追加
ALTER TABLE students ADD COLUMN birth_date DATE;

-- 3. student_nameカラムのデータ型変更
ALTER TABLE students MODIFY COLUMN student_name VARCHAR(150);
```

```
-- 4. テーブル構造の確認
DESCRIBE students;
```

解答33-2

```
-- 1. emailカラムの追加
ALTER TABLE teachers ADD COLUMN email VARCHAR(255);

-- 2. hire_dateカラムの追加
ALTER TABLE teachers ADD COLUMN hire_date DATE;

-- 3. emailカラムにUNIQUE制約を追加
ALTER TABLE teachers ADD CONSTRAINT uk_teachers_email UNIQUE (email);

-- 4. hire_dateにCHECK制約を追加（MySQL 8.0以降）
ALTER TABLE teachers ADD CONSTRAINT chk_hire_date
CHECK (hire_date >= '2000-01-01');

-- 5. teacher_nameのNOT NULL制約確認
DESCRIBE teachers;
-- または
SHOW CREATE TABLE teachers;
```

解答33-3

```
-- 1. student_idの単一インデックス追加
ALTER TABLE grades ADD INDEX idx_grades_student (student_id);

-- 2. course_idとgrade_typeの複合インデックス追加
ALTER TABLE grades ADD INDEX idx_grades_course_type (course_id, grade_type);

-- 3. submission_dateの単一インデックス追加
ALTER TABLE grades ADD INDEX idx_grades_submission_date (submission_date);

-- 4. インデックスの確認
SHOW INDEX FROM grades;

-- インデックスの削除
ALTER TABLE grades DROP INDEX idx_grades_student;
ALTER TABLE grades DROP INDEX idx_grades_course_type;
ALTER TABLE grades DROP INDEX idx_grades_submission_date;

-- 削除確認
SHOW INDEX FROM grades;
```

解答33-4

```
-- 1. テーブル関係の確認
DESCRIBE attendance;
DESCRIBE students;

-- 2. 外部キー制約の追加
ALTER TABLE attendance
ADD CONSTRAINT fk_attendance_student
FOREIGN KEY (student_id) REFERENCES students(student_id)
ON DELETE CASCADE ON UPDATE CASCADE;

-- 3. 制約の確認
SHOW CREATE TABLE attendance;

-- または制約一覧で確認
SELECT
    constraint_name,
    table_name,
    referenced_table_name,
    delete_rule,
    update_rule
FROM information_schema.referential_constraints
WHERE constraint_schema = DATABASE()
AND table_name = 'attendance';
```

解答33-5

```
-- coursesテーブルの拡張
ALTER TABLE courses
ADD COLUMN description TEXT,
ADD COLUMN credits INT DEFAULT 2,
ADD COLUMN max_students INT DEFAULT 30,
ADD COLUMN status ENUM('active', 'inactive', 'planning') DEFAULT 'planning',
ADD COLUMN created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
ADD COLUMN updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP;

-- CHECK制約の追加
ALTER TABLE courses
ADD CONSTRAINT chk_credits_range CHECK (credits >= 1 AND credits <= 8),
ADD CONSTRAINT chk_max_students_range CHECK (max_students >= 1 AND max_students <=
200);

-- インデックスの追加
ALTER TABLE courses
ADD INDEX idx_courses_status (status),
ADD INDEX idx_courses_credits (credits),
ADD INDEX idx_courses_created_at (created_at);

-- 結果確認
```

```
DESCRIBE courses;
SHOW INDEX FROM courses;
```

解答33-6

```
-- 複雑なデータ移行を伴う変更

-- Step 1: バックアップ作成
CREATE TABLE students_name_backup AS
SELECT student_id, student_name FROM students;

-- Step 2: 新しいカラムを追加
ALTER TABLE students
ADD COLUMN first_name VARCHAR(50),
ADD COLUMN last_name VARCHAR(50);

-- Step 3: 既存データの分割（簡単な例：スペースで分割）
UPDATE students
SET
    last_name = SUBSTRING_INDEX(student_name, ' ', 1),
    first_name = CASE
        WHEN LOCATE(' ', student_name) > 0
        THEN SUBSTRING(student_name, LOCATE(' ', student_name) + 1)
        ELSE student_name
    END
WHERE student_name IS NOT NULL;

-- Step 4: データ整合性確認
SELECT
    student_id,
    student_name,
    last_name,
    first_name
FROM students
LIMIT 10;

-- Step 5: 新しいカラムに制約を追加
ALTER TABLE students
MODIFY COLUMN first_name VARCHAR(50) NOT NULL,
MODIFY COLUMN last_name VARCHAR(50) NOT NULL;

-- Step 6: インデックスを追加
ALTER TABLE students
ADD INDEX idx_students_last_name (last_name),
ADD INDEX idx_students_full_name (last_name, first_name);

-- Step 7: 元のカラムを削除
ALTER TABLE students DROP COLUMN student_name;

-- Step 8: 最終確認
DESCRIBE students;
```

```
SELECT
    student_id,
    last_name,
    first_name,
    CONCAT(last_name, ' ', first_name) as full_name
FROM students
LIMIT 10;

-- Step 9: 問題がなければバックアップテーブル削除
-- DROP TABLE students_name_backup;
```

まとめ

この章では、ALTER TABLE文について詳しく学びました：

1. **ALTER TABLE文の基本概念**：
    - 既存テーブル構造の動的変更
    - DDLロックとパフォーマンスへの影響
    - スキーマ変更の重要性
2. **カラムの操作**：
    - ADD COLUMNによる新しいカラムの追加
    - DROP COLUMNによるカラムの削除
    - MODIFYとCHANGEによるカラムの変更
    - 位置指定（FIRST、AFTER）の活用
3. **制約の管理**：
    - 主キー制約の追加・削除
    - 外部キー制約の設定とカスケードオプション
    - UNIQUE制約とCHECK制約の活用
    - NOT NULL制約の安全な追加
4. **インデックスの管理**：
    - パフォーマンス向上のためのインデックス追加
    - 複合インデックスの設計
    - 不要インデックスの削除
5. **実践的な変更手法**：
    - 段階的なテーブル拡張
    - データ移行を伴う変更
    - 大規模テーブルでの安全な変更
6. **エラー対処と予防**：
    - データ型変換エラーの回避
    - 外部キー制約エラーの解決

- 事前のデータ整合性確認

7. **ベストプラクティス**：
    - 変更前の十分な計画と準備
    - バックアップの作成
    - 段階的なリリース戦略
    - パフォーマンス影響の最小化

ALTER TABLE文は非常に強力ですが、既存データへの影響やシステムの可用性を慎重に考慮する必要があります。特に本番環境では、十分なテストと計画的な実行が重要です。

次の章では、「DROP/TRUNCATE：テーブルの削除とクリア」について学び、テーブルの削除と初期化の方法を理解していきます。

---

34. テーブルの削除とクリア：DROP TABLE文とTRUNCATE文

はじめに

前章では、ALTER TABLE文を使用してテーブル構造を変更する方法を学びました。この章では、テーブルそのものを削除したり、テーブル内のデータをすべて削除したりするための「DROP TABLE文」と「TRUNCATE文」について学習します。

これらの操作は、データベース管理において非常に強力で便利ですが、**一度実行すると元に戻すことができない**危険な操作でもあります。誤って実行するとデータの完全な損失につながるため、特に慎重な扱いが必要です。

DROP TABLE文とTRUNCATE文が必要となる場面の例：

- 「テスト用に作成したテーブルが不要になったので削除したい」
- 「システム移行が完了したので、古いテーブルを削除したい」
- 「一時的な集計テーブルを毎回初期化して使いたい」
- 「大量のログデータを一括削除して容量を確保したい」
- 「開発環境のテーブルをクリアして新しいテストデータを投入したい」
- 「不要になった中間テーブルを削除してデータベースを整理したい」
- 「AUTO_INCREMENTの値をリセットしたい」

この章では、これらの操作の基本構文から、安全な実行方法、注意点まで詳しく学んでいきます。

DROP TABLE文とTRUNCATE文とは

DROP TABLE文

DROP TABLE文は、テーブル全体（構造とデータの両方）をデータベースから完全に削除するSQL文です。実行後、そのテーブルは存在しなくなります。

TRUNCATE文

TRUNCATE文は、テーブルの構造は残したまま、テーブル内のすべてのデータを高速に削除するSQL文です。テーブル自体は残るため、再度データを挿入することができます。

> **用語解説：**
>
> - **DROP TABLE文**：テーブルの構造とデータを完全に削除するDDL文です。
> - **TRUNCATE文**：テーブル構造を残してデータのみを高速削除するDDL文です。
> - **カスケード削除**：外部キー制約で関連付けられたテーブルも連鎖的に削除することです。
> - **フルテーブルスキャン**：テーブル全体を順次読み取る処理です。
> - **ページ削除**：データベースの内部構造単位でデータを削除する高速な方法です。
> - **AUTO_INCREMENT リセット**：自動増加カウンタを初期値に戻すことです。
> - **外部キー制約**：他のテーブルとの関連性を保証する制約で、削除操作に影響します。
> - **参照整合性**：関連するテーブル間でデータの整合性が保たれている状態です。
> - **ロールバック**：トランザクションを取り消してデータを元の状態に戻すことです。
> - **DDLロック**：DDL操作中にテーブルがロックされることです。

基本構文

DROP TABLE文の構文

```
-- 基本構文
DROP TABLE テーブル名;

-- 複数テーブルの同時削除
DROP TABLE テーブル名1, テーブル名2, テーブル名3;

-- 存在しない場合でもエラーにしない
DROP TABLE IF EXISTS テーブル名;

-- 外部キー制約を無視して削除（注意が必要）
SET FOREIGN_KEY_CHECKS = 0;
DROP TABLE テーブル名;
SET FOREIGN_KEY_CHECKS = 1;
```

TRUNCATE文の構文

```
-- 基本構文
TRUNCATE TABLE テーブル名;

-- またはTABLEキーワードを省略
TRUNCATE テーブル名;
```

DROP TABLE文の詳細

1. 基本的なテーブル削除

```
-- テスト用テーブルの作成
CREATE TABLE test_table (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- データの挿入
INSERT INTO test_table (name) VALUES ('テストデータ1'), ('テストデータ2');

-- テーブルの確認
SELECT * FROM test_table;

-- テーブルの削除
DROP TABLE test_table;

-- 削除確認（エラーになる）
-- SELECT * FROM test_table;  -- エラー: Table 'test_table' doesn't exist
```

2. 複数テーブルの同時削除

```
-- 複数のテスト用テーブルを作成
CREATE TABLE temp_table1 (id INT, data VARCHAR(50));
CREATE TABLE temp_table2 (id INT, data VARCHAR(50));
CREATE TABLE temp_table3 (id INT, data VARCHAR(50));

-- テーブル一覧で確認
SHOW TABLES LIKE 'temp_%';

-- 複数テーブルを同時削除
DROP TABLE temp_table1, temp_table2, temp_table3;

-- 削除確認
SHOW TABLES LIKE 'temp_%';
```

3. IF EXISTS句の使用

```
-- 存在しないテーブルを削除しようとした場合（エラー）
-- DROP TABLE non_existent_table;  -- エラー: Unknown table 'non_existent_table'

-- IF EXISTS句を使用した安全な削除
DROP TABLE IF EXISTS non_existent_table;  -- エラーにならない

-- 複数のテーブルに対するIF EXISTS
DROP TABLE IF EXISTS old_table1, old_table2, old_table3;
```

TRUNCATE文の詳細

1. 基本的なデータクリア

```
-- テスト用データの準備
CREATE TABLE truncate_test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    value INT
);

INSERT INTO truncate_test (name, value) VALUES
('データ1', 100),
('データ2', 200),
('データ3', 300);

-- データの確認
SELECT * FROM truncate_test;

-- テーブル構造の確認
DESCRIBE truncate_test;

-- TRUNCATEでデータをクリア
TRUNCATE TABLE truncate_test;

-- データがクリアされたことを確認
SELECT * FROM truncate_test;

-- テーブル構造は残っていることを確認
DESCRIBE truncate_test;
```

2. AUTO_INCREMENTのリセット

```
-- データを再挿入
INSERT INTO truncate_test (name, value) VALUES ('新データ1', 1000);

-- IDが1から開始されることを確認
SELECT * FROM truncate_test;

-- AUTO_INCREMENT値の確認
SHOW TABLE STATUS LIKE 'truncate_test';
```

DELETE文、TRUNCATE文、DROP TABLE文の比較

比較表

項目	DELETE	TRUNCATE	DROP TABLE
対象	データのみ	データのみ	構造とデータ
実行速度	遅い（行単位）	高速（ページ単位）	高速

項目	DELETE	TRUNCATE	DROP TABLE
WHERE句	使用可能	使用不可	使用不可
ロールバック	可能	不可（DDL）	不可（DDL）
AUTO_INCREMENT	リセットされない	リセットされる	N/A
トリガー	実行される	実行されない	実行されない
外部キー制約	チェックされる	制約があると実行不可	CASCADE必要
ログ記録	詳細ログ	最小限ログ	最小限ログ

実例による比較

```
-- 比較用テーブルの作成
CREATE TABLE comparison_test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO comparison_test (name) VALUES
('データ1'), ('データ2'), ('データ3'), ('データ4'), ('データ5');

-- 現在の状態確認
SELECT * FROM comparison_test;
SHOW TABLE STATUS LIKE 'comparison_test';

-- 1. DELETE文での削除（条件付き）
DELETE FROM comparison_test WHERE id <= 3;
SELECT * FROM comparison_test;  -- id=4,5が残る

-- 新しいデータ挿入
INSERT INTO comparison_test (name) VALUES ('新データ1');
SELECT * FROM comparison_test;  -- AUTO_INCREMENTは6から開始

-- すべてのデータを削除
DELETE FROM comparison_test;
SELECT * FROM comparison_test;  -- データなし

-- 新しいデータ挿入
INSERT INTO comparison_test (name) VALUES ('削除後データ');
SELECT * FROM comparison_test;  -- AUTO_INCREMENTは7から開始

-- TRUNCATEでのクリア
TRUNCATE TABLE comparison_test;
INSERT INTO comparison_test (name) VALUES ('TRUNCATE後データ');
SELECT * FROM comparison_test;  -- AUTO_INCREMENTは1から開始
```

外部キー制約がある場合の処理

1. 外部キー制約の確認

```
-- 現在の外部キー制約を確認
SELECT
    constraint_name,
    table_name,
    referenced_table_name,
    delete_rule
FROM information_schema.referential_constraints
WHERE constraint_schema = DATABASE();
```

2. 外部キー制約による削除エラー

```
-- 参照されているテーブルの削除を試行（エラーになる例）
-- DROP TABLE students;  -- エラー: Cannot delete or update a parent row

-- 子テーブル（参照する側）から先に削除
-- DROP TABLE grades;     -- 成功
-- DROP TABLE students;   -- その後で親テーブルを削除可能
```

3. CASCADE削除の活用

```
-- CASCADE削除設定のテーブル作成例
CREATE TABLE parent_table (
    id INT PRIMARY KEY,
    name VARCHAR(100)
);

CREATE TABLE child_table (
    id INT PRIMARY KEY,
    parent_id INT,
    data VARCHAR(100),
    FOREIGN KEY (parent_id) REFERENCES parent_table(id) ON DELETE CASCADE
);

-- データ挿入
INSERT INTO parent_table VALUES (1, '親データ1'), (2, '親データ2');
INSERT INTO child_table VALUES (1, 1, '子データ1'), (2, 1, '子データ2'), (3, 2, '子データ3');

-- 親テーブルの特定行を削除（CASCADE削除）
DELETE FROM parent_table WHERE id = 1;

-- 関連する子データも自動削除されることを確認
SELECT * FROM child_table;
```

4. 外部キー制約を一時的に無効化

```
-- 注意：本番環境では慎重に使用すること

-- 外部キー制約の無効化
SET FOREIGN_KEY_CHECKS = 0;

-- テーブル削除（通常はエラーになるものも削除可能）
DROP TABLE IF EXISTS parent_table;
DROP TABLE IF EXISTS child_table;

-- 外部キー制約の再有効化
SET FOREIGN_KEY_CHECKS = 1;
```

実践的な削除例

例1：ログテーブルの定期クリア

```
-- ログテーブルの作成
CREATE TABLE access_logs (
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT,
    action VARCHAR(100),
    ip_address VARCHAR(45),
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_created_at (created_at),
    INDEX idx_user_id (user_id)
);

-- サンプルデータの挿入
INSERT INTO access_logs (user_id, action, ip_address) VALUES
(101, 'login', '192.168.1.1'),
(102, 'view_page', '192.168.1.2'),
(103, 'logout', '192.168.1.3');

-- 定期的なログクリア（月次）
-- 古いログデータの確認
SELECT COUNT(*) as old_logs
FROM access_logs
WHERE created_at < DATE_SUB(NOW(), INTERVAL 1 MONTH);

-- 古いデータの削除（DELETE使用 - 条件付き削除）
DELETE FROM access_logs
WHERE created_at < DATE_SUB(NOW(), INTERVAL 1 MONTH);

-- または、全ログのクリア（TRUNCATE使用 - 高速）
-- TRUNCATE TABLE access_logs;
```

例2：テスト環境のデータリセット

```
-- テスト環境での安全なデータリセット手順

-- Step 1: 現在のデータ量を確認
SELECT
    'students' as table_name, COUNT(*) as record_count FROM students
UNION ALL
SELECT 'grades', COUNT(*) FROM grades
UNION ALL
SELECT 'attendance', COUNT(*) FROM attendance
UNION ALL
SELECT 'student_courses', COUNT(*) FROM student_courses;

-- Step 2: 外部キー制約順に考慮してTRUNCATE
-- 子テーブルから先にクリア
TRUNCATE TABLE attendance;
TRUNCATE TABLE grades;
TRUNCATE TABLE student_courses;

-- Step 3: 親テーブルのクリア
TRUNCATE TABLE course_schedule;
-- students と courses は他のテーブルから参照されているので最後

-- Step 4: 結果確認
SELECT
    'students' as table_name, COUNT(*) as record_count FROM students
UNION ALL
SELECT 'grades', COUNT(*) FROM grades
UNION ALL
SELECT 'attendance', COUNT(*) FROM attendance
UNION ALL
SELECT 'student_courses', COUNT(*) FROM student_courses;
```

例3：一時テーブルの管理

```
-- 処理用一時テーブルの作成
CREATE TEMPORARY TABLE temp_calculation (
    id INT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT,
    total_score DECIMAL(8,2),
    average_score DECIMAL(5,2)
);

-- 計算処理
INSERT INTO temp_calculation (student_id, total_score, average_score)
SELECT
    student_id,
    SUM(score) as total_score,
    AVG(score) as average_score
```

```
FROM grades
GROUP BY student_id;

-- 結果確認
SELECT * FROM temp_calculation LIMIT 5;

-- 処理完了後のクリーンアップ
DROP TEMPORARY TABLE temp_calculation;
```

安全な削除手順

1. 削除前のチェックリスト

```
-- チェック1: テーブルの依存関係確認
SELECT
    table_name,
    referenced_table_name,
    constraint_name
FROM information_schema.referential_constraints
WHERE constraint_schema = DATABASE()
AND (table_name = 'target_table' OR referenced_table_name = 'target_table');

-- チェック2: データ量の確認
SELECT
    COUNT(*) as record_count,
    MAX(created_at) as latest_record,
    MIN(created_at) as oldest_record
FROM target_table;

-- チェック3: 使用中のトランザクション確認
SHOW PROCESSLIST;
```

2. バックアップ作成

```
-- 重要なテーブルの削除前バックアップ
CREATE TABLE students_backup_20231201 AS SELECT * FROM students;

-- バックアップの確認
SELECT COUNT(*) FROM students_backup_20231201;

-- テーブル削除の実行
DROP TABLE students;

-- 必要に応じて復元
-- CREATE TABLE students AS SELECT * FROM students_backup_20231201;
```

3. 段階的な削除

```
-- 大量データの段階的削除

-- Step 1: 削除対象の確認
SELECT COUNT(*) as target_count
FROM large_table
WHERE created_at < '2023-01-01';

-- Step 2: 小さなバッチで削除
DELETE FROM large_table
WHERE created_at < '2023-01-01'
LIMIT 1000;

-- Step 3: 進捗確認
SELECT COUNT(*) as remaining_count
FROM large_table
WHERE created_at < '2023-01-01';

-- Step 4: 必要に応じて繰り返し

-- Step 5: 最終的にTRUNCATEで高速クリア（全削除の場合）
-- TRUNCATE TABLE large_table;
```

エラーと対処法

1. 外部キー制約エラー

```
-- エラー例
-- DROP TABLE courses;
-- エラー: Cannot delete or update a parent row: a foreign key constraint fails

-- 対処法1: 依存関係を確認
SELECT
    table_name,
    constraint_name,
    referenced_table_name
FROM information_schema.referential_constraints
WHERE referenced_table_name = 'courses';

-- 対処法2: 子テーブルから順に削除
DROP TABLE grades;
DROP TABLE student_courses;
DROP TABLE course_schedule;
DROP TABLE courses;

-- 対処法3: 外部キー制約の一時無効化（慎重に）
SET FOREIGN_KEY_CHECKS = 0;
DROP TABLE courses;
SET FOREIGN_KEY_CHECKS = 1;
```

2. TRUNCATE制約エラー

```
-- エラー例
-- TRUNCATE TABLE students;
-- エラー: Cannot truncate a table referenced in a foreign key constraint

-- 対処法1: 外部キー制約を一時的に削除
ALTER TABLE grades DROP FOREIGN KEY fk_grades_student;
TRUNCATE TABLE students;
ALTER TABLE grades ADD CONSTRAINT fk_grades_student
FOREIGN KEY (student_id) REFERENCES students(student_id);

-- 対処法2: DELETEを使用
DELETE FROM students;
-- 注意: AUTO_INCREMENTはリセットされない

-- 対処法3: 外部キー制約の一時無効化
SET FOREIGN_KEY_CHECKS = 0;
TRUNCATE TABLE students;
SET FOREIGN_KEY_CHECKS = 1;
```

3. 権限エラー

```
-- エラー例
-- DROP TABLE some_table;
-- エラー: Access denied; you need the DROP privilege for this operation

-- 対処法: 管理者に権限確認を依頼
SHOW GRANTS FOR CURRENT_USER();

-- 必要な権限の確認
-- DROP権限、ALTER権限などが必要
```

パフォーマンスと容量の考慮

1. 削除操作のパフォーマンス比較

```
-- 大量データでのパフォーマンステスト用テーブル作成
CREATE TABLE performance_test (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 大量データ挿入（時間測定）
INSERT INTO performance_test (data)
SELECT CONCAT('test_data_', n)
```

```
FROM (
    SELECT a.N + b.N * 10 + c.N * 100 + d.N * 1000 + 1 n
    FROM
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
a,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
b,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
c,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9) d
) numbers
LIMIT 10000;

-- データ量確認
SELECT COUNT(*) as total_records FROM performance_test;

-- DELETE文での削除（時間測定）
-- SET @start_time = NOW(6);
-- DELETE FROM performance_test;
-- SET @end_time = NOW(6);
-- SELECT TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time) as
delete_microseconds;

-- TRUNCATE文での削除（時間測定）
SET @start_time = NOW(6);
TRUNCATE TABLE performance_test;
SET @end_time = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time) as
truncate_microseconds;
```

2. 容量回復の確認

```
-- テーブルサイズの確認
SELECT
    table_name,
    table_rows,
    data_length / 1024 / 1024 as data_mb,
    index_length / 1024 / 1024 as index_mb,
    (data_length + index_length) / 1024 / 1024 as total_mb
FROM information_schema.tables
WHERE table_schema = DATABASE()
AND table_name = 'performance_test';

-- TRUNCATE後の容量確認
TRUNCATE TABLE performance_test;

-- 再度サイズ確認
```

```
SELECT
    table_name,
    table_rows,
    data_length / 1024 / 1024 as data_mb,
    index_length / 1024 / 1024 as index_mb,
    (data_length + index_length) / 1024 / 1024 as total_mb
FROM information_schema.tables
WHERE table_schema = DATABASE()
AND table_name = 'performance_test';
```

ベストプラクティス

1. 削除操作の標準手順

```
-- 標準的な削除手順テンプレート

-- Phase 1: 事前確認
-- 1.1 依存関係の確認
SELECT table_name, referenced_table_name
FROM information_schema.referential_constraints
WHERE constraint_schema = DATABASE();

-- 1.2 データ量の確認
SELECT COUNT(*) as record_count FROM target_table;

-- 1.3 最終アクセス時刻の確認
SELECT MAX(updated_at) as last_update FROM target_table;

-- Phase 2: バックアップ作成（重要なデータの場合）
CREATE TABLE target_table_backup AS SELECT * FROM target_table;

-- Phase 3: 削除実行
TRUNCATE TABLE target_table;
-- または
-- DROP TABLE target_table;

-- Phase 4: 確認
-- データベース一覧やレコード数で確認

-- Phase 5: クリーンアップ（問題なければバックアップ削除）
-- DROP TABLE target_table_backup;
```

2. 自動化スクリプトの例

```
-- ログテーブル自動クリーンアップのプロシージャ例
DELIMITER //

CREATE PROCEDURE CleanupOldLogs(IN retention_days INT)
BEGIN
```

```
    DECLARE record_count INT;

    -- 削除対象レコード数の確認
    SELECT COUNT(*) INTO record_count
    FROM access_logs
    WHERE created_at < DATE_SUB(NOW(), INTERVAL retention_days DAY);

    -- ログ出力
    INSERT INTO cleanup_log (table_name, deleted_count, cleanup_date)
    VALUES ('access_logs', record_count, NOW());

    -- 実際の削除
    DELETE FROM access_logs
    WHERE created_at < DATE_SUB(NOW(), INTERVAL retention_days DAY);

END //

DELIMITER ;

-- 使用例: 30日より古いログを削除
-- CALL CleanupOldLogs(30);
```

3. 環境別の削除戦略

```
-- 開発環境：積極的なクリーンアップ
-- 毎日リセット
-- TRUNCATE TABLE session_data;
-- TRUNCATE TABLE temp_calculations;

-- ステージング環境：定期的なクリーンアップ
-- 週次でテストデータリセット
-- TRUNCATE TABLE test_results;

-- 本番環境：慎重な削除
-- 長期保存後の削除、必ずバックアップ作成
-- CREATE TABLE old_logs_backup AS SELECT * FROM logs WHERE created_at < '2023-01-
01';
-- DELETE FROM logs WHERE created_at < '2023-01-01';
```

練習問題

問題34-1：基本的なテーブル削除

以下の手順を実行してください：

1. `practice_table1`という名前のテーブルを作成（id: INT PRIMARY KEY、name: VARCHAR(50)）
2. テストデータを3件挿入
3. `DROP TABLE`文でテーブルを削除
4. 削除されたことを確認

問題34-2：TRUNCATE文の使用

以下の手順を実行してください：

1. `practice_table2`という名前のテーブルを作成（id: INT AUTO_INCREMENT PRIMARY KEY、data: VARCHAR(100)、created_at: TIMESTAMP DEFAULT CURRENT_TIMESTAMP）
2. テストデータを5件挿入し、最大のidを確認
3. `TRUNCATE`文でデータをクリア
4. 新しいデータを1件挿入し、idが1から開始されることを確認

問題34-3：DELETE、TRUNCATE、DROPの比較

以下の比較実験を行ってください：

1. 同じ構造のテーブルを3つ作成（`delete_test`、`truncate_test`、`drop_test`）
2. 各テーブルに同じテストデータを挿入
3. それぞれ`DELETE`、`TRUNCATE`、`DROP TABLE`で処理
4. 実行後の状態を比較し、違いをまとめる

問題34-4：外部キー制約がある場合の削除

以下の手順で外部キー制約を考慮した削除を実行してください：

1. 親テーブル`departments`（dept_id: INT PRIMARY KEY、dept_name: VARCHAR(100)）を作成
2. 子テーブル`employees`（emp_id: INT PRIMARY KEY、name: VARCHAR(100)、dept_id: INT、外部キー制約あり）を作成
3. 両テーブルにテストデータを挿入
4. 正しい順序でテーブルを削除
5. 外部キー制約があるときに親テーブルを先に削除しようとするとどうなるか確認

問題34-5：大量データの効率的削除

以下の大量データ削除シナリオを実装してください：

1. `large_data_table`を作成（id: BIGINT AUTO_INCREMENT PRIMARY KEY、content: TEXT、created_at: TIMESTAMP）
2. 1000件以上のテストデータを挿入
3. `DELETE`文と`TRUNCATE`文の実行時間を比較
4. 各方法のメリット・デメリットを考察

問題34-6：安全な削除手順の実装

以下の安全な削除手順を実装してください：

1. 重要なデータが入った`important_table`を作成
2. バックアップテーブルを作成する手順
3. 元テーブルを削除する手順
4. 必要に応じて復元する手順
5. 最終的にバックアップテーブルも削除する手順 すべてをSQL文で実装し、各ステップで適切な確認を行ってください。

解答

解答34-1

```sql
-- 1. テーブル作成
CREATE TABLE practice_table1 (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);

-- 2. テストデータ挿入
INSERT INTO practice_table1 VALUES
(1, 'データ1'),
(2, 'データ2'),
(3, 'データ3');

-- データ確認
SELECT * FROM practice_table1;

-- 3. テーブル削除
DROP TABLE practice_table1;

-- 4. 削除確認（エラーになることを確認）
-- SELECT * FROM practice_table1;  -- エラー: Table doesn't exist

-- または存在確認
SHOW TABLES LIKE 'practice_table1';  -- 結果: 0行
```

解答34-2

```sql
-- 1. テーブル作成
CREATE TABLE practice_table2 (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. テストデータ挿入
INSERT INTO practice_table2 (data) VALUES
('データ1'), ('データ2'), ('データ3'), ('データ4'), ('データ5');

-- 最大idの確認
SELECT MAX(id) as max_id FROM practice_table2;

-- 3. TRUNCATEでデータクリア
TRUNCATE TABLE practice_table2;

-- データがクリアされたことを確認
SELECT COUNT(*) as record_count FROM practice_table2;
```

```
-- 4. 新しいデータ挿入
INSERT INTO practice_table2 (data) VALUES ('新しいデータ');

-- idが1から開始されることを確認
SELECT * FROM practice_table2;
```

解答34-3

```
-- 1. 3つのテーブル作成
CREATE TABLE delete_test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE truncate_test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE drop_test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. 各テーブルに同じデータを挿入
INSERT INTO delete_test (data) VALUES ('データA'), ('データB'), ('データC');
INSERT INTO truncate_test (data) VALUES ('データA'), ('データB'), ('データC');
INSERT INTO drop_test (data) VALUES ('データA'), ('データB'), ('データC');

-- 挿入後の状態確認
SELECT 'delete_test' as table_name, COUNT(*) as count FROM delete_test
UNION ALL
SELECT 'truncate_test', COUNT(*) FROM truncate_test
UNION ALL
SELECT 'drop_test', COUNT(*) FROM drop_test;

-- 3. それぞれの方法で処理
-- DELETE
DELETE FROM delete_test;

-- TRUNCATE
TRUNCATE TABLE truncate_test;

-- DROP
DROP TABLE drop_test;

-- 4. 結果比較
-- DELETEの結果
```

```
SELECT 'delete_test after DELETE' as status, COUNT(*) as count FROM delete_test;
DESCRIBE delete_test;  -- テーブル構造は残っている

-- TRUNCATEの結果
SELECT 'truncate_test after TRUNCATE' as status, COUNT(*) as count FROM
truncate_test;
DESCRIBE truncate_test;  -- テーブル構造は残っている

-- DROPの結果
-- DESCRIBE drop_test;  -- エラー: テーブルが存在しない

-- 新しいデータ挿入でAUTO_INCREMENTの違いを確認
INSERT INTO delete_test (data) VALUES ('DELETE後データ');
INSERT INTO truncate_test (data) VALUES ('TRUNCATE後データ');

SELECT 'DELETE後のAUTO_INCREMENT' as type, id FROM delete_test;
SELECT 'TRUNCATE後のAUTO_INCREMENT' as type, id FROM truncate_test;
```

解答34-4

```
-- 1. 親テーブル作成
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(100)
);

-- 2. 子テーブル作成（外部キー制約付き）
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(100),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);

-- 3. テストデータ挿入
INSERT INTO departments VALUES (1, '営業部'), (2, '開発部');
INSERT INTO employees VALUES (101, '田中', 1), (102, '佐藤', 2), (103, '鈴木', 1);

-- データ確認
SELECT * FROM departments;
SELECT * FROM employees;

-- 4. 親テーブルを先に削除しようとする（エラーになることを確認）
-- DROP TABLE departments;
-- エラー: Cannot delete or update a parent row: a foreign key constraint fails

-- 正しい順序での削除（子テーブルから先に）
DROP TABLE employees;
DROP TABLE departments;

-- 削除確認
```

```
SHOW TABLES LIKE 'departments';
SHOW TABLES LIKE 'employees';
```

解答34-5

```
-- 1. テーブル作成
CREATE TABLE large_data_table (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    content TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. 大量データ挿入（1000件以上）
INSERT INTO large_data_table (content)
SELECT CONCAT('コンテンツ', n) as content
FROM (
    SELECT a.N + b.N * 10 + c.N * 100 + d.N * 1000 + 1 n
    FROM
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
a,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
b,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
c,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9) d
) numbers
LIMIT 2000;

-- データ量確認
SELECT COUNT(*) as total_records FROM large_data_table;

-- 3. DELETEの実行時間測定
-- まずデータを複製
CREATE TABLE large_data_delete AS SELECT * FROM large_data_table;
CREATE TABLE large_data_truncate AS SELECT * FROM large_data_table;

-- DELETE文の時間測定
SET @start_time = NOW(6);
DELETE FROM large_data_delete;
SET @end_time = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time) as delete_microseconds;

-- TRUNCATE文の時間測定
SET @start_time = NOW(6);
TRUNCATE TABLE large_data_truncate;
SET @end_time = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time) as
```

```
truncate_microseconds;

-- 4. 考察
/*
DELETEのメリット：
- 条件指定可能
- ロールバック可能
- トリガー実行

DELETEのデメリット：
- 処理が遅い
- ログ容量が大きい

TRUNCATEのメリット：
- 高速処理
- AUTO_INCREMENT リセット
- 少ないログ使用

TRUNCATEのデメリット：
- 条件指定不可
- ロールバック不可
- 外部キー制約で制限
*/

-- クリーンアップ
DROP TABLE large_data_table;
DROP TABLE large_data_delete;
DROP TABLE large_data_truncate;
```

解答34-6

```
-- 1. 重要なテーブル作成
CREATE TABLE important_table (
    id INT AUTO_INCREMENT PRIMARY KEY,
    critical_data VARCHAR(255),
    amount DECIMAL(10,2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 重要なデータ挿入
INSERT INTO important_table (critical_data, amount) VALUES
('重要データ1', 1000.00),
('重要データ2', 2000.00),
('重要データ3', 1500.00);

-- データ確認
SELECT * FROM important_table;

-- 2. バックアップテーブル作成
CREATE TABLE important_table_backup_20231201 AS
SELECT * FROM important_table;
```

```
-- バックアップの確認
SELECT COUNT(*) as backup_count FROM important_table_backup_20231201;
SELECT * FROM important_table_backup_20231201;

-- 3. 元テーブル削除前の最終確認
SELECT
    COUNT(*) as original_count,
    SUM(amount) as total_amount,
    MAX(created_at) as latest_record
FROM important_table;

-- 元テーブルの削除
DROP TABLE important_table;

-- 削除確認
SHOW TABLES LIKE 'important_table';

-- 4. 復元手順（必要に応じて）
CREATE TABLE important_table AS
SELECT * FROM important_table_backup_20231201;

-- 復元確認
SELECT COUNT(*) as restored_count FROM important_table;
SELECT * FROM important_table;

-- データ整合性確認
SELECT
    o.id, o.critical_data, o.amount,
    CASE WHEN b.id IS NOT NULL THEN '一致' ELSE '不一致' END as integrity_check
FROM important_table o
LEFT JOIN important_table_backup_20231201 b ON o.id = b.id AND o.critical_data =
b.critical_data;

-- 5. 最終クリーンアップ（すべて確認後）
-- バックアップテーブルの削除
DROP TABLE important_table_backup_20231201;

-- 最終確認
SHOW TABLES LIKE '%important%';
```

まとめ

この章では、DROP TABLE文とTRUNCATE文について詳しく学びました：

1. **基本概念の理解**：
    - DROP TABLE：テーブル構造とデータの完全削除
    - TRUNCATE：テーブル構造を保持してデータのみ高速削除
    - DELETE文との違いと使い分け
2. **基本構文と操作**：

- DROP TABLEの基本構文とIF EXISTSオプション
- TRUNCATEの基本構文とAUTO_INCREMENTリセット
- 複数テーブルの同時削除

3. **外部キー制約の考慮**：

- 参照整合性による削除制限
- CASCADE削除の活用
- 制約の一時無効化（注意が必要）

4. **パフォーマンスの比較**：

- DELETE、TRUNCATE、DROPの処理速度差
- ログ記録量の違い
- ロールバック可能性の違い

5. **実践的な削除戦略**：

- ログテーブルの定期クリア
- テスト環境のデータリセット
- 一時テーブルの管理

6. **安全な削除手順**：

- 事前のバックアップ作成
- 依存関係の確認
- 段階的な削除実行

7. **エラー対処法**：

- 外部キー制約エラーの解決
- 権限エラーへの対応
- TRUNCATE制約エラーの回避

8. **ベストプラクティス**：

- 環境別の削除戦略
- 自動化スクリプトの活用
- リスク管理の重要性

DROP TABLE文とTRUNCATE文は非常に強力ですが、**一度実行すると元に戻せない**危険な操作です。特に本番環境では、必ずバックアップを作成し、十分な確認を行ってから実行することが重要です。

次の章では、「制約：主キー、外部キー、CHECK制約」について学び、データの整合性を保つための制約機能を詳しく理解していきます。

---

35. 制約：主キー、外部キー、CHECK制約

はじめに

前章では、テーブルの削除とクリアについて学習しました。この章では、データベースの整合性と品質を保つための「制約（Constraint）」について詳しく学習します。

制約は、テーブルに格納されるデータが満たすべき条件や規則を定義する仕組みです。制約を適切に設定することで、不正なデータの挿入を防ぎ、データベース全体の整合性を保つことができます。

制約が重要となる場面の例：

- 「学生IDは重複してはいけない（一意性の保証）」
- 「成績の点数は0点以上100点以下でなければならない（値の範囲制限）」
- 「学生の受講記録は、実在する学生と講座を参照しなければならない（参照整合性）」
- 「教師名は必須項目であり、空白は許可しない（必須項目の保証）」
- 「メールアドレスは一意でなければならない（重複防止）」
- 「入学日は現在日より前でなければならない（論理的整合性）」
- 「学年は1年から6年の範囲内でなければならない（値の妥当性）」

この章では、MySQLの主要な制約である主キー制約、外部キー制約、CHECK制約、UNIQUE制約、NOT NULL制約について詳しく学んでいきます。

制約とは

制約（Constraint）は、テーブルのカラムやテーブル全体に対して設定する、データの整合性を保つための規則です。制約に違反するデータの挿入や更新は自動的に拒否され、エラーが発生します。

> **用語解説**：
>
> - **制約（Constraint）**：データベースに格納されるデータが満たすべき条件や規則です。
> - **主キー制約（PRIMARY KEY）**：テーブル内の各行を一意に識別するための制約です。
> - **外部キー制約（FOREIGN KEY）**：他のテーブルとの関連性を保証する制約です。
> - **CHECK制約**：カラムの値が特定の条件を満たすことを保証する制約です。
> - **UNIQUE制約**：カラムの値が一意（重複なし）であることを保証する制約です。
> - **NOT NULL制約**：カラムにNULL値の格納を禁止する制約です。
> - **DEFAULT制約**：カラムの値が指定されなかった場合のデフォルト値を設定する制約です。
> - **参照整合性**：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。
> - **カスケード動作**：参照先の変更時に参照元も自動的に変更される動作です。
> - **制約違反**：データが制約の条件を満たさない状態で、エラーが発生します。

制約の種類と概要

制約の種類	目的	例
PRIMARY KEY	行の一意識別	学生ID、講座ID
FOREIGN KEY	テーブル間の関連性保証	成績テーブルの学生ID
UNIQUE	値の一意性保証	メールアドレス、学籍番号
NOT NULL	必須項目の保証	学生名、講座名
CHECK	値の範囲・条件制限	成績（0-100点）、学年（1-6年）

制約の種類	目的	例
DEFAULT	デフォルト値の設定	登録日時、ステータス

主キー制約（PRIMARY KEY）

1. 主キー制約の基本概念

主キー制約は、テーブル内の各行を一意に識別するためのカラム（または複数カラムの組み合わせ）を指定します。

主キーの特性

- **一意性**：同じ値を持つ行は存在できません
- **NOT NULL**：NULL値は許可されません
- **不変性**：一度設定された主キー値は変更すべきではありません
- **単一性**：1つのテーブルに1つだけ設定できます

2. 主キー制約の設定方法

テーブル作成時の設定

```
-- 方法1: カラム定義時に指定
CREATE TABLE students_pk_demo (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL
);

-- 方法2: テーブル制約として指定
CREATE TABLE teachers_pk_demo (
    teacher_id BIGINT NOT NULL,
    teacher_name VARCHAR(100) NOT NULL,

    PRIMARY KEY (teacher_id)
);

-- 方法3: 複合主キー（複数カラムの組み合わせ）
CREATE TABLE enrollment_pk_demo (
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    enrollment_date DATE NOT NULL,

    PRIMARY KEY (student_id, course_id)
);
```

AUTO_INCREMENTとの組み合わせ

```
-- 自動増加する主キー
CREATE TABLE auto_pk_demo (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- データ挿入（IDは自動設定）
INSERT INTO auto_pk_demo (data) VALUES ('データ1'), ('データ2');

-- 結果確認
SELECT * FROM auto_pk_demo;
```

3. 既存テーブルへの主キー追加

```
-- 主キーなしのテーブル作成
CREATE TABLE no_pk_table (
    id INT,
    name VARCHAR(100)
);

-- データ挿入
INSERT INTO no_pk_table VALUES (1, 'データ1'), (2, 'データ2');

-- 主キー制約の追加
ALTER TABLE no_pk_table ADD PRIMARY KEY (id);

-- 確認
DESCRIBE no_pk_table;
```

4. 主キー制約のエラー例

```
-- 重複エラーのテスト
CREATE TABLE pk_error_test (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);

INSERT INTO pk_error_test VALUES (1, 'データ1');
-- INSERT INTO pk_error_test VALUES (1, 'データ2');  -- エラー: Duplicate entry '1'

-- NULL値エラーのテスト
-- INSERT INTO pk_error_test VALUES (NULL, 'データ3');  -- エラー: Column 'id' cannot
be null
```

外部キー制約（FOREIGN KEY）

1. 外部キー制約の基本概念

外部キー制約は、あるテーブルのカラムが別のテーブルの主キーを参照することを保証します。これにより、テーブル間の参照整合性が維持されます。

外部キーの特性

- **参照整合性**：参照先のレコードが存在することを保証
- **カスケード動作**：参照先の変更時の動作を制御
- **パフォーマンス**：結合処理の最適化に貢献

2. 外部キー制約の設定方法

基本的な外部キー設定

```
-- 親テーブル（参照される側）
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(100) NOT NULL
);

-- 子テーブル（参照する側）
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100) NOT NULL,
    dept_id INT,

    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);

-- テストデータ挿入
INSERT INTO departments VALUES (1, '営業部'), (2, '開発部'), (3, '総務部');
INSERT INTO employees VALUES (101, '田中太郎', 1), (102, '佐藤花子', 2);
```

カスケード動作の設定

```
-- カスケード動作付きの外部キー制約
CREATE TABLE students_fk_demo (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE
);

CREATE TABLE enrollments_fk_demo (
    enrollment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_name VARCHAR(100),
    enrollment_date DATE,
```

```
    FOREIGN KEY (student_id) REFERENCES students_fk_demo(student_id)
        ON DELETE CASCADE        -- 親が削除されたら子も削除
        ON UPDATE CASCADE        -- 親が更新されたら子も更新
);

-- テストデータ
INSERT INTO students_fk_demo VALUES (301, '山田次郎', 'yamada@example.com');
INSERT INTO enrollments_fk_demo (student_id, course_name, enrollment_date)
VALUES (301, 'データベース基礎', '2025-04-01');

-- カスケード削除のテスト
DELETE FROM students_fk_demo WHERE student_id = 301;

-- 関連する enrollment レコードも自動削除されることを確認
SELECT * FROM enrollments_fk_demo WHERE student_id = 301;
```

3. カスケード動作の種類

動作	説明	使用場面
CASCADE	親の変更に合わせて子も変更	強い関連性がある場合
SET NULL	親が削除されたら子はNULLに	参照が必須でない場合
SET DEFAULT	親が削除されたらデフォルト値に	デフォルト値が設定されている場合
RESTRICT	子がある限り親の変更を禁止	データ保護が重要な場合
NO ACTION	RESTRICTと同様（MySQLでは同じ）	RESTRICTと同じ

```
-- 異なるカスケード動作の例
CREATE TABLE courses_fk_demo (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL
);

CREATE TABLE grades_fk_demo (
    grade_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT,
    course_id VARCHAR(16),
    score DECIMAL(5,2),

    FOREIGN KEY (student_id) REFERENCES students_fk_demo(student_id)
        ON DELETE SET NULL       -- 学生削除時はNULLに
        ON UPDATE CASCADE,       -- 学生ID更新時は追従

    FOREIGN KEY (course_id) REFERENCES courses_fk_demo(course_id)
        ON DELETE RESTRICT       -- 講座に成績がある場合は削除禁止
        ON UPDATE CASCADE        -- 講座ID更新時は追従
);
```

4. 外部キー制約のエラー例

```
-- 参照整合性エラーのテスト
-- INSERT INTO employees VALUES (103, '鈴木一郎', 99);  -- エラー: 存在しない部署ID

-- 親レコード削除エラーのテスト（RESTRICT設定時）
-- DELETE FROM departments WHERE dept_id = 1;  -- エラー: 参照している子レコードが存在
```

CHECK制約（MySQL 8.0以降）

1. CHECK制約の基本概念

CHECK制約は、カラムに格納される値が特定の条件を満たすことを保証します。値の範囲制限や形式チェックに使用されます。

2. CHECK制約の設定方法

基本的なCHECK制約

```
-- 成績管理テーブルのCHECK制約例
CREATE TABLE student_scores (
    score_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    subject VARCHAR(100) NOT NULL,
    score DECIMAL(5,2) NOT NULL,
    grade_level INT NOT NULL,
    exam_date DATE NOT NULL,

    -- 点数の範囲制限
    CHECK (score >= 0 AND score <= 100),

    -- 学年の範囲制限
    CHECK (grade_level >= 1 AND grade_level <= 6),

    -- 試験日は現在日以前
    CHECK (exam_date <= CURRENT_DATE)
);

-- 有効なデータの挿入
INSERT INTO student_scores (student_id, subject, score, grade_level, exam_date)
VALUES (301, '数学', 85.5, 3, '2025-05-20');

-- 無効なデータの挿入（エラーになる）
-- INSERT INTO student_scores (student_id, subject, score, grade_level, exam_date)
-- VALUES (302, '英語', 105, 3, '2025-05-20');  -- エラー: score > 100
```

複雑なCHECK制約

```
-- より複雑な条件のCHECK制約
CREATE TABLE course_registrations (
    registration_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    registration_date DATE NOT NULL,
    status ENUM('pending', 'approved', 'rejected', 'cancelled') NOT NULL,
    semester ENUM('spring', 'summer', 'fall', 'winter') NOT NULL,
    academic_year INT NOT NULL,

    -- 年度の妥当性チェック
    CHECK (academic_year >= 2000 AND academic_year <= 2100),

    -- 登録日と年度の整合性チェック
    CHECK (YEAR(registration_date) = academic_year OR YEAR(registration_date) =
academic_year - 1),

    -- ステータスと日付の論理チェック
    CHECK (status = 'pending' OR registration_date < CURRENT_DATE)
);
```

文字列パターンのCHECK制約

```
-- 文字列形式のチェック
CREATE TABLE student_contacts (
    contact_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    email VARCHAR(255),
    phone VARCHAR(20),
    postal_code VARCHAR(10),

    -- メールアドレスの形式チェック（簡単な例）
    CHECK (email IS NULL OR email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-
zA-Z]{2,}$'),

    -- 電話番号の形式チェック（日本の形式）
    CHECK (phone IS NULL OR phone REGEXP '^[0-9]{2,4}-[0-9]{2,4}-[0-9]{4}$'),

    -- 郵便番号の形式チェック（日本の形式）
    CHECK (postal_code IS NULL OR postal_code REGEXP '^[0-9]{3}-[0-9]{4}$')
);

-- 有効なデータの挿入
INSERT INTO student_contacts (student_id, email, phone, postal_code)
VALUES (301, 'student@example.com', '03-1234-5678', '100-0001');

-- 無効なデータの挿入テスト
-- INSERT INTO student_contacts (student_id, email, phone, postal_code)
-- VALUES (302, 'invalid-email', '123', 'invalid');  -- エラー: 形式違反
```

3. CHECK制約の管理

```
-- CHECK制約の追加
ALTER TABLE student_scores
ADD CONSTRAINT chk_score_not_negative CHECK (score >= 0);

-- CHECK制約の削除
ALTER TABLE student_scores
DROP CHECK chk_score_not_negative;

-- CHECK制約の確認
SELECT
    constraint_name,
    check_clause
FROM information_schema.check_constraints
WHERE constraint_schema = DATABASE();
```

UNIQUE制約

1. UNIQUE制約の基本概念

UNIQUE制約は、カラムの値が一意（重複なし）であることを保証します。主キーとは異なり、NULL値は許可されます（ただし、複数のNULL値は許可される場合があります）。

2. UNIQUE制約の設定方法

単一カラムのUNIQUE制約

```
-- 学生テーブルのUNIQUE制約例
CREATE TABLE students_unique_demo (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE,            -- メールアドレスは一意
    student_number VARCHAR(20) UNIQUE,    -- 学籍番号は一意
    phone VARCHAR(20)
);

-- 有効なデータの挿入
INSERT INTO students_unique_demo VALUES
(1, '田中太郎', 'tanaka@example.com', 'S2025001', '090-1234-5678'),
(2, '佐藤花子', 'sato@example.com', 'S2025002', '090-2345-6789');

-- UNIQUE制約違反のテスト
-- INSERT INTO students_unique_demo VALUES
-- (3, '鈴木次郎', 'tanaka@example.com', 'S2025003', '090-3456-7890');  -- エラー: 重複email
```

複合UNIQUE制約

```
-- 複数カラムの組み合わせでの一意性
CREATE TABLE class_schedules_unique (
    schedule_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    classroom_id VARCHAR(16) NOT NULL,
    day_of_week TINYINT NOT NULL,
    period INT NOT NULL,
    course_id VARCHAR(16),

    -- 同じ教室、同じ曜日、同じ時限は重複不可
    UNIQUE KEY unique_classroom_time (classroom_id, day_of_week, period)
);

-- 有効なデータ
INSERT INTO class_schedules_unique (classroom_id, day_of_week, period, course_id)
VALUES ('101A', 1, 1, 'MATH001'), ('101A', 1, 2, 'ENG001'), ('102B', 1, 1,
'SCI001');

-- 重複エラーのテスト
-- INSERT INTO class_schedules_unique (classroom_id, day_of_week, period,
course_id)
-- VALUES ('101A', 1, 1, 'HIST001');  -- エラー: 同じ教室・曜日・時限の重複
```

3. UNIQUE制約の管理

```
-- UNIQUE制約の追加
ALTER TABLE students_unique_demo
ADD CONSTRAINT uk_phone UNIQUE (phone);

-- UNIQUE制約の削除
ALTER TABLE students_unique_demo
DROP INDEX uk_phone;

-- UNIQUE制約の確認
SHOW INDEX FROM students_unique_demo WHERE Non_duplicate = 0;
```

NOT NULL制約

1. NOT NULL制約の基本概念

NOT NULL制約は、カラムにNULL値の格納を禁止し、必ず何らかの値が入力されることを保証します。

2. NOT NULL制約の設定方法

```
-- NOT NULL制約の例
CREATE TABLE required_fields_demo (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,              -- 必須項目
```

```
    email VARCHAR(255) NOT NULL,              -- 必須項目
    phone VARCHAR(20),                        -- オプション項目
    description TEXT,                         -- オプション項目
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP   -- 必須・デフォルト値あり
);

-- 有効なデータの挿入
INSERT INTO required_fields_demo (name, email, phone)
VALUES ('山田太郎', 'yamada@example.com', '090-1111-2222');

-- NULL値挿入エラーのテスト
-- INSERT INTO required_fields_demo (name, phone)
-- VALUES ('田中次郎', '090-3333-4444');  -- エラー: emailがNULL
```

3. NOT NULL制約の管理

```
-- NOT NULL制約の追加（既存データの確認が必要）
-- まずNULL値を適切な値で更新
UPDATE required_fields_demo SET phone = '未登録' WHERE phone IS NULL;

-- NOT NULL制約を追加
ALTER TABLE required_fields_demo
MODIFY COLUMN phone VARCHAR(20) NOT NULL;

-- NOT NULL制約の削除（NULL許可に変更）
ALTER TABLE required_fields_demo
MODIFY COLUMN phone VARCHAR(20);
```

DEFAULT制約

1. DEFAULT制約の基本概念

DEFAULT制約は、INSERT文でカラムの値が指定されなかった場合に自動的に設定される値を定義します。

2. DEFAULT制約の設定方法

```
-- DEFAULT制約の例
CREATE TABLE default_values_demo (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    status VARCHAR(20) DEFAULT 'active',              -- 文字列のデフォルト
    priority INT DEFAULT 1,                           -- 数値のデフォルト
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,   -- 現在日時
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
               ON UPDATE CURRENT_TIMESTAMP,           -- 更新時自動更新
    is_enabled BOOLEAN DEFAULT TRUE,                  -- 真偽値のデフォルト
    discount_rate DECIMAL(5,2) DEFAULT 0.00          -- 小数のデフォルト
);
```

```
-- デフォルト値を使用した挿入
INSERT INTO default_values_demo (name) VALUES ('テストユーザー');

-- 結果確認
SELECT * FROM default_values_demo;

-- 明示的な値の指定
INSERT INTO default_values_demo (name, status, priority, is_enabled)
VALUES ('カスタムユーザー', 'inactive', 5, FALSE);
```

3. DEFAULT制約の管理

```
-- DEFAULT制約の追加
ALTER TABLE default_values_demo
ALTER COLUMN priority SET DEFAULT 3;

-- DEFAULT制約の削除
ALTER TABLE default_values_demo
ALTER COLUMN priority DROP DEFAULT;

-- DEFAULT値の確認
DESCRIBE default_values_demo;
```

実践的な制約設計

例1：包括的な学生管理テーブル

```
-- 実際の学生管理システムの例
CREATE TABLE comprehensive_students (
    -- 主キー
    student_id BIGINT AUTO_INCREMENT PRIMARY KEY,

    -- 基本情報（必須項目）
    student_number VARCHAR(20) NOT NULL UNIQUE,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,

    -- 連絡先情報
    email VARCHAR(255) UNIQUE,
    phone VARCHAR(20),

    -- 学籍情報
    admission_date DATE NOT NULL,
    graduation_date DATE,
    grade_level INT NOT NULL,
    status ENUM('enrolled', 'graduated', 'withdrawn', 'suspended') DEFAULT
'enrolled',

    -- 個人情報
```

```
    birth_date DATE,
    gender ENUM('male', 'female', 'other', 'prefer_not_to_say'),

    -- システム情報
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,

    -- CHECK制約
    CHECK (grade_level >= 1 AND grade_level <= 6),
    CHECK (birth_date <= CURRENT_DATE),
    CHECK (admission_date >= '2000-01-01'),
    CHECK (graduation_date IS NULL OR graduation_date >= admission_date),
    CHECK (email IS NULL OR email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-
zA-Z]{2,}$'),

    -- インデックス
    INDEX idx_student_number (student_number),
    INDEX idx_name (last_name, first_name),
    INDEX idx_grade_status (grade_level, status),
    INDEX idx_admission_date (admission_date)
);
```

例2：成績管理システムの制約設計

```
-- 包括的な成績管理テーブル
CREATE TABLE comprehensive_grades (
    -- 主キー
    grade_id BIGINT AUTO_INCREMENT PRIMARY KEY,

    -- 外部キー
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    teacher_id BIGINT NOT NULL,

    -- 評価情報
    assessment_type ENUM('quiz', 'midterm', 'final', 'project', 'homework') NOT
NULL,
    assessment_name VARCHAR(100) NOT NULL,
    score DECIMAL(5,2) NOT NULL,
    max_score DECIMAL(5,2) NOT NULL DEFAULT 100.00,
    weight DECIMAL(5,2) DEFAULT 1.00,

    -- 日付情報
    assessment_date DATE NOT NULL,
    submission_deadline DATETIME,
    submitted_at DATETIME,
    graded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    -- ステータス
    status ENUM('draft', 'submitted', 'graded', 'returned') DEFAULT 'draft',
```

```
    -- 追加情報
    comments TEXT,
    is_extra_credit BOOLEAN DEFAULT FALSE,

    -- 外部キー制約
    FOREIGN KEY (student_id) REFERENCES comprehensive_students(student_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,

    -- CHECK制約
    CHECK (score >= 0 AND score <= max_score),
    CHECK (max_score > 0),
    CHECK (weight >= 0 AND weight <= 10),
    CHECK (assessment_date <= CURRENT_DATE),
    CHECK (submitted_at IS NULL OR submitted_at <= CURRENT_TIMESTAMP),
    CHECK (submission_deadline IS NULL OR assessment_date <=
DATE(submission_deadline)),

    -- UNIQUE制約（同一学生・講座・評価タイプ・評価名は一意）
    UNIQUE KEY unique_assessment (student_id, course_id, assessment_type,
assessment_name),

    -- インデックス
    INDEX idx_student_course (student_id, course_id),
    INDEX idx_assessment_date (assessment_date),
    INDEX idx_teacher_course (teacher_id, course_id),
    INDEX idx_status (status)
);
```

制約のエラーと対処法

1. 主キー制約違反

```
-- 重複エラーの対処
CREATE TABLE pk_conflict_test (
    id INT PRIMARY KEY,
    data VARCHAR(100)
);

INSERT INTO pk_conflict_test VALUES (1, 'データ1');

-- 重複挿入の試行
-- INSERT INTO pk_conflict_test VALUES (1, 'データ2');  -- エラー

-- 対処法1: INSERT IGNORE
INSERT IGNORE INTO pk_conflict_test VALUES (1, 'データ2');  -- エラーにならない
```

```
-- 対処法2: ON DUPLICATE KEY UPDATE
INSERT INTO pk_conflict_test VALUES (1, 'データ2更新')
ON DUPLICATE KEY UPDATE data = VALUES(data);

-- 結果確認
SELECT * FROM pk_conflict_test;
```

2. 外部キー制約違反

```
-- 参照整合性エラーの対処
CREATE TABLE fk_parent (id INT PRIMARY KEY, name VARCHAR(50));
CREATE TABLE fk_child (
    id INT PRIMARY KEY,
    parent_id INT,
    data VARCHAR(50),
    FOREIGN KEY (parent_id) REFERENCES fk_parent(id)
);

INSERT INTO fk_parent VALUES (1, '親1'), (2, '親2');

-- 存在しない親への参照
-- INSERT INTO fk_child VALUES (1, 99, '子1');  -- エラー

-- 対処法1: 事前に親の存在確認
INSERT INTO fk_child (id, parent_id, data)
SELECT 1, 1, '子1'
WHERE EXISTS (SELECT 1 FROM fk_parent WHERE id = 1);

-- 対処法2: 親データの事前挿入
INSERT IGNORE INTO fk_parent VALUES (99, '新親');
INSERT INTO fk_child VALUES (2, 99, '子2');
```

3. CHECK制約違反

```
-- CHECK制約エラーの対処
CREATE TABLE check_error_test (
    id INT PRIMARY KEY,
    score INT,
    grade CHAR(1),
    CHECK (score >= 0 AND score <= 100),
    CHECK (grade IN ('A', 'B', 'C', 'D', 'F'))
);

-- 範囲外データの挿入
-- INSERT INTO check_error_test VALUES (1, 150, 'A');  -- エラー

-- 対処法1: データの事前検証と修正
INSERT INTO check_error_test
SELECT 1, LEAST(GREATEST(score_input, 0), 100), grade_input
```

```
FROM (SELECT 150 as score_input, 'A' as grade_input) input;

-- 対処法2: 条件付き挿入
INSERT INTO check_error_test (id, score, grade)
SELECT 2, 85, 'B'
WHERE 85 BETWEEN 0 AND 100 AND 'B' IN ('A', 'B', 'C', 'D', 'F');
```

制約の確認と管理

1. 制約情報の確認

```
-- テーブルの制約一覧確認
SELECT
    constraint_name,
    constraint_type,
    table_name
FROM information_schema.table_constraints
WHERE table_schema = DATABASE()
AND table_name = 'comprehensive_students';

-- 外部キー制約の詳細確認
SELECT
    constraint_name,
    table_name,
    column_name,
    referenced_table_name,
    referenced_column_name,
    delete_rule,
    update_rule
FROM information_schema.key_column_usage k
JOIN information_schema.referential_constraints r
    ON k.constraint_name = r.constraint_name
WHERE k.constraint_schema = DATABASE();

-- CHECK制約の確認
SELECT
    constraint_name,
    check_clause
FROM information_schema.check_constraints
WHERE constraint_schema = DATABASE();
```

2. 制約の無効化と有効化

```
-- 外部キー制約の一時無効化（注意深く使用）
SET FOREIGN_KEY_CHECKS = 0;

-- データの一括操作
-- ...
```

```
-- 外部キー制約の再有効化
SET FOREIGN_KEY_CHECKS = 1;

-- CHECK制約の無効化（MySQL 8.0.16以降）
-- ALTER TABLE table_name ALTER CHECK constraint_name NOT ENFORCED;
-- ALTER TABLE table_name ALTER CHECK constraint_name ENFORCED;
```

ベストプラクティス

1. 制約設計の原則

```
-- 良い制約設計の例
CREATE TABLE best_practice_example (
    -- 1. 適切な主キー設計
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    uuid CHAR(36) UNIQUE DEFAULT (UUID()),

    -- 2. 必須項目の明確化
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,

    -- 3. 適切なデフォルト値
    status ENUM('active', 'inactive', 'pending') DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    -- 4. 論理的な制約
    birth_date DATE,
    registration_date DATE DEFAULT (CURRENT_DATE),

    -- 5. 包括的なCHECK制約
    CHECK (birth_date IS NULL OR birth_date <= CURRENT_DATE),
    CHECK (registration_date >= birth_date OR birth_date IS NULL),
    CHECK (email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'),

    -- 6. 適切なインデックス
    INDEX idx_email (email),
    INDEX idx_status (status),
    INDEX idx_registration_date (registration_date)
);
```

2. 制約の段階的実装

```
-- Phase 1: 基本制約
CREATE TABLE gradual_constraints (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

```sql
-- Phase 2: 一意性制約追加
ALTER TABLE gradual_constraints
ADD COLUMN email VARCHAR(255) UNIQUE;

-- Phase 3: CHECK制約追加
ALTER TABLE gradual_constraints
ADD COLUMN age INT,
ADD CONSTRAINT chk_age CHECK (age >= 0 AND age <= 150);

-- Phase 4: 外部キー制約追加
ALTER TABLE gradual_constraints
ADD COLUMN department_id INT,
ADD FOREIGN KEY (department_id) REFERENCES departments(dept_id);
```

3. パフォーマンスを考慮した制約設計

```sql
-- パフォーマンスを考慮した制約設計
CREATE TABLE performance_optimized (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,

    -- 頻繁に検索されるカラムにインデックス付きUNIQUE制約
    code VARCHAR(20) UNIQUE,

    -- 複合インデックスを考慮したUNIQUE制約
    category_id INT,
    subcategory_id INT,
    item_name VARCHAR(100),

    -- 範囲検索に適したカラムの制約
    price DECIMAL(10,2),
    stock_quantity INT DEFAULT 0,

    -- CHECK制約（インデックスは作成されない）
    CHECK (price >= 0),
    CHECK (stock_quantity >= 0),

    -- 複合UNIQUE制約（複合インデックスも作成）
    UNIQUE KEY unique_item (category_id, subcategory_id, item_name),

    -- 個別インデックス
    INDEX idx_price (price),
    INDEX idx_stock (stock_quantity)
);
```

練習問題

問題35-1：基本的な制約設定

以下の要件を満たすテーブル`library_books`を作成してください：

- 主キー：book_id（自動増加する整数）
- ISBN：isbn（13文字、必須、一意）
- タイトル：title（最大200文字、必須）
- 著者：author（最大100文字、必須）
- 出版年：publication_year（整数、1900年以上現在年以下）
- 価格：price（小数点2桁、0以上）
- ステータス：status（'available', 'borrowed', 'maintenance'のいずれか、デフォルト'available'）

問題35-2：外部キー制約の実装

以下の2つのテーブルを作成し、適切な外部キー制約を設定してください：

1. categoriesテーブル（category_id: 主キー、category_name: 必須）
2. productsテーブル（product_id: 主キー、product_name: 必須、category_id: 外部キー） 外部キー制約には以下を設定：

- 親テーブル削除時：RESTRICT
- 親テーブル更新時：CASCADE

問題35-3：CHECK制約の活用

学生の個人情報テーブルstudent_profilesを作成してください：

- 学生ID：student_id（主キー）
- 生年月日：birth_date（現在日以前）
- 学年：grade（1-12の範囲）
- GPA：gpa（0.0-4.0の範囲）
- メールアドレス：email（基本的な形式チェック）
- 電話番号：phone（日本の形式：XXX-XXXX-XXXX）
- 郵便番号：postal_code（日本の形式：XXX-XXXX）

問題35-4：複合制約の設計

時間割管理テーブルclass_timetableを作成してください：

- 主キー：timetable_id（自動増加）
- 教室ID：classroom_id（必須）
- 曜日：day_of_week（1-7の範囲、1=月曜日）
- 時限：period（1-8の範囲）
- 講座ID：course_id（外部キー、coursesテーブル参照）
- 教師ID：teacher_id（外部キー、teachersテーブル参照）
- 学期：semester（'spring', 'summer', 'fall', 'winter'）
- 年度：academic_year（2000年以降） 制約：同じ教室・曜日・時限・学期・年度の組み合わせは一意

問題35-5：制約エラーの対処

以下のシナリオで適切なエラー対処を実装してください：

1. 主キー重複エラーが発生した場合のINSERT IGNORE使用
2. 外部キー制約違反時の事前チェックと条件付き挿入

3. CHECK制約違反時のデータ補正と挿入 具体的なSQL文を書いて、エラーが発生する例と対処法を示してください。

問題35-6：包括的な制約設計

オンライン学習システムの`course_enrollments`テーブルを設計してください：

- 登録ID：`enrollment_id`（主キー、自動増加）
- 学生ID：`student_id`（外部キー、CASCADE削除）
- 講座ID：`course_id`（外部キー、RESTRICT削除）
- 登録日：`enrollment_date`（デフォルト：現在日）
- 開始日：`start_date`（登録日以降）
- 修了日：`completion_date`（開始日以降、NULL許可）
- ステータス：`status`（'enrolled', 'in_progress', 'completed', 'dropped'、デフォルト'enrolled'）
- 進捗率：`progress_percentage`（0-100の範囲、デフォルト0）
- 評価：`rating`（1-5の範囲、NULL許可）
- 支払い状況：`payment_status`（'pending', 'paid', 'refunded'、デフォルト'pending'） すべての適切な制約を含めて設計してください。

解答

解答35-1

```
CREATE TABLE library_books (
    book_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    isbn CHAR(13) NOT NULL UNIQUE,
    title VARCHAR(200) NOT NULL,
    author VARCHAR(100) NOT NULL,
    publication_year INT,
    price DECIMAL(8,2),
    status ENUM('available', 'borrowed', 'maintenance') DEFAULT 'available',

    -- CHECK制約
    CHECK (publication_year >= 1900 AND publication_year <= YEAR(CURDATE())),
    CHECK (price >= 0)
);

-- テスト用データ挿入
INSERT INTO library_books (isbn, title, author, publication_year, price) VALUES
('9784123456789', 'データベース設計入門', '山田太郎', 2023, 2800.00),
('9784987654321', 'SQL実践ガイド', '佐藤花子', 2022, 3200.50);

-- 制約違反テスト（コメントアウト）
-- INSERT INTO library_books (isbn, title, author, publication_year, price) VALUES
-- ('9784111111111', 'テスト本', '著者名', 1800, 2000.00);  -- エラー: 出版年が1900年未
満
```

解答35-2

```
-- 親テーブル作成
CREATE TABLE categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(100) NOT NULL
);

-- 子テーブル作成（外部キー制約付き）
CREATE TABLE products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    product_name VARCHAR(200) NOT NULL,
    category_id INT NOT NULL,

    FOREIGN KEY (category_id) REFERENCES categories(category_id)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);

-- テストデータ挿入
INSERT INTO categories (category_name) VALUES ('電子機器'), ('書籍'), ('食品');
INSERT INTO products (product_name, category_id) VALUES
('ノートパソコン', 1),
('データベース教科書', 2),
('有機野菜セット', 3);

-- 外部キー制約のテスト
-- 親テーブル更新（CASCADE）
UPDATE categories SET category_id = 10 WHERE category_id = 1;
SELECT * FROM products WHERE category_id = 10;  -- 自動更新確認

-- 親テーブル削除試行（RESTRICT）
-- DELETE FROM categories WHERE category_id = 10;  -- エラー: 参照されているため削除不可
```

解答35-3

```
CREATE TABLE student_profiles (
    student_id BIGINT PRIMARY KEY,
    birth_date DATE,
    grade INT,
    gpa DECIMAL(3,2),
    email VARCHAR(255),
    phone VARCHAR(15),
    postal_code VARCHAR(8),

    -- CHECK制約
    CHECK (birth_date <= CURRENT_DATE),
    CHECK (grade >= 1 AND grade <= 12),
    CHECK (gpa >= 0.0 AND gpa <= 4.0),
    CHECK (email IS NULL OR email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-
zA-Z]{2,}$'),
    CHECK (phone IS NULL OR phone REGEXP '^[0-9]{3}-[0-9]{4}-[0-9]{4}$'),
```

```
    CHECK (postal_code IS NULL OR postal_code REGEXP '^[0-9]{3}-[0-9]{4}$')
);

-- 有効データの挿入
INSERT INTO student_profiles VALUES
(1001, '2005-04-15', 10, 3.75, 'student@example.com', '090-1234-5678', '100-
0001');

-- 制約違反テスト（コメントアウト）
-- INSERT INTO student_profiles VALUES
-- (1002, '2030-01-01', 10, 3.75, 'student@example.com', '090-1234-5678', '100-
0001');  -- エラー: 未来の生年月日
```

解答35-4

```
CREATE TABLE class_timetable (
    timetable_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    classroom_id VARCHAR(16) NOT NULL,
    day_of_week TINYINT NOT NULL,
    period TINYINT NOT NULL,
    course_id VARCHAR(16),
    teacher_id BIGINT,
    semester ENUM('spring', 'summer', 'fall', 'winter') NOT NULL,
    academic_year INT NOT NULL,

    -- CHECK制約
    CHECK (day_of_week >= 1 AND day_of_week <= 7),
    CHECK (period >= 1 AND period <= 8),
    CHECK (academic_year >= 2000),

    -- 外部キー制約
    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE SET NULL,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id) ON DELETE SET NULL,

    -- 複合UNIQUE制約
    UNIQUE KEY unique_classroom_schedule (classroom_id, day_of_week, period,
semester, academic_year)
);

-- テストデータ挿入
INSERT INTO class_timetable (classroom_id, day_of_week, period, course_id,
teacher_id, semester, academic_year)
VALUES ('101A', 1, 1, '1', 101, 'spring', 2025);

-- 重複エラーテスト（コメントアウト）
-- INSERT INTO class_timetable (classroom_id, day_of_week, period, course_id,
teacher_id, semester, academic_year)
-- VALUES ('101A', 1, 1, '2', 102, 'spring', 2025);  -- エラー: 同じ時間割の重複
```

解答35-5

```
-- テーブル作成
CREATE TABLE constraint_error_demo (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    score INT,
    foreign_ref INT,

    CHECK (score >= 0 AND score <= 100),
    FOREIGN KEY (foreign_ref) REFERENCES categories(category_id)
);

-- 1. 主キー重複エラーの対処
INSERT INTO constraint_error_demo VALUES (1, 'データ1', 85, 10);

-- 重複挿入（エラーになる）
-- INSERT INTO constraint_error_demo VALUES (1, 'データ2', 90, 10);  -- エラー

-- INSERT IGNOREで重複エラー回避
INSERT IGNORE INTO constraint_error_demo VALUES (1, 'データ2', 90, 10);  -- エラーにならない

-- ON DUPLICATE KEY UPDATEで更新
INSERT INTO constraint_error_demo VALUES (1, 'データ更新', 95, 10)
ON DUPLICATE KEY UPDATE name = VALUES(name), score = VALUES(score);

-- 2. 外部キー制約違反の対処
-- 存在チェック付き挿入
INSERT INTO constraint_error_demo (id, name, score, foreign_ref)
SELECT 2, 'データ3', 88, 2
WHERE EXISTS (SELECT 1 FROM categories WHERE category_id = 2);

-- 3. CHECK制約違反の対処
-- データ補正付き挿入
INSERT INTO constraint_error_demo (id, name, score, foreign_ref)
SELECT 3, 'データ4', LEAST(GREATEST(150, 0), 100), 2;  -- 150を100に補正

-- 結果確認
SELECT * FROM constraint_error_demo;
```

解答35-6

```
CREATE TABLE course_enrollments (
    enrollment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    start_date DATE,
    completion_date DATE,
    status ENUM('enrolled', 'in_progress', 'completed', 'dropped') DEFAULT
'enrolled',
```

```
    progress_percentage DECIMAL(5,2) DEFAULT 0.00,
    rating TINYINT,
    payment_status ENUM('pending', 'paid', 'refunded') DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    -- 外部キー制約
    FOREIGN KEY (student_id) REFERENCES students(student_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,

    -- CHECK制約
    CHECK (start_date >= enrollment_date),
    CHECK (completion_date IS NULL OR completion_date >= start_date),
    CHECK (progress_percentage >= 0.00 AND progress_percentage <= 100.00),
    CHECK (rating IS NULL OR (rating >= 1 AND rating <= 5)),

    -- 論理的制約
    CHECK (
        (status = 'completed' AND completion_date IS NOT NULL) OR
        (status != 'completed')
    ),
    CHECK (
        (status = 'completed' AND progress_percentage = 100.00) OR
        (status != 'completed')
    ),

    -- 複合UNIQUE制約（同じ学生が同じ講座を重複受講しない）
    UNIQUE KEY unique_student_course (student_id, course_id),

    -- インデックス
    INDEX idx_student_id (student_id),
    INDEX idx_course_id (course_id),
    INDEX idx_enrollment_date (enrollment_date),
    INDEX idx_status (status),
    INDEX idx_payment_status (payment_status)
);

-- テストデータ挿入
INSERT INTO course_enrollments (student_id, course_id, start_date,
progress_percentage)
VALUES (301, '1', '2025-05-25', 25.50);

-- 制約のテスト
-- 進捗率100%で未完了ステータス（論理矛盾、エラーになる）
-- INSERT INTO course_enrollments (student_id, course_id, start_date,
progress_percentage, status)
-- VALUES (302, '2', '2025-05-25', 100.00, 'in_progress');  -- エラー: 論理制約違反

-- 正しい完了データ
INSERT INTO course_enrollments (student_id, course_id, start_date,
completion_date, progress_percentage, status)
VALUES (302, '2', '2025-05-25', '2025-06-20', 100.00, 'completed');
```

```
-- 結果確認
SELECT * FROM course_enrollments;
```

まとめ

この章では、データベースの制約について詳しく学びました：

1. **制約の基本概念**：
    - データの整合性と品質保証の重要性
    - 制約違反時の自動エラー機能
    - 各制約の役割と特性
2. **主キー制約（PRIMARY KEY）**：
    - 一意性とNOT NULL保証
    - AUTO_INCREMENTとの組み合わせ
    - 複合主キーの活用
3. **外部キー制約（FOREIGN KEY）**：
    - 参照整合性の保証
    - カスケード動作の制御
    - テーブル間関係の明確化
4. **CHECK制約**：
    - 値の範囲と条件制限
    - 複雑な論理チェック
    - データ形式の検証
5. **UNIQUE制約**：
    - 重複防止機能
    - 単一・複合カラムでの一意性
    - 主キーとの使い分け
6. **NOT NULL・DEFAULT制約**：
    - 必須項目の保証
    - デフォルト値の自動設定
    - データ入力の簡略化
7. **実践的な制約設計**：
    - 包括的なテーブル設計
    - パフォーマンスとの両立
    - 段階的な制約実装
8. **エラー対処と管理**：

    - 制約違反エラーの解決方法
    - 事前チェックと条件付き操作
    - 制約情報の確認手順

9. **ベストプラクティス**：
    - 論理的で実用的な制約設計
    - パフォーマンスを考慮した実装
    - 保守性を重視した管理方法

制約は、データベースの信頼性を支える重要な機能です。適切に設計・実装することで、データの品質を保ち、アプリケーションの安定性を大幅に向上させることができます。

次の章では、「インデックス：パフォーマンス最適化」について学び、検索性能の向上方法を詳しく理解していきます。

---

36. シーケンス/AUTO_INCREMENT：連番生成

はじめに

前章では、制約を使ってデータの整合性を保つ方法を学習しました。この章では、データベースで一意な識別子を自動生成する「連番生成」について詳しく学習します。

MySQLでは、主に「AUTO_INCREMENT」機能を使用して連番を生成します。他のデータベース管理システム（PostgreSQL、Oracle、SQL Serverなど）で提供される「SEQUENCE」オブジェクトとは異なりますが、同様の機能を実現できます。

連番生成が必要となる場面の例：

- 「学生に一意の学生IDを自動で割り当てたい」
- 「注文番号、請求書番号などのビジネス文書に連番を付けたい」
- 「ログレコードに時系列の識別子を付けたい」
- 「複数のテーブルで共通の連番体系を使いたい」
- 「年度別、月別の連番を管理したい」
- 「欠番を避けて確実に連続した番号を生成したい」
- 「高負荷環境で競合状態を避けながら連番を生成したい」

この章では、AUTO_INCREMENTの基本的な使用方法から、複雑な連番管理システムの設計まで詳しく学んでいきます。

連番生成とは

連番生成は、データベースで一意な識別子を自動的に生成する仕組みです。主キーとして使用されることが多く、新しいレコードが挿入されるたびに自動的に次の番号が割り当てられます。

> **用語解説**：
>
> - **AUTO_INCREMENT**：MySQLでカラムに自動的に連番を割り当てる機能です。

> - **シーケンス（SEQUENCE）**：PostgreSQL、Oracle等で提供される連番生成オブジェクトです（MySQLにはありません）。
> - **連番（Sequential Number）**：1, 2, 3...のように連続した数値です。
> - **一意識別子（Unique Identifier）**：各レコードを一意に識別するための値です。
> - **開始値（Start Value）**：連番の開始値を指定します。
> - **増分値（Increment）**：連番の増加幅を指定します（通常は1）。
> - **最大値（Maximum Value）**：連番の上限値です。
> - **欠番（Gap）**：連番の途中で抜けている番号です。
> - **競合状態（Race Condition）**：複数の処理が同時に連番を取得しようとする状況です。
> - **ロック（Lock）**：競合を避けるためにリソースを一時的に占有することです。

AUTO_INCREMENTの基本

1. AUTO_INCREMENTの基本概念

AUTO_INCREMENTは、MySQLでテーブルのカラムに対して自動的に連番を割り当てる機能です。

AUTO_INCREMENTの特性

- **自動増加**：INSERT時に自動的に次の値が設定される
- **一意性**：重複しない値が保証される
- **整数型限定**：整数型のカラムにのみ設定可能
- **主キー推奨**：通常は主キーまたはUNIQUEキーに設定
- **テーブル単位**：1つのテーブルに1つのAUTO_INCREMENTカラムのみ

2. 基本的なAUTO_INCREMENTの使用

```
-- 基本的なAUTO_INCREMENTテーブル
CREATE TABLE basic_auto_increment (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- データ挿入（IDは自動設定）
INSERT INTO basic_auto_increment (name) VALUES
('データ1'),
('データ2'),
('データ3');

-- 結果確認
SELECT * FROM basic_auto_increment;

-- 次のAUTO_INCREMENT値を確認
SHOW TABLE STATUS LIKE 'basic_auto_increment';
```

3. AUTO_INCREMENTの詳細情報確認

```
-- AUTO_INCREMENT情報の取得
SELECT
    TABLE_NAME,
    AUTO_INCREMENT,
    TABLE_COMMENT
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = DATABASE()
AND TABLE_NAME = 'basic_auto_increment';

-- 最後に挿入されたAUTO_INCREMENT値を取得
SELECT LAST_INSERT_ID();

-- より詳細な情報
SHOW CREATE TABLE basic_auto_increment;
```

AUTO_INCREMENTの詳細設定

1. 開始値の設定

```
-- テーブル作成時に開始値を指定
CREATE TABLE custom_start_value (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(100)
) AUTO_INCREMENT = 1000;

-- データ挿入
INSERT INTO custom_start_value (data) VALUES ('データA'), ('データB');

-- 結果確認（1000, 1001から開始）
SELECT * FROM custom_start_value;

-- 既存テーブルの開始値変更
ALTER TABLE custom_start_value AUTO_INCREMENT = 2000;

-- 次の挿入は2000から
INSERT INTO custom_start_value (data) VALUES ('データC');
SELECT * FROM custom_start_value;
```

2. データ型別のAUTO_INCREMENT

```
-- 異なるデータ型でのAUTO_INCREMENT
CREATE TABLE different_types_demo (
    tiny_id TINYINT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50)
);

CREATE TABLE int_demo (
    int_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    data VARCHAR(50)
);

CREATE TABLE bigint_demo (
    bigint_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50)
);

-- TINYINTの上限テスト（127まで）
INSERT INTO different_types_demo (data)
SELECT CONCAT('データ', n)
FROM (
    SELECT 1 as n UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5
) numbers;

SELECT * FROM different_types_demo;

-- 上限値の確認
SELECT
    'TINYINT' as type,
    POWER(2, 7) - 1 as signed_max,
    POWER(2, 8) - 1 as unsigned_max
UNION ALL
SELECT 'INT', POWER(2, 31) - 1, POWER(2, 32) - 1
UNION ALL
SELECT 'BIGINT', POWER(2, 63) - 1, POWER(2, 64) - 1;
```

3. UNSIGNED型との組み合わせ

```
-- UNSIGNED型でのAUTO_INCREMENT
CREATE TABLE unsigned_demo (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    description VARCHAR(100)
);

-- より大きな範囲の連番が可能
INSERT INTO unsigned_demo (description) VALUES ('UNSIGNEDテスト');

-- 最大値の確認
SELECT
    COLUMN_TYPE,
    IS_NULLABLE,
    COLUMN_DEFAULT,
    EXTRA
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = DATABASE()
AND TABLE_NAME = 'unsigned_demo'
AND COLUMN_NAME = 'id';
```

カスタム連番生成

1. 手動連番管理テーブル

MySQLにはSEQUENCEオブジェクトがないため、手動で連番を管理するテーブルを作成できます。

```
-- 連番管理テーブル
CREATE TABLE sequence_generators (
    sequence_name VARCHAR(50) PRIMARY KEY,
    current_value BIGINT UNSIGNED NOT NULL DEFAULT 0,
    increment_by INT NOT NULL DEFAULT 1,
    min_value BIGINT UNSIGNED NOT NULL DEFAULT 1,
    max_value BIGINT UNSIGNED NOT NULL DEFAULT 9223372036854775807,
    cycle_flag BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 連番定義の初期化
INSERT INTO sequence_generators (sequence_name, current_value, increment_by)
VALUES
('student_id_seq', 1000, 1),
('order_number_seq', 100000, 1),
('invoice_seq', 202500001, 1);

-- 結果確認
SELECT * FROM sequence_generators;
```

2. 連番取得関数の作成

```
-- 連番取得のストアドファンクション
DELIMITER //

CREATE FUNCTION get_next_sequence(seq_name VARCHAR(50))
RETURNS BIGINT
READS SQL DATA
MODIFIES SQL DATA
DETERMINISTIC
BEGIN
    DECLARE next_val BIGINT;
    DECLARE max_val BIGINT;
    DECLARE increment_val INT;
    DECLARE cycle_enabled BOOLEAN;

    -- 現在値と設定を取得（排他制御）
    SELECT
        current_value + increment_by,
        max_value,
        increment_by,
        cycle_flag
    INTO next_val, max_val, increment_val, cycle_enabled
    FROM sequence_generators
```

```
    WHERE sequence_name = seq_name
    FOR UPDATE;

    -- 最大値チェック
    IF next_val > max_val THEN
        IF cycle_enabled THEN
            SET next_val = (SELECT min_value FROM sequence_generators WHERE
sequence_name = seq_name);
        ELSE
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Sequence maximum value
exceeded';
        END IF;
    END IF;

    -- 値を更新
    UPDATE sequence_generators
    SET current_value = next_val,
        updated_at = CURRENT_TIMESTAMP
    WHERE sequence_name = seq_name;

    RETURN next_val;
END //

DELIMITER ;

-- 関数の使用例
SELECT get_next_sequence('student_id_seq') as next_student_id;
SELECT get_next_sequence('order_number_seq') as next_order_number;
SELECT get_next_sequence('invoice_seq') as next_invoice_number;

-- 連番管理テーブルの確認
SELECT * FROM sequence_generators;
```

3. カスタム連番を使用したテーブル作成

```
-- カスタム連番を使用する学生テーブル
CREATE TABLE students_with_custom_seq (
    student_id BIGINT PRIMARY KEY,
    student_number VARCHAR(20) UNIQUE,
    student_name VARCHAR(100) NOT NULL,
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 学生登録のストアドプロシージャ
DELIMITER //

CREATE PROCEDURE register_student(
    IN p_student_name VARCHAR(100),
    IN p_enrollment_date DATE
)
```

```
BEGIN
    DECLARE new_student_id BIGINT;
    DECLARE new_student_number VARCHAR(20);

    -- カスタム連番を取得
    SET new_student_id = get_next_sequence('student_id_seq');
    SET new_student_number = CONCAT('S', YEAR(IFNULL(p_enrollment_date,
CURRENT_DATE)),
                                    LPAD(new_student_id, 6, '0'));

    -- 学生データを挿入
    INSERT INTO students_with_custom_seq (student_id, student_number,
student_name, enrollment_date)
    VALUES (new_student_id, new_student_number, p_student_name,
p_enrollment_date);

    -- 結果を返す
    SELECT new_student_id as student_id, new_student_number as student_number;
END //

DELIMITER ;

-- プロシージャの使用
CALL register_student('田中太郎', '2025-04-01');
CALL register_student('佐藤花子', '2025-04-01');
CALL register_student('鈴木次郎', '2025-04-02');

-- 結果確認
SELECT * FROM students_with_custom_seq;
```

年度別・月別連番

1. 年度別連番システム

```
-- 年度別連番管理
CREATE TABLE yearly_sequences (
    sequence_name VARCHAR(50),
    year_value INT,
    current_value BIGINT UNSIGNED DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    PRIMARY KEY (sequence_name, year_value)
);

-- 年度別連番取得関数
DELIMITER //

CREATE FUNCTION get_yearly_sequence(seq_name VARCHAR(50), target_year INT)
RETURNS BIGINT
READS SQL DATA
MODIFIES SQL DATA
```

```
DETERMINISTIC
BEGIN
    DECLARE next_val BIGINT DEFAULT 1;

    -- 該当年度の連番レコードが存在するかチェック
    IF NOT EXISTS (SELECT 1 FROM yearly_sequences
                   WHERE sequence_name = seq_name AND year_value = target_year)
THEN
        INSERT INTO yearly_sequences (sequence_name, year_value, current_value)
        VALUES (seq_name, target_year, 1);
        RETURN 1;
    END IF;

    -- 連番を更新して取得
    UPDATE yearly_sequences
    SET current_value = current_value + 1
    WHERE sequence_name = seq_name AND year_value = target_year;

    SELECT current_value INTO next_val
    FROM yearly_sequences
    WHERE sequence_name = seq_name AND year_value = target_year;

    RETURN next_val;
END //

DELIMITER ;

-- 年度別注文番号の生成例
SELECT
    CONCAT(YEAR(CURRENT_DATE), '-',
           LPAD(get_yearly_sequence('order_seq', YEAR(CURRENT_DATE)), 6, '0'))
    as order_number;

-- 複数回実行して確認
SELECT
    CONCAT(YEAR(CURRENT_DATE), '-',
           LPAD(get_yearly_sequence('order_seq', YEAR(CURRENT_DATE)), 6, '0'))
    as order_number;

-- 年度別連番の状況確認
SELECT * FROM yearly_sequences;
```

2. 月別連番システム

```
-- 月別連番管理
CREATE TABLE monthly_sequences (
    sequence_name VARCHAR(50),
    year_month VARCHAR(7),  -- YYYY-MM形式
    current_value BIGINT UNSIGNED DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    PRIMARY KEY (sequence_name, year_month)
);

-- 月別連番取得関数
DELIMITER //

CREATE FUNCTION get_monthly_sequence(seq_name VARCHAR(50), target_year_month
VARCHAR(7))
RETURNS BIGINT
READS SQL DATA
MODIFIES SQL DATA
DETERMINISTIC
BEGIN
    DECLARE next_val BIGINT DEFAULT 1;

    -- 該当月の連番レコードが存在するかチェック
    IF NOT EXISTS (SELECT 1 FROM monthly_sequences
                   WHERE sequence_name = seq_name AND year_month =
target_year_month) THEN
        INSERT INTO monthly_sequences (sequence_name, year_month, current_value)
        VALUES (seq_name, target_year_month, 1);
        RETURN 1;
    END IF;

    -- 連番を更新して取得
    UPDATE monthly_sequences
    SET current_value = current_value + 1
    WHERE sequence_name = seq_name AND year_month = target_year_month;

    SELECT current_value INTO next_val
    FROM monthly_sequences
    WHERE sequence_name = seq_name AND year_month = target_year_month;

    RETURN next_val;
END //

DELIMITER ;

-- 月別請求書番号の生成
SELECT
    CONCAT(DATE_FORMAT(CURRENT_DATE, '%Y%m'), '-',
           LPAD(get_monthly_sequence('invoice_seq', DATE_FORMAT(CURRENT_DATE, '%Y-
%m')), 4, '0'))
    as invoice_number;

-- 結果確認
SELECT * FROM monthly_sequences;
```

AUTO_INCREMENTの調整と管理

1. AUTO_INCREMENT値のリセット

```
-- テスト用テーブル作成
CREATE TABLE reset_test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50)
);

-- データ挿入
INSERT INTO reset_test (data) VALUES ('データ1'), ('データ2'), ('データ3');
SELECT * FROM reset_test;

-- AUTO_INCREMENT値をリセット
TRUNCATE TABLE reset_test;  -- データクリア + AUTO_INCREMENTリセット

-- または
-- DELETE FROM reset_test;
-- ALTER TABLE reset_test AUTO_INCREMENT = 1;

-- 新しいデータ挿入（1から開始）
INSERT INTO reset_test (data) VALUES ('新データ1');
SELECT * FROM reset_test;
```

2. AUTO_INCREMENT値の調整

```
-- 現在の値を確認
SELECT AUTO_INCREMENT FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = DATABASE() AND TABLE_NAME = 'reset_test';

-- 値を大きく設定
ALTER TABLE reset_test AUTO_INCREMENT = 1000;

-- 新しいデータ挿入
INSERT INTO reset_test (data) VALUES ('1000番台データ');
SELECT * FROM reset_test;

-- 現在の最大値より小さい値に設定しようとした場合
ALTER TABLE reset_test AUTO_INCREMENT = 500;  -- 効果なし（現在の最大値より小さいため）

-- 確認
SELECT AUTO_INCREMENT FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = DATABASE() AND TABLE_NAME = 'reset_test';
```

3. 欠番の確認と対処

```
-- 欠番確認用テーブル
CREATE TABLE gap_analysis (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

-- データ挿入
INSERT INTO gap_analysis (data) VALUES
('データ1'), ('データ2'), ('データ3'), ('データ4'), ('データ5');

-- 特定のレコードを削除（欠番を作成）
DELETE FROM gap_analysis WHERE id IN (2, 4);

-- 欠番の確認
SELECT * FROM gap_analysis ORDER BY id;

-- 欠番検出クエリ
SELECT
    id + 1 as gap_start,
    (SELECT MIN(id) - 1 FROM gap_analysis ga2 WHERE ga2.id > ga1.id) as gap_end
FROM gap_analysis ga1
WHERE NOT EXISTS (SELECT 1 FROM gap_analysis ga2 WHERE ga2.id = ga1.id + 1)
AND id < (SELECT MAX(id) FROM gap_analysis);

-- 連続性チェック
SELECT
    COUNT(*) as total_records,
    MAX(id) as max_id,
    MAX(id) - COUNT(*) as gaps_count
FROM gap_analysis;
```

高負荷環境での連番管理

1. 競合状態の回避

```

-- 安全な連番取得のためのテーブル
CREATE TABLE safe_counter (
    counter_name VARCHAR(50) PRIMARY KEY,
    current_value BIGINT UNSIGNED NOT NULL DEFAULT 0,
    lock_timeout INT DEFAULT 10,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 初期値設定
INSERT INTO safe_counter (counter_name, current_value) VALUES
('order_counter', 100000),
('ticket_counter', 1000000);

-- 安全な連番取得プロシージャ
DELIMITER //

CREATE PROCEDURE get_safe_sequence(
    IN counter_name VARCHAR(50),
    OUT next_value BIGINT
)
```

```
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    START TRANSACTION;

    -- 行レベルロックで排他制御
    SELECT current_value + 1 INTO next_value
    FROM safe_counter
    WHERE counter_name = counter_name
    FOR UPDATE;

    -- カウンタを更新
    UPDATE safe_counter
    SET current_value = next_value
    WHERE counter_name = counter_name;

    COMMIT;
END //

DELIMITER ;

-- 使用例
CALL get_safe_sequence('order_counter', @order_id);
SELECT @order_id as new_order_id;

CALL get_safe_sequence('ticket_counter', @ticket_id);
SELECT @ticket_id as new_ticket_id;

-- カウンタ状況確認
SELECT * FROM safe_counter;
```

2. パフォーマンス最適化

```
-- バッチ処理用の連番予約システム
CREATE TABLE sequence_reservations (
    sequence_name VARCHAR(50) PRIMARY KEY,
    reserved_start BIGINT UNSIGNED NOT NULL,
    reserved_end BIGINT UNSIGNED NOT NULL,
    current_position BIGINT UNSIGNED NOT NULL,
    batch_size INT NOT NULL DEFAULT 100,
    last_reservation TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);

-- バッチ連番取得プロシージャ
DELIMITER //
```

```
CREATE PROCEDURE reserve_sequence_batch(
    IN seq_name VARCHAR(50),
    IN batch_size INT,
    OUT start_value BIGINT,
    OUT end_value BIGINT
)
BEGIN
    DECLARE current_max BIGINT DEFAULT 0;

    -- 現在の最大値を取得
    SELECT IFNULL(MAX(reserved_end), 0) INTO current_max
    FROM sequence_reservations
    WHERE sequence_name = seq_name;

    -- 新しい範囲を計算
    SET start_value = current_max + 1;
    SET end_value = current_max + batch_size;

    -- 予約レコードを挿入または更新
    INSERT INTO sequence_reservations
        (sequence_name, reserved_start, reserved_end, current_position,
batch_size)
    VALUES
        (seq_name, start_value, end_value, start_value, batch_size)
    ON DUPLICATE KEY UPDATE
        reserved_start = start_value,
        reserved_end = end_value,
        current_position = start_value,
        batch_size = batch_size;
END //

DELIMITER ;

-- バッチ予約の使用例
CALL reserve_sequence_batch('bulk_order_seq', 1000, @start_val, @end_val);
SELECT @start_val as start_value, @end_val as end_value;

-- 予約状況確認
SELECT * FROM sequence_reservations;
```

実践的な連番設計例

1. 学校システムの包括的連番設計

```
-- 学校システム用連番管理
CREATE TABLE school_sequences (
    seq_type ENUM('student', 'teacher', 'course', 'grade_report',
'attendance_sheet') PRIMARY KEY,
    prefix VARCHAR(10) NOT NULL,
    current_number BIGINT UNSIGNED NOT NULL DEFAULT 0,
    year_reset BOOLEAN DEFAULT TRUE,
```

```
    last_reset_year INT,
    format_template VARCHAR(50) NOT NULL,
    description VARCHAR(200),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 連番設定の初期化
INSERT INTO school_sequences VALUES
('student', 'S', 0, TRUE, YEAR(CURRENT_DATE), '{PREFIX}{YEAR}{NUMBER:06d}', '学生
ID生成', NOW(), NOW()),
('teacher', 'T', 0, FALSE, NULL, '{PREFIX}{NUMBER:04d}', '教師ID生成', NOW(),
NOW()),
('course', 'C', 0, TRUE, YEAR(CURRENT_DATE), '{PREFIX}{YEAR}{NUMBER:03d}', '講座ID
生成', NOW(), NOW()),
('grade_report', 'GR', 0, TRUE, YEAR(CURRENT_DATE), '{PREFIX}{YEAR}{MONTH:02d}
{NUMBER:04d}', '成績表番号', NOW(), NOW()),
('attendance_sheet', 'AS', 0, TRUE, YEAR(CURRENT_DATE), '{PREFIX}{YEAR}{MONTH:02d}
{NUMBER:04d}', '出席表番号', NOW(), NOW());

-- 学校システム用連番生成関数
DELIMITER //

CREATE FUNCTION generate_school_id(seq_type_param ENUM('student', 'teacher',
'course', 'grade_report', 'attendance_sheet'))
RETURNS VARCHAR(50)
READS SQL DATA
MODIFIES SQL DATA
DETERMINISTIC
BEGIN
    DECLARE next_number BIGINT;
    DECLARE prefix_val VARCHAR(10);
    DECLARE format_template VARCHAR(50);
    DECLARE year_reset_flag BOOLEAN;
    DECLARE last_reset_year_val INT;
    DECLARE current_year INT DEFAULT YEAR(CURRENT_DATE);
    DECLARE current_month INT DEFAULT MONTH(CURRENT_DATE);
    DECLARE result_id VARCHAR(50);

    -- 設定を取得
    SELECT prefix, current_number, year_reset, last_reset_year, format_template
    INTO prefix_val, next_number, year_reset_flag, last_reset_year_val,
format_template
    FROM school_sequences
    WHERE seq_type = seq_type_param
    FOR UPDATE;

    -- 年度リセットチェック
    IF year_reset_flag AND (last_reset_year_val IS NULL OR last_reset_year_val <
current_year) THEN
        SET next_number = 1;
        UPDATE school_sequences
        SET current_number = 1, last_reset_year = current_year
        WHERE seq_type = seq_type_param;
```

```
    ELSE
        SET next_number = next_number + 1;
        UPDATE school_sequences
        SET current_number = next_number
        WHERE seq_type = seq_type_param;
    END IF;

    -- フォーマットに応じてIDを生成
    CASE format_template
        WHEN '{PREFIX}{YEAR}{NUMBER:06d}' THEN
            SET result_id = CONCAT(prefix_val, current_year, LPAD(next_number, 6,
'0'));
        WHEN '{PREFIX}{NUMBER:04d}' THEN
            SET result_id = CONCAT(prefix_val, LPAD(next_number, 4, '0'));
        WHEN '{PREFIX}{YEAR}{NUMBER:03d}' THEN
            SET result_id = CONCAT(prefix_val, current_year, LPAD(next_number, 3,
'0'));
        WHEN '{PREFIX}{YEAR}{MONTH:02d}{NUMBER:04d}' THEN
            SET result_id = CONCAT(prefix_val, current_year, LPAD(current_month,
2, '0'), LPAD(next_number, 4, '0'));
        ELSE
            SET result_id = CONCAT(prefix_val, next_number);
    END CASE;

    RETURN result_id;
END //

DELIMITER ;

-- 学校システムIDの生成テスト
SELECT
    generate_school_id('student') as student_id,
    generate_school_id('teacher') as teacher_id,
    generate_school_id('course') as course_id,
    generate_school_id('grade_report') as grade_report_id,
    generate_school_id('attendance_sheet') as attendance_sheet_id;

-- 連番状況確認
SELECT * FROM school_sequences;
```

2. 複数システム間での連番共有

```
-- システム間連番共有テーブル
CREATE TABLE global_sequences (
    system_name VARCHAR(50),
    sequence_name VARCHAR(50),
    current_value BIGINT UNSIGNED NOT NULL DEFAULT 0,
    increment_step INT NOT NULL DEFAULT 1,
    node_offset INT NOT NULL DEFAULT 0,  -- 分散システム用オフセット
    max_value BIGINT UNSIGNED,
```

```
    PRIMARY KEY (system_name, sequence_name),
    INDEX idx_sequence_name (sequence_name)
);

-- 分散システム用連番設定
INSERT INTO global_sequences VALUES
('school_system', 'global_student_id', 10000, 10, 1, 999999999),   -- ノード1:
10001, 10011, 10021...
('library_system', 'global_book_id', 20000, 10, 2, 999999999),    -- ノード2: 20002,
20012, 20022...
('exam_system', 'global_exam_id', 30000, 10, 3, 999999999);       -- ノード3: 30003,
30013, 30023...

-- 分散対応連番取得関数
DELIMITER //

CREATE FUNCTION get_distributed_sequence(
    system_name_param VARCHAR(50),
    sequence_name_param VARCHAR(50)
) RETURNS BIGINT
READS SQL DATA
MODIFIES SQL DATA
DETERMINISTIC
BEGIN
    DECLARE next_val BIGINT;
    DECLARE step_val INT;
    DECLARE offset_val INT;
    DECLARE max_val BIGINT;

    -- 設定値を取得
    SELECT current_value + increment_step, increment_step, node_offset, max_value
    INTO next_val, step_val, offset_val, max_val
    FROM global_sequences
    WHERE system_name = system_name_param AND sequence_name = sequence_name_param
    FOR UPDATE;

    -- 最大値チェック
    IF next_val > max_val THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Sequence maximum value
exceeded';
    END IF;

    -- 値を更新
    UPDATE global_sequences
    SET current_value = next_val
    WHERE system_name = system_name_param AND sequence_name = sequence_name_param;

    -- オフセットを適用した値を返す
    RETURN next_val + offset_val;
END //

DELIMITER ;

-- 分散連番の使用例
```

```
SELECT
    get_distributed_sequence('school_system', 'global_student_id') as
school_student_id,
    get_distributed_sequence('library_system', 'global_book_id') as
library_book_id,
    get_distributed_sequence('exam_system', 'global_exam_id') as exam_id;

-- 結果確認
SELECT * FROM global_sequences;
```

連番のバックアップと復旧

1. 連番状態のバックアップ

```
-- 連番バックアップテーブル
CREATE TABLE sequence_backups (
    backup_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    backup_name VARCHAR(100) NOT NULL,
    table_name VARCHAR(64) NOT NULL,
    sequence_name VARCHAR(50),
    sequence_value BIGINT NOT NULL,
    backup_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    backup_type ENUM('manual', 'scheduled', 'pre_maintenance') DEFAULT 'manual',
    notes TEXT
);

-- バックアップ作成プロシージャ
DELIMITER //

CREATE PROCEDURE backup_sequences(IN backup_name_param VARCHAR(100))
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tbl_name VARCHAR(64);
    DECLARE auto_inc_val BIGINT;

    DECLARE sequence_cursor CURSOR FOR
        SELECT TABLE_NAME, AUTO_INCREMENT
        FROM INFORMATION_SCHEMA.TABLES
        WHERE TABLE_SCHEMA = DATABASE()
        AND AUTO_INCREMENT IS NOT NULL;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN sequence_cursor;

    sequence_loop: LOOP
        FETCH sequence_cursor INTO tbl_name, auto_inc_val;
        IF done THEN
            LEAVE sequence_loop;
        END IF;
```

```
        INSERT INTO sequence_backups (backup_name, table_name, sequence_name,
sequence_value, backup_type)
        VALUES (backup_name_param, tbl_name, 'AUTO_INCREMENT', auto_inc_val,
'manual');
    END LOOP;

    CLOSE sequence_cursor;

    -- カスタム連番もバックアップ
    INSERT INTO sequence_backups (backup_name, table_name, sequence_name,
sequence_value, backup_type)
    SELECT backup_name_param, 'sequence_generators', sequence_name, current_value,
'manual'
    FROM sequence_generators;

END //

DELIMITER ;

-- バックアップ実行
CALL backup_sequences('daily_backup_20250522');

-- バックアップ確認
SELECT * FROM sequence_backups WHERE backup_name = 'daily_backup_20250522';
```

2. 連番の復旧

```
-- 復旧プロシージャ
DELIMITER //

CREATE PROCEDURE restore_sequences(IN backup_name_param VARCHAR(100))
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tbl_name VARCHAR(64);
    DECLARE seq_name VARCHAR(50);
    DECLARE seq_value BIGINT;

    DECLARE restore_cursor CURSOR FOR
        SELECT table_name, sequence_name, sequence_value
        FROM sequence_backups
        WHERE backup_name = backup_name_param;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN restore_cursor;

    restore_loop: LOOP
        FETCH restore_cursor INTO tbl_name, seq_name, seq_value;
        IF done THEN
            LEAVE restore_loop;
        END IF;
```

```
        -- AUTO_INCREMENTの復旧
        IF seq_name = 'AUTO_INCREMENT' THEN
            SET @sql = CONCAT('ALTER TABLE ', tbl_name, ' AUTO_INCREMENT = ',
seq_value);
            PREPARE stmt FROM @sql;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;
        END IF;

        -- カスタム連番の復旧
        IF tbl_name = 'sequence_generators' THEN
            UPDATE sequence_generators
            SET current_value = seq_value
            WHERE sequence_name = seq_name;
        END IF;

    END LOOP;

    CLOSE restore_cursor;
END //

DELIMITER ;

-- 復旧実行例（緊急時のみ）
-- CALL restore_sequences('daily_backup_20250522');
```

パフォーマンスと注意点

1. AUTO_INCREMENTのパフォーマンス特性

```
-- パフォーマンステスト用テーブル
CREATE TABLE performance_test_auto (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE performance_test_manual (
    id BIGINT PRIMARY KEY,
    data VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- AUTO_INCREMENTのパフォーマンステスト
-- 大量データ挿入の時間測定
SET @start_time = NOW(6);

INSERT INTO performance_test_auto (data)
SELECT CONCAT('test_data_', n)
FROM (
```

```
    SELECT a.N + b.N * 10 + c.N * 100 + 1 n
    FROM
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
a,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9)
b,
    (SELECT 0 AS N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4
     UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9) c
) numbers
LIMIT 1000;

SET @end_time = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @start_time, @end_time) as
auto_increment_microseconds;

-- レコード数確認
SELECT COUNT(*) as record_count FROM performance_test_auto;
```

2. 並行処理での注意点

```
-- 並行処理テスト用設定
SET SESSION innodb_autoinc_lock_mode = 1;  -- 従来のロックモード

-- 同時挿入のシミュレーション用プロシージャ
DELIMITER //

CREATE PROCEDURE concurrent_insert_test()
BEGIN
    DECLARE i INT DEFAULT 1;

    WHILE i <= 100 DO
        INSERT INTO performance_test_auto (data)
        VALUES (CONCAT('concurrent_', CONNECTION_ID(), '_', i));
        SET i = i + 1;
    END WHILE;
END //

DELIMITER ;

-- 同時実行テスト（複数のセッションで実行）
-- CALL concurrent_insert_test();

-- 結果の整合性確認
SELECT
    COUNT(*) as total_records,
    COUNT(DISTINCT id) as unique_ids,
    MAX(id) - MIN(id) + 1 as id_range
FROM performance_test_auto;
```

3. 設定の最適化

```
-- AUTO_INCREMENTに関する設定確認
SHOW VARIABLES LIKE '%auto_increment%';

-- 主要な設定項目の説明
SELECT
    'auto_increment_increment' as variable_name,
    'AUTO_INCREMENTの増分値' as description
UNION ALL
SELECT 'auto_increment_offset', 'AUTO_INCREMENTの開始オフセット'
UNION ALL
SELECT 'innodb_autoinc_lock_mode', 'AUTO_INCREMENTロックモード（0:従来, 1:連続, 2:インターリーブ）';

-- 現在の設定値
SELECT
    @@auto_increment_increment as increment_value,
    @@auto_increment_offset as offset_value,
    @@innodb_autoinc_lock_mode as lock_mode;
```

練習問題

問題36-1：基本的なAUTO_INCREMENT

以下の要件を満たすテーブル`products`を作成してください：

- 商品ID：`product_id`（AUTO_INCREMENT、主キー、1000から開始）
- 商品名：`product_name`（必須）
- 価格：`price`（小数点2桁）
- 作成日時：`created_at`（現在日時がデフォルト） テーブル作成後、3件のテストデータを挿入し、AUTO_INCREMENT値を確認してください。

問題36-2：カスタム連番システム

以下の機能を持つカスタム連番システムを実装してください：

1. 連番管理テーブル`custom_sequences`を作成
2. 連番取得関数`get_sequence(sequence_name)`を作成
3. 以下の連番を定義：
    - 'order_seq'：100000から開始、増分1
    - 'invoice_seq'：2025001から開始、増分1
4. 各連番を5回取得して動作確認

問題36-3：年度別連番

年度が変わると1から始まる年度別連番システムを実装してください：

1. 年度別連番管理テーブルを作成
2. 年度別連番取得関数を作成

3. 学籍番号形式「S{年度}{連番6桁}」で生成
4. 2024年度と2025年度で各3件ずつ生成してテスト

問題36-4：複合連番システム

以下の仕様で請求書番号生成システムを作成してください：

- 形式：「INV-{年}{月}{連番4桁}」（例：INV-20250501）
- 月が変わると連番は1からリセット
- 月別連番管理テーブルとプロシージャを実装
- 2025年5月と6月で各5件ずつ生成してテスト

問題36-5：欠番検出システム

以下の機能を実装してください：

1. テスト用テーブルを作成してAUTO_INCREMENTデータを10件挿入
2. ランダムに3件のレコードを削除して欠番を作成
3. 欠番を検出するクエリを作成
4. 欠番の範囲（開始番号、終了番号）を表示
5. 全体の欠番数を計算

問題36-6：高負荷対応連番システム

並行処理に対応した安全な連番システムを実装してください：

1. ロック機能付き連番テーブル`safe_sequences`を作成
2. 排他制御を行う連番取得プロシージャを実装
3. バッチ処理用の連番予約機能を実装
4. 競合状態をテストするプロシージャを作成
5. 同時実行時の整合性を確認

解答

解答36-1

```
-- productsテーブル作成
CREATE TABLE products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    product_name VARCHAR(200) NOT NULL,
    price DECIMAL(10,2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) AUTO_INCREMENT = 1000;

-- テストデータ挿入
INSERT INTO products (product_name, price) VALUES
('ノートパソコン', 89800.00),
('マウス', 2980.00),
('キーボード', 5980.00);

-- 結果確認
```

```
SELECT * FROM products;

-- AUTO_INCREMENT値確認
SELECT AUTO_INCREMENT FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = DATABASE() AND TABLE_NAME = 'products';

SHOW TABLE STATUS LIKE 'products';
```

解答36-2

```
-- 1. 連番管理テーブル作成
CREATE TABLE custom_sequences (
    sequence_name VARCHAR(50) PRIMARY KEY,
    current_value BIGINT UNSIGNED NOT NULL DEFAULT 0,
    increment_by INT NOT NULL DEFAULT 1,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 2. 連番取得関数作成
DELIMITER //
CREATE FUNCTION get_sequence(seq_name VARCHAR(50))
RETURNS BIGINT
READS SQL DATA
MODIFIES SQL DATA
DETERMINISTIC
BEGIN
    DECLARE next_val BIGINT;

    UPDATE custom_sequences
    SET current_value = current_value + increment_by,
        updated_at = CURRENT_TIMESTAMP
    WHERE sequence_name = seq_name;

    SELECT current_value INTO next_val
    FROM custom_sequences
    WHERE sequence_name = seq_name;

    RETURN next_val;
END //
DELIMITER ;

-- 3. 連番定義
INSERT INTO custom_sequences (sequence_name, current_value) VALUES
('order_seq', 99999),    -- 次回取得時に100000
('invoice_seq', 2025000); -- 次回取得時に2025001

-- 4. 動作確認
SELECT get_sequence('order_seq') as order_number;
SELECT get_sequence('order_seq') as order_number;
SELECT get_sequence('order_seq') as order_number;
```

```
SELECT get_sequence('order_seq') as order_number;
SELECT get_sequence('order_seq') as order_number;

SELECT get_sequence('invoice_seq') as invoice_number;
SELECT get_sequence('invoice_seq') as invoice_number;
SELECT get_sequence('invoice_seq') as invoice_number;
SELECT get_sequence('invoice_seq') as invoice_number;
SELECT get_sequence('invoice_seq') as invoice_number;

-- 連番状況確認
SELECT * FROM custom_sequences;
```

解答36-3

```
-- 1. 年度別連番管理テーブル
CREATE TABLE yearly_student_sequences (
    academic_year INT PRIMARY KEY,
    current_number INT UNSIGNED DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 2. 年度別連番取得関数
DELIMITER //
CREATE FUNCTION get_student_number(year_val INT)
RETURNS VARCHAR(20)
READS SQL DATA
MODIFIES SQL DATA
DETERMINISTIC
BEGIN
    DECLARE next_num INT;
    DECLARE student_number VARCHAR(20);

    -- 年度レコードが存在しない場合は作成
    INSERT IGNORE INTO yearly_student_sequences (academic_year, current_number)
    VALUES (year_val, 0);

    -- 連番を更新して取得
    UPDATE yearly_student_sequences
    SET current_number = current_number + 1,
        updated_at = CURRENT_TIMESTAMP
    WHERE academic_year = year_val;

    SELECT current_number INTO next_num
    FROM yearly_student_sequences
    WHERE academic_year = year_val;

    -- 学籍番号形式で返す
    SET student_number = CONCAT('S', year_val, LPAD(next_num, 6, '0'));

    RETURN student_number;
```

```
END //
DELIMITER ;

-- 3. テスト実行
-- 2024年度
SELECT get_student_number(2024) as student_number_2024;
SELECT get_student_number(2024) as student_number_2024;
SELECT get_student_number(2024) as student_number_2024;

-- 2025年度
SELECT get_student_number(2025) as student_number_2025;
SELECT get_student_number(2025) as student_number_2025;
SELECT get_student_number(2025) as student_number_2025;

-- 結果確認
SELECT * FROM yearly_student_sequences;
```

解答36-4

```
-- 1. 月別連番管理テーブル
CREATE TABLE monthly_invoice_sequences (
    year_month CHAR(6) PRIMARY KEY,  -- YYYYMM形式
    current_number INT UNSIGNED DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 2. 請求書番号生成プロシージャ
DELIMITER //
CREATE PROCEDURE generate_invoice_number(OUT invoice_number VARCHAR(20))
BEGIN
    DECLARE year_month_str CHAR(6);
    DECLARE next_num INT;

    SET year_month_str = DATE_FORMAT(CURRENT_DATE, '%Y%m');

    -- 月別レコードが存在しない場合は作成
    INSERT IGNORE INTO monthly_invoice_sequences (year_month, current_number)
    VALUES (year_month_str, 0);

    -- 連番を更新して取得
    UPDATE monthly_invoice_sequences
    SET current_number = current_number + 1,
        updated_at = CURRENT_TIMESTAMP
    WHERE year_month = year_month_str;

    SELECT current_number INTO next_num
    FROM monthly_invoice_sequences
    WHERE year_month = year_month_str;

    -- 請求書番号形式で生成
```

```
    SET invoice_number = CONCAT('INV-', year_month_str, LPAD(next_num, 4, '0'));
END //
DELIMITER ;

-- 3. テスト実行
-- 2025年5月のテスト
CALL generate_invoice_number(@inv1); SELECT @inv1 as invoice_may_1;
CALL generate_invoice_number(@inv2); SELECT @inv2 as invoice_may_2;
CALL generate_invoice_number(@inv3); SELECT @inv3 as invoice_may_3;
CALL generate_invoice_number(@inv4); SELECT @inv4 as invoice_may_4;
CALL generate_invoice_number(@inv5); SELECT @inv5 as invoice_may_5;

-- 手動で6月分をテスト（実際の月が変わった時のシミュレーション）
INSERT IGNORE INTO monthly_invoice_sequences (year_month, current_number) VALUES
('202506', 0);
UPDATE monthly_invoice_sequences SET current_number = 1 WHERE year_month =
'202506';
SELECT CONCAT('INV-', '202506', LPAD(1, 4, '0')) as invoice_june_1;
UPDATE monthly_invoice_sequences SET current_number = 2 WHERE year_month =
'202506';
SELECT CONCAT('INV-', '202506', LPAD(2, 4, '0')) as invoice_june_2;

-- 結果確認
SELECT * FROM monthly_invoice_sequences;
```

解答36-5

```
-- 1. テスト用テーブル作成とデータ挿入
CREATE TABLE gap_detection_test (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO gap_detection_test (data) VALUES
('データ1'), ('データ2'), ('データ3'), ('データ4'), ('データ5'),
('データ6'), ('データ7'), ('データ8'), ('データ9'), ('データ10');

-- 2. ランダムに3件削除（例：2, 5, 8を削除）
DELETE FROM gap_detection_test WHERE id IN (2, 5, 8);

-- 削除後の状態確認
SELECT * FROM gap_detection_test ORDER BY id;

-- 3. 欠番検出クエリ
-- 連続する欠番の範囲を検出
SELECT
    gap_start,
    CASE
        WHEN gap_end IS NULL THEN gap_start
        ELSE gap_end
```

```
    END as gap_end,
    CASE
        WHEN gap_end IS NULL THEN 1
        ELSE gap_end - gap_start + 1
    END as gap_count
FROM (
    SELECT
        id + 1 as gap_start,
        (SELECT MIN(id) - 1 FROM gap_detection_test g2 WHERE g2.id > g1.id) as
gap_end
    FROM gap_detection_test g1
    WHERE NOT EXISTS (SELECT 1 FROM gap_detection_test g2 WHERE g2.id = g1.id + 1)
    AND id < (SELECT MAX(id) FROM gap_detection_test)
) gaps;

-- 4. 個別の欠番リスト
SELECT missing_id
FROM (
    SELECT MIN(id) + (n.n - 1) as missing_id
    FROM gap_detection_test
    CROSS JOIN (
        SELECT 1 as n UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5
        UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9 UNION SELECT
10
    ) n
    WHERE MIN(id) + (n.n - 1) <= (SELECT MAX(id) FROM gap_detection_test)
) all_possible
WHERE missing_id NOT IN (SELECT id FROM gap_detection_test)
ORDER BY missing_id;

-- 5. 全体の欠番数計算
SELECT
    (SELECT MAX(id) FROM gap_detection_test) - (SELECT MIN(id) FROM
gap_detection_test) + 1 as expected_count,
    COUNT(*) as actual_count,
    (SELECT MAX(id) FROM gap_detection_test) - (SELECT MIN(id) FROM
gap_detection_test) + 1 - COUNT(*) as gap_count
FROM gap_detection_test;
```

解答36-6

```
-- 1. ロック機能付き連番テーブル
CREATE TABLE safe_sequences (
    sequence_name VARCHAR(50) PRIMARY KEY,
    current_value BIGINT UNSIGNED NOT NULL DEFAULT 0,
    batch_size INT DEFAULT 1,
    reserved_start BIGINT UNSIGNED DEFAULT 0,
    reserved_end BIGINT UNSIGNED DEFAULT 0,
    lock_holder VARCHAR(100),
    locked_at TIMESTAMP NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 2. 排他制御付き連番取得プロシージャ
DELIMITER //
CREATE PROCEDURE get_safe_sequence_value(
    IN seq_name VARCHAR(50),
    OUT next_value BIGINT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    START TRANSACTION;

    -- 排他ロックで連番を取得・更新
    SELECT current_value + 1 INTO next_value
    FROM safe_sequences
    WHERE sequence_name = seq_name
    FOR UPDATE;

    UPDATE safe_sequences
    SET current_value = next_value,
        updated_at = CURRENT_TIMESTAMP
    WHERE sequence_name = seq_name;

    COMMIT;
END //

-- 3. バッチ処理用連番予約プロシージャ
CREATE PROCEDURE reserve_sequence_batch_safe(
    IN seq_name VARCHAR(50),
    IN batch_size_param INT,
    OUT start_value BIGINT,
    OUT end_value BIGINT
)
BEGIN
    DECLARE current_val BIGINT;

    START TRANSACTION;

    SELECT current_value INTO current_val
    FROM safe_sequences
    WHERE sequence_name = seq_name
    FOR UPDATE;

    SET start_value = current_val + 1;
    SET end_value = current_val + batch_size_param;

    UPDATE safe_sequences
    SET current_value = end_value,
```

```
        reserved_start = start_value,
        reserved_end = end_value,
        batch_size = batch_size_param,
        updated_at = CURRENT_TIMESTAMP
    WHERE sequence_name = seq_name;

    COMMIT;
END //

-- 4. 競合状態テストプロシージャ
CREATE PROCEDURE concurrent_sequence_test(IN seq_name VARCHAR(50))
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE result BIGINT;

    WHILE i <= 10 DO
        CALL get_safe_sequence_value(seq_name, result);
        INSERT INTO sequence_test_log VALUES (CONNECTION_ID(), result, NOW());
        SET i = i + 1;
    END WHILE;
END //
DELIMITER ;

-- テスト用ログテーブル
CREATE TABLE sequence_test_log (
    connection_id BIGINT,
    sequence_value BIGINT,
    generated_at TIMESTAMP
);

-- 初期設定
INSERT INTO safe_sequences (sequence_name, current_value) VALUES
('test_seq', 0),
('batch_test_seq', 0);

-- 5. 整合性確認テスト
-- 単一アクセステスト
CALL get_safe_sequence_value('test_seq', @val1); SELECT @val1;
CALL get_safe_sequence_value('test_seq', @val2); SELECT @val2;
CALL get_safe_sequence_value('test_seq', @val3); SELECT @val3;

-- バッチ予約テスト
CALL reserve_sequence_batch_safe('batch_test_seq', 100, @start, @end);
SELECT @start as batch_start, @end as batch_end;

-- 競合テスト用（複数セッションで同時実行）
-- CALL concurrent_sequence_test('test_seq');

-- 結果確認
SELECT * FROM safe_sequences;
SELECT * FROM sequence_test_log ORDER BY generated_at;

-- 整合性チェック
SELECT
```

```
    COUNT(*) as total_generated,
    COUNT(DISTINCT sequence_value) as unique_values,
    MIN(sequence_value) as min_val,
    MAX(sequence_value) as max_val
FROM sequence_test_log;
```

まとめ

この章では、シーケンス/AUTO_INCREMENTによる連番生成について詳しく学びました：

1. **AUTO_INCREMENTの基本**：
   - MySQLでの自動連番生成機能
   - 基本的な使用方法と特性
   - データ型別の設定方法
2. **詳細設定と管理**：
   - 開始値のカスタマイズ
   - AUTO_INCREMENT値の調整
   - 欠番の検出と対処
3. **カスタム連番システム**：
   - 手動連番管理テーブルの設計
   - 連番取得関数の実装
   - 複雑な連番ルールの実現
4. **期間別連番管理**：
   - 年度別連番システム
   - 月別連番システム
   - 自動リセット機能
5. **高負荷環境への対応**：
   - 競合状態の回避
   - 排他制御の実装
   - バッチ処理最適化
6. **実践的な設計例**：
   - 学校システムでの包括的連番設計
   - 分散システム対応
   - 複数システム間での連番共有
7. **バックアップと復旧**：
   - 連番状態の保存
   - 緊急時の復旧手順
   - 整合性の確認

8. **パフォーマンス考慮点**：
    - 並行処理での注意点
    - 設定の最適化
    - ロックモードの理解

AUTO_INCREMENTは簡単に使用できる一方で、高負荷環境や複雑な要件では慎重な設計が必要です。適切に実装することで、信頼性の高い一意識別子システムを構築できます。

次の章では、「マテリアライズドビュー：結果を保存するビュー」について学び、パフォーマンス向上のための高度なビュー技術を理解していきます。

---

37. マテリアライズドビュー：結果を保存するビュー

はじめに

前章では、AUTO_INCREMENTによる連番生成について学習しました。この章では、クエリ結果を物理的に保存してパフォーマンスを向上させる「マテリアライズドビュー」について学習します。

重要な注意事項：MySQLには標準的なマテリアライズドビュー機能はありませんが、同様の効果を得るための手法を学ぶことで、大幅なパフォーマンス向上を実現できます。

マテリアライズドビューが有効な場面の例：

- 「複雑な集計クエリの実行時間を短縮したい」
- 「リアルタイムでのレポート生成が遅すぎる」
- 「複数のテーブルを結合した重い処理を高速化したい」
- 「ダッシュボードの表示速度を改善したい」
- 「定期的なレポート作成を効率化したい」
- 「大量データの分析クエリを最適化したい」
- 「読み取り専用の集計データを高速提供したい」

この章では、MySQLでマテリアライズドビューと同等の機能を実現する方法から、自動更新システムの構築、実践的な運用まで詳しく学んでいきます。

マテリアライズドビューとは

マテリアライズドビューは、ビューの実行結果を物理的なテーブルとして保存する仕組みです。通常のビューが実行時にクエリを動的に実行するのに対し、マテリアライズドビューは事前に計算された結果を保存しているため、非常に高速にデータを取得できます。

> **用語解説**：
>
> - **マテリアライズドビュー（Materialized View）**：クエリ結果を物理的に保存するビューです。
> - **通常のビュー（Regular View）**：実行時にクエリを動的に実行する仮想テーブルです。
> - **リフレッシュ（Refresh）**：マテリアライズドビューのデータを最新状態に更新することです。
> - **完全リフレッシュ（Complete Refresh）**：全データを再計算して更新する方法です。
> - **増分リフレッシュ（Incremental Refresh）**：変更された部分のみを更新する方法です。
> - **即座リフレッシュ（Immediate Refresh）**：元データの変更と同時に更新する方法です。

> - **遅延リフレッシュ（Deferred Refresh）**：定期的または手動でまとめて更新する方法です。
> - **集計テーブル（Summary Table）**：マテリアライズドビューの代替として使用される集計用テーブルです。
> - **ETLプロセス**：Extract, Transform, Loadの略で、データの抽出・変換・格納処理です。

MySQLでのマテリアライズドビュー実装

1. 基本的な実装パターン

MySQLではマテリアライズドビューを手動で実装します。

```
-- 元データの確認（学生の成績統計）
SELECT
    s.student_id,
    s.student_name,
    COUNT(g.grade_id) as total_grades,
    ROUND(AVG(g.score), 2) as average_score,
    MAX(g.score) as highest_score,
    MIN(g.score) as lowest_score
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id
GROUP BY s.student_id, s.student_name
LIMIT 5;

-- マテリアライズドビュー相当のテーブルを作成
CREATE TABLE mv_student_grade_summary (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100),
    total_grades INT DEFAULT 0,
    average_score DECIMAL(5,2),
    highest_score DECIMAL(5,2),
    lowest_score DECIMAL(5,2),
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_average_score (average_score),
    INDEX idx_student_name (student_name)
);

-- 初期データの投入
INSERT INTO mv_student_grade_summary
    (student_id, student_name, total_grades, average_score, highest_score,
lowest_score)
SELECT
    s.student_id,
    s.student_name,
    COUNT(g.grade_id) as total_grades,
    ROUND(AVG(g.score), 2) as average_score,
    MAX(g.score) as highest_score,
    MIN(g.score) as lowest_score
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id
GROUP BY s.student_id, s.student_name;
```

```
-- 結果確認
SELECT * FROM mv_student_grade_summary ORDER BY average_score DESC LIMIT 10;
```

2. 更新プロシージャの作成

```
-- 完全リフレッシュ用プロシージャ
DELIMITER //

CREATE PROCEDURE refresh_student_grade_summary()
BEGIN
    -- 既存データをクリア
    TRUNCATE TABLE mv_student_grade_summary;

    -- 最新データで再構築
    INSERT INTO mv_student_grade_summary
        (student_id, student_name, total_grades, average_score, highest_score,
lowest_score)
    SELECT
        s.student_id,
        s.student_name,
        COUNT(g.grade_id) as total_grades,
        ROUND(AVG(g.score), 2) as average_score,
        MAX(g.score) as highest_score,
        MIN(g.score) as lowest_score
    FROM students s
    LEFT JOIN grades g ON s.student_id = g.student_id
    GROUP BY s.student_id, s.student_name;

    -- 更新ログ
    INSERT INTO mv_refresh_log (mv_name, refresh_type, refresh_time, record_count)
    SELECT 'mv_student_grade_summary', 'COMPLETE', NOW(), COUNT(*)
    FROM mv_student_grade_summary;
END //

DELIMITER ;

-- 更新ログテーブル
CREATE TABLE mv_refresh_log (
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    mv_name VARCHAR(100) NOT NULL,
    refresh_type ENUM('COMPLETE', 'INCREMENTAL', 'MANUAL') NOT NULL,
    refresh_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    record_count INT,
    execution_time_ms INT,
    notes TEXT
);

-- プロシージャの実行
CALL refresh_student_grade_summary();
```

```
-- ログ確認
SELECT * FROM mv_refresh_log;
```

増分更新システム

1. 変更追跡テーブル

```
-- 変更追跡用テーブル
CREATE TABLE mv_change_log (
    change_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    table_name VARCHAR(64) NOT NULL,
    operation_type ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,
    record_id BIGINT NOT NULL,
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed BOOLEAN DEFAULT FALSE,

    INDEX idx_table_processed (table_name, processed),
    INDEX idx_changed_at (changed_at)
);

-- 成績データの増分更新プロシージャ
DELIMITER //

CREATE PROCEDURE incremental_refresh_student_grades()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE affected_student_id BIGINT;
    DECLARE operation_type VARCHAR(10);

    -- 未処理の変更を取得するカーソル
    DECLARE change_cursor CURSOR FOR
        SELECT DISTINCT record_id, operation_type
        FROM mv_change_log
        WHERE table_name = 'grades' AND processed = FALSE;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN change_cursor;

    change_loop: LOOP
        FETCH change_cursor INTO affected_student_id, operation_type;
        IF done THEN
            LEAVE change_loop;
        END IF;

        -- 該当学生の統計を再計算
        INSERT INTO mv_student_grade_summary
            (student_id, student_name, total_grades, average_score, highest_score,
lowest_score)
        SELECT
            s.student_id,
```

```
            s.student_name,
            COUNT(g.grade_id) as total_grades,
            ROUND(AVG(g.score), 2) as average_score,
            MAX(g.score) as highest_score,
            MIN(g.score) as lowest_score
        FROM students s
        LEFT JOIN grades g ON s.student_id = g.student_id
        WHERE s.student_id = affected_student_id
        GROUP BY s.student_id, s.student_name
        ON DUPLICATE KEY UPDATE
            total_grades = VALUES(total_grades),
            average_score = VALUES(average_score),
            highest_score = VALUES(highest_score),
            lowest_score = VALUES(lowest_score),
            last_updated = CURRENT_TIMESTAMP;

    END LOOP;

    CLOSE change_cursor;

    -- 処理済みマークを設定
    UPDATE mv_change_log
    SET processed = TRUE
    WHERE table_name = 'grades' AND processed = FALSE;

    -- ログ記録
    INSERT INTO mv_refresh_log (mv_name, refresh_type, record_count)
    SELECT 'mv_student_grade_summary', 'INCREMENTAL', COUNT(DISTINCT record_id)
    FROM mv_change_log
    WHERE table_name = 'grades' AND processed = TRUE;

END //

DELIMITER ;
```

2. トリガーによる自動変更追跡

```
-- 成績テーブルの変更を自動追跡するトリガー
DELIMITER //

CREATE TRIGGER tr_grades_insert_mv
AFTER INSERT ON grades
FOR EACH ROW
BEGIN
    INSERT INTO mv_change_log (table_name, operation_type, record_id)
    VALUES ('grades', 'INSERT', NEW.student_id);
END //

CREATE TRIGGER tr_grades_update_mv
AFTER UPDATE ON grades
FOR EACH ROW
```

```
BEGIN
    INSERT INTO mv_change_log (table_name, operation_type, record_id)
    VALUES ('grades', 'UPDATE', NEW.student_id);

    -- 学生IDが変更された場合は旧IDも追跡
    IF OLD.student_id != NEW.student_id THEN
        INSERT INTO mv_change_log (table_name, operation_type, record_id)
        VALUES ('grades', 'UPDATE', OLD.student_id);
    END IF;
END //

CREATE TRIGGER tr_grades_delete_mv
AFTER DELETE ON grades
FOR EACH ROW
BEGIN
    INSERT INTO mv_change_log (table_name, operation_type, record_id)
    VALUES ('grades', 'DELETE', OLD.student_id);
END //

DELIMITER ;

-- トリガーのテスト
INSERT INTO grades (student_id, course_id, grade_type, score, max_score)
VALUES (301, '1', 'テストトリガー', 88.5, 100);

-- 変更ログの確認
SELECT * FROM mv_change_log WHERE table_name = 'grades' ORDER BY change_id DESC
LIMIT 5;

-- 増分更新の実行
CALL incremental_refresh_student_grades();

-- 更新結果の確認
SELECT * FROM mv_student_grade_summary WHERE student_id = 301;
```

複雑なマテリアライズドビューの例

1. 月別出席統計ビュー

```
-- 月別出席統計のマテリアライズドビュー
CREATE TABLE mv_monthly_attendance_stats (
    year_month VARCHAR(7) PRIMARY KEY,  -- YYYY-MM
    total_classes INT DEFAULT 0,
    total_students INT DEFAULT 0,
    present_count INT DEFAULT 0,
    late_count INT DEFAULT 0,
    absent_count INT DEFAULT 0,
    attendance_rate DECIMAL(5,2) DEFAULT 0.00,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_attendance_rate (attendance_rate),
```

```
    INDEX idx_year_month (year_month)
);

-- 月別出席統計更新プロシージャ
DELIMITER //

CREATE PROCEDURE refresh_monthly_attendance_stats(IN target_year_month VARCHAR(7))
BEGIN
    DECLARE total_classes_count INT DEFAULT 0;
    DECLARE total_students_count INT DEFAULT 0;
    DECLARE present_count_val INT DEFAULT 0;
    DECLARE late_count_val INT DEFAULT 0;
    DECLARE absent_count_val INT DEFAULT 0;
    DECLARE attendance_rate_val DECIMAL(5,2) DEFAULT 0.00;

    -- 統計を計算
    SELECT
        COUNT(DISTINCT cs.schedule_id),
        COUNT(DISTINCT a.student_id),
        SUM(CASE WHEN a.status = 'present' THEN 1 ELSE 0 END),
        SUM(CASE WHEN a.status = 'late' THEN 1 ELSE 0 END),
        SUM(CASE WHEN a.status = 'absent' THEN 1 ELSE 0 END)
    INTO
        total_classes_count,
        total_students_count,
        present_count_val,
        late_count_val,
        absent_count_val
    FROM course_schedule cs
    LEFT JOIN attendance a ON cs.schedule_id = a.schedule_id
    WHERE DATE_FORMAT(cs.schedule_date, '%Y-%m') = target_year_month;

    -- 出席率を計算
    IF (present_count_val + late_count_val + absent_count_val) > 0 THEN
        SET attendance_rate_val = ROUND((present_count_val * 100.0) /
(present_count_val + late_count_val + absent_count_val), 2);
    END IF;

    -- マテリアライズドビューを更新
    INSERT INTO mv_monthly_attendance_stats
        (year_month, total_classes, total_students, present_count, late_count,
absent_count, attendance_rate)
    VALUES
        (target_year_month, total_classes_count, total_students_count,
present_count_val, late_count_val, absent_count_val, attendance_rate_val)
    ON DUPLICATE KEY UPDATE
        total_classes = VALUES(total_classes),
        total_students = VALUES(total_students),
        present_count = VALUES(present_count),
        late_count = VALUES(late_count),
        absent_count = VALUES(absent_count),
        attendance_rate = VALUES(attendance_rate),
        last_updated = CURRENT_TIMESTAMP;
```

```
    -- ログ記録
    INSERT INTO mv_refresh_log (mv_name, refresh_type, notes)
    VALUES ('mv_monthly_attendance_stats', 'MANUAL', CONCAT('Updated: ',
target_year_month));

END //

DELIMITER ;

-- 月別統計の更新
CALL refresh_monthly_attendance_stats('2025-05');

-- 結果確認
SELECT * FROM mv_monthly_attendance_stats;
```

2. 講座別パフォーマンス分析ビュー

```
-- 講座別パフォーマンス分析のマテリアライズドビュー
CREATE TABLE mv_course_performance_analysis (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128),
    teacher_name VARCHAR(64),
    enrolled_students INT DEFAULT 0,
    total_grades INT DEFAULT 0,
    average_score DECIMAL(5,2) DEFAULT 0.00,
    pass_rate DECIMAL(5,2) DEFAULT 0.00,  -- 60点以上の割合
    excellent_rate DECIMAL(5,2) DEFAULT 0.00,  -- 90点以上の割合
    total_classes INT DEFAULT 0,
    average_attendance_rate DECIMAL(5,2) DEFAULT 0.00,
    course_rating ENUM('Excellent', 'Good', 'Average', 'Below Average', 'Poor')
DEFAULT 'Average',
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_average_score (average_score),
    INDEX idx_pass_rate (pass_rate),
    INDEX idx_course_rating (course_rating)
);

-- 講座パフォーマンス分析更新プロシージャ
DELIMITER //

CREATE PROCEDURE refresh_course_performance_analysis()
BEGIN
    TRUNCATE TABLE mv_course_performance_analysis;

    INSERT INTO mv_course_performance_analysis
        (course_id, course_name, teacher_name, enrolled_students, total_grades,
average_score,
         pass_rate, excellent_rate, total_classes, average_attendance_rate,
course_rating)
    SELECT
```

```
        c.course_id,
        c.course_name,
        t.teacher_name,
        -- 受講者数
        (SELECT COUNT(*) FROM student_courses sc WHERE sc.course_id = c.course_id)
as enrolled_students,
        -- 成績統計
        COUNT(g.grade_id) as total_grades,
        ROUND(AVG(g.score), 2) as average_score,
        ROUND(AVG(CASE WHEN g.score >= 60 THEN 100.0 ELSE 0 END), 2) as pass_rate,
        ROUND(AVG(CASE WHEN g.score >= 90 THEN 100.0 ELSE 0 END), 2) as
excellent_rate,
        -- 出席統計
        (SELECT COUNT(DISTINCT cs.schedule_id)
         FROM course_schedule cs
         WHERE cs.course_id = c.course_id) as total_classes,
        (SELECT ROUND(AVG(CASE WHEN a.status = 'present' THEN 100.0 ELSE 0 END),
2)
         FROM course_schedule cs2
         LEFT JOIN attendance a ON cs2.schedule_id = a.schedule_id
         WHERE cs2.course_id = c.course_id) as average_attendance_rate,
        -- 総合評価
        CASE
            WHEN AVG(g.score) >= 85 AND AVG(CASE WHEN g.score >= 60 THEN 100.0
ELSE 0 END) >= 90 THEN 'Excellent'
            WHEN AVG(g.score) >= 75 AND AVG(CASE WHEN g.score >= 60 THEN 100.0
ELSE 0 END) >= 80 THEN 'Good'
            WHEN AVG(g.score) >= 65 AND AVG(CASE WHEN g.score >= 60 THEN 100.0
ELSE 0 END) >= 70 THEN 'Average'
            WHEN AVG(g.score) >= 55 THEN 'Below Average'
            ELSE 'Poor'
        END as course_rating
    FROM courses c
    JOIN teachers t ON c.teacher_id = t.teacher_id
    LEFT JOIN grades g ON c.course_id = g.course_id
    GROUP BY c.course_id, c.course_name, t.teacher_name;

    -- ログ記録
    INSERT INTO mv_refresh_log (mv_name, refresh_type, record_count)
    SELECT 'mv_course_performance_analysis', 'COMPLETE', COUNT(*)
    FROM mv_course_performance_analysis;

END //

DELIMITER ;

-- 講座パフォーマンス分析の更新
CALL refresh_course_performance_analysis();

-- 結果確認
SELECT
    course_name,
    teacher_name,
    enrolled_students,
```

```
    average_score,
    pass_rate,
    average_attendance_rate,
    course_rating
FROM mv_course_performance_analysis
ORDER BY course_rating, average_score DESC;
```

自動更新スケジューリング

1. 更新スケジュール管理

```
-- マテリアライズドビュー更新スケジュール管理
CREATE TABLE mv_refresh_schedule (
    schedule_id INT AUTO_INCREMENT PRIMARY KEY,
    mv_name VARCHAR(100) NOT NULL,
    refresh_type ENUM('COMPLETE', 'INCREMENTAL') NOT NULL,
    refresh_frequency ENUM('HOURLY', 'DAILY', 'WEEKLY', 'MONTHLY', 'MANUAL') NOT
NULL,
    refresh_time TIME,  -- 実行時刻
    last_refresh TIMESTAMP NULL,
    next_refresh TIMESTAMP NULL,
    is_active BOOLEAN DEFAULT TRUE,
    max_execution_time INT DEFAULT 3600,  -- 最大実行時間（秒）

    INDEX idx_next_refresh (next_refresh),
    INDEX idx_active (is_active)
);

-- スケジュール設定
INSERT INTO mv_refresh_schedule
    (mv_name, refresh_type, refresh_frequency, refresh_time, is_active)
VALUES
    ('mv_student_grade_summary', 'INCREMENTAL', 'HOURLY', '00:00:00', TRUE),
    ('mv_monthly_attendance_stats', 'COMPLETE', 'DAILY', '02:00:00', TRUE),
    ('mv_course_performance_analysis', 'COMPLETE', 'WEEKLY', '03:00:00', TRUE);

-- 次回実行時刻の計算と更新
UPDATE mv_refresh_schedule
SET next_refresh = CASE
    WHEN refresh_frequency = 'HOURLY' THEN
        DATE_ADD(CONCAT(CURDATE(), ' ', refresh_time), INTERVAL 1 HOUR)
    WHEN refresh_frequency = 'DAILY' THEN
        DATE_ADD(CONCAT(CURDATE(), ' ', refresh_time), INTERVAL 1 DAY)
    WHEN refresh_frequency = 'WEEKLY' THEN
        DATE_ADD(CONCAT(CURDATE(), ' ', refresh_time), INTERVAL 1 WEEK)
    WHEN refresh_frequency = 'MONTHLY' THEN
        DATE_ADD(CONCAT(CURDATE(), ' ', refresh_time), INTERVAL 1 MONTH)
END
WHERE is_active = TRUE;
```

```
-- スケジュール確認
SELECT * FROM mv_refresh_schedule;
```

2. 自動実行プロシージャ

```
-- 自動更新実行プロシージャ
DELIMITER //

CREATE PROCEDURE execute_scheduled_refreshes()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE mv_name_var VARCHAR(100);
    DECLARE refresh_type_var VARCHAR(20);
    DECLARE refresh_freq_var VARCHAR(20);
    DECLARE start_time TIMESTAMP;
    DECLARE execution_time INT;

    DECLARE schedule_cursor CURSOR FOR
        SELECT mv_name, refresh_type, refresh_frequency
        FROM mv_refresh_schedule
        WHERE is_active = TRUE
        AND next_refresh <= NOW();

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN schedule_cursor;

    refresh_loop: LOOP
        FETCH schedule_cursor INTO mv_name_var, refresh_type_var,
refresh_freq_var;
        IF done THEN
            LEAVE refresh_loop;
        END IF;

        SET start_time = NOW();

        -- マテリアライズドビューに応じた更新実行
        CASE mv_name_var
            WHEN 'mv_student_grade_summary' THEN
                IF refresh_type_var = 'COMPLETE' THEN
                    CALL refresh_student_grade_summary();
                ELSE
                    CALL incremental_refresh_student_grades();
                END IF;

            WHEN 'mv_monthly_attendance_stats' THEN
                CALL refresh_monthly_attendance_stats(DATE_FORMAT(NOW(), '%Y-
%m'));

            WHEN 'mv_course_performance_analysis' THEN
                CALL refresh_course_performance_analysis();
```

```
        END CASE;

        SET execution_time = TIMESTAMPDIFF(SECOND, start_time, NOW());

        -- スケジュール更新
        UPDATE mv_refresh_schedule
        SET
            last_refresh = start_time,
            next_refresh = CASE
                WHEN refresh_freq_var = 'HOURLY' THEN DATE_ADD(start_time,
INTERVAL 1 HOUR)
                WHEN refresh_freq_var = 'DAILY' THEN DATE_ADD(start_time, INTERVAL
1 DAY)
                WHEN refresh_freq_var = 'WEEKLY' THEN DATE_ADD(start_time,
INTERVAL 1 WEEK)
                WHEN refresh_freq_var = 'MONTHLY' THEN DATE_ADD(start_time,
INTERVAL 1 MONTH)
            END
        WHERE mv_name = mv_name_var;

        -- 実行ログ更新
        UPDATE mv_refresh_log
        SET execution_time_ms = execution_time * 1000
        WHERE mv_name = mv_name_var
        AND refresh_time = start_time;

    END LOOP;

    CLOSE schedule_cursor;
END //

DELIMITER ;

-- 手動実行テスト
CALL execute_scheduled_refreshes();

-- 実行結果確認
SELECT * FROM mv_refresh_log ORDER BY refresh_time DESC LIMIT 10;
SELECT * FROM mv_refresh_schedule;
```

パフォーマンス比較と最適化

1. パフォーマンステスト

```
-- 通常のビューでのクエリ実行時間測定
CREATE VIEW v_student_performance AS
SELECT
    s.student_id,
    s.student_name,
    COUNT(g.grade_id) as total_grades,
    ROUND(AVG(g.score), 2) as average_score,
```

```
    MAX(g.score) as highest_score,
    MIN(g.score) as lowest_score,
    ROUND(AVG(CASE WHEN a.status = 'present' THEN 100.0 ELSE 0 END), 2) as
attendance_rate
FROM students s
LEFT JOIN grades g ON s.student_id = g.student_id
LEFT JOIN attendance a ON s.student_id = a.student_id
GROUP BY s.student_id, s.student_name;

-- 通常ビューでの実行時間測定
SET @start_time = NOW(6);
SELECT * FROM v_student_performance WHERE average_score >= 80 ORDER BY
average_score DESC;
SET @view_time = TIMESTAMPDIFF(MICROSECOND, @start_time, NOW(6));

-- マテリアライズドビューでの実行時間測定
SET @start_time = NOW(6);
SELECT * FROM mv_student_grade_summary WHERE average_score >= 80 ORDER BY
average_score DESC;
SET @mv_time = TIMESTAMPDIFF(MICROSECOND, @start_time, NOW(6));

-- パフォーマンス比較
SELECT
    @view_time as regular_view_microseconds,
    @mv_time as materialized_view_microseconds,
    ROUND(@view_time / @mv_time, 2) as performance_improvement_ratio;
```

2. ストレージ使用量の確認

```
-- テーブルサイズの比較
SELECT
    table_name,
    table_rows,
    ROUND(data_length / 1024 / 1024, 2) as data_mb,
    ROUND(index_length / 1024 / 1024, 2) as index_mb,
    ROUND((data_length + index_length) / 1024 / 1024, 2) as total_mb
FROM information_schema.tables
WHERE table_schema = DATABASE()
AND table_name IN ('students', 'grades', 'attendance', 'mv_student_grade_summary',
'mv_course_performance_analysis')
ORDER BY (data_length + index_length) DESC;
```

3. 最適化テクニック

```
-- パーティション化されたマテリアライズドビュー（年月別）
CREATE TABLE mv_monthly_grade_summary (
    year_month VARCHAR(7),
    student_id BIGINT,
    total_grades INT DEFAULT 0,
```

```
    average_score DECIMAL(5,2),
    grade_distribution JSON,  -- {'A': 5, 'B': 3, 'C': 2, 'D': 1, 'F': 0}
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    PRIMARY KEY (year_month, student_id),
    INDEX idx_average_score (average_score),
    INDEX idx_year_month (year_month)
);

-- 月別成績サマリー更新プロシージャ
DELIMITER //

CREATE PROCEDURE refresh_monthly_grade_summary(IN target_year_month VARCHAR(7))
BEGIN
    -- 該当月のデータを削除
    DELETE FROM mv_monthly_grade_summary WHERE year_month = target_year_month;

    -- 新しいデータを挿入
    INSERT INTO mv_monthly_grade_summary
        (year_month, student_id, total_grades, average_score, grade_distribution)
    SELECT
        target_year_month,
        s.student_id,
        COUNT(g.grade_id) as total_grades,
        ROUND(AVG(g.score), 2) as average_score,
        JSON_OBJECT(
            'A', SUM(CASE WHEN g.score >= 90 THEN 1 ELSE 0 END),
            'B', SUM(CASE WHEN g.score >= 80 AND g.score < 90 THEN 1 ELSE 0 END),
            'C', SUM(CASE WHEN g.score >= 70 AND g.score < 80 THEN 1 ELSE 0 END),
            'D', SUM(CASE WHEN g.score >= 60 AND g.score < 70 THEN 1 ELSE 0 END),
            'F', SUM(CASE WHEN g.score < 60 THEN 1 ELSE 0 END)
        ) as grade_distribution
    FROM students s
    LEFT JOIN grades g ON s.student_id = g.student_id
        AND DATE_FORMAT(g.submission_date, '%Y-%m') = target_year_month
    GROUP BY s.student_id
    HAVING COUNT(g.grade_id) > 0;  -- 成績がある学生のみ

END //

DELIMITER ;

-- 月別サマリーの更新
CALL refresh_monthly_grade_summary('2025-05');

-- JSON データの活用例
SELECT
    student_id,
    average_score,
    JSON_EXTRACT(grade_distribution, '$.A') as grade_A_count,
    JSON_EXTRACT(grade_distribution, '$.B') as grade_B_count,
    JSON_EXTRACT(grade_distribution, '$.C') as grade_C_count
FROM mv_monthly_grade_summary
WHERE year_month = '2025-05'
```

```
ORDER BY average_score DESC
LIMIT 10;
```

実践的な運用例

1. ダッシュボード用マテリアライズドビュー

```
-- 管理者ダッシュボード用統合ビュー
CREATE TABLE mv_admin_dashboard (
    metric_name VARCHAR(50) PRIMARY KEY,
    metric_value DECIMAL(15,2),
    metric_unit VARCHAR(20),
    previous_value DECIMAL(15,2),
    change_percentage DECIMAL(5,2),
    status ENUM('UP', 'DOWN', 'STABLE') DEFAULT 'STABLE',
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- ダッシュボード更新プロシージャ
DELIMITER //

CREATE PROCEDURE refresh_admin_dashboard()
BEGIN
    DECLARE total_students_val INT;
    DECLARE total_courses_val INT;
    DECLARE avg_attendance_val DECIMAL(5,2);
    DECLARE avg_grade_val DECIMAL(5,2);

    -- 現在の統計を計算
    SELECT COUNT(*) INTO total_students_val FROM students;
    SELECT COUNT(*) INTO total_courses_val FROM courses;

    SELECT ROUND(AVG(CASE WHEN a.status = 'present' THEN 100.0 ELSE 0 END), 2)
    INTO avg_attendance_val
    FROM attendance a
    JOIN course_schedule cs ON a.schedule_id = cs.schedule_id
    WHERE cs.schedule_date >= DATE_SUB(CURRENT_DATE, INTERVAL 30 DAY);

    SELECT ROUND(AVG(score), 2)
    INTO avg_grade_val
    FROM grades
    WHERE submission_date >= DATE_SUB(CURRENT_DATE, INTERVAL 30 DAY);

    -- メトリクス更新（前回値も保存）
    INSERT INTO mv_admin_dashboard (metric_name, metric_value, metric_unit,
previous_value)
    VALUES
        ('total_students', total_students_val, 'count', NULL),
        ('total_courses', total_courses_val, 'count', NULL),
        ('avg_attendance_rate', avg_attendance_val, 'percentage', NULL),
        ('avg_grade_30days', avg_grade_val, 'score', NULL)
```

```
    ON DUPLICATE KEY UPDATE
        previous_value = metric_value,
        metric_value = VALUES(metric_value),
        change_percentage = CASE
            WHEN previous_value > 0 THEN ROUND(((VALUES(metric_value) -
previous_value) / previous_value) * 100, 2)
            ELSE 0
        END,
        status = CASE
            WHEN VALUES(metric_value) > previous_value THEN 'UP'
            WHEN VALUES(metric_value) < previous_value THEN 'DOWN'
            ELSE 'STABLE'
        END,
        last_updated = CURRENT_TIMESTAMP;

END //

DELIMITER ;

-- ダッシュボード更新
CALL refresh_admin_dashboard();

-- ダッシュボード表示
SELECT
    metric_name,
    metric_value,
    metric_unit,
    CONCAT(
        CASE WHEN change_percentage > 0 THEN '+' ELSE '' END,
        change_percentage, '%'
    ) as change_from_previous,
    status,
    last_updated
FROM mv_admin_dashboard
ORDER BY metric_name;
```

2. 監査用マテリアライズドビュー

```
-- データ品質監査用ビュー
CREATE TABLE mv_data_quality_audit (
    audit_date DATE PRIMARY KEY,
    total_students INT,
    students_without_grades INT,
    students_without_attendance INT,
    courses_without_students INT,
    duplicate_grades_count INT,
    data_quality_score DECIMAL(5,2),
    issues_found JSON,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

```
-- データ品質監査プロシージャ
DELIMITER //

CREATE PROCEDURE refresh_data_quality_audit()
BEGIN
    DECLARE audit_date_val DATE DEFAULT CURRENT_DATE;
    DECLARE total_students_val INT;
    DECLARE students_no_grades_val INT;
    DECLARE students_no_attendance_val INT;
    DECLARE courses_no_students_val INT;
    DECLARE duplicate_grades_val INT;
    DECLARE quality_score DECIMAL(5,2);
    DECLARE issues JSON;

    -- データ品質指標の計算
    SELECT COUNT(*) INTO total_students_val FROM students;

    SELECT COUNT(*) INTO students_no_grades_val
    FROM students s
    LEFT JOIN grades g ON s.student_id = g.student_id
    WHERE g.student_id IS NULL;

    SELECT COUNT(*) INTO students_no_attendance_val
    FROM students s
    LEFT JOIN attendance a ON s.student_id = a.student_id
    WHERE a.student_id IS NULL;

    SELECT COUNT(*) INTO courses_no_students_val
    FROM courses c
    LEFT JOIN student_courses sc ON c.course_id = sc.course_id
    WHERE sc.course_id IS NULL;

    SELECT COUNT(*) INTO duplicate_grades_val
    FROM (
        SELECT student_id, course_id, grade_type, COUNT(*) as cnt
        FROM grades
        GROUP BY student_id, course_id, grade_type
        HAVING cnt > 1
    ) duplicates;

    -- 品質スコアの計算（100点満点）
    SET quality_score = 100 -
        (students_no_grades_val / total_students_val * 20) -
        (students_no_attendance_val / total_students_val * 15) -
        (courses_no_students_val * 10) -
        (duplicate_grades_val * 5);

    -- 問題点をJSON形式で記録
    SET issues = JSON_OBJECT(
        'students_without_grades', students_no_grades_val,
        'students_without_attendance', students_no_attendance_val,
        'courses_without_students', courses_no_students_val,
        'duplicate_grades', duplicate_grades_val
    );
```

```
    -- 監査結果を保存
    INSERT INTO mv_data_quality_audit
        (audit_date, total_students, students_without_grades,
students_without_attendance,
         courses_without_students, duplicate_grades_count, data_quality_score,
issues_found)
    VALUES
        (audit_date_val, total_students_val, students_no_grades_val,
students_no_attendance_val,
         courses_no_students_val, duplicate_grades_val, quality_score, issues)
    ON DUPLICATE KEY UPDATE
        total_students = VALUES(total_students),
        students_without_grades = VALUES(students_without_grades),
        students_without_attendance = VALUES(students_without_attendance),
        courses_without_students = VALUES(courses_without_students),
        duplicate_grades_count = VALUES(duplicate_grades_count),
        data_quality_score = VALUES(data_quality_score),
        issues_found = VALUES(issues_found),
        last_updated = CURRENT_TIMESTAMP;

END //

DELIMITER ;

-- データ品質監査実行
CALL refresh_data_quality_audit();

-- 監査結果確認
SELECT
    audit_date,
    data_quality_score,
    students_without_grades,
    students_without_attendance,
    courses_without_students,
    duplicate_grades_count,
    JSON_PRETTY(issues_found) as detailed_issues
FROM mv_data_quality_audit
ORDER BY audit_date DESC;
```

練習問題

問題37-1：基本的なマテリアライズドビュー

以下の要件でマテリアライズドビュー`mv_teacher_workload`を作成してください：

- 各教師の担当講座数、担当学生総数、平均成績を集計
- 更新プロシージャ`refresh_teacher_workload()`も作成
- 初期データを投入して動作確認

問題37-2：増分更新システム

学生テーブルの変更を追跡して、学生一覧のマテリアライズドビューを増分更新するシステムを実装してください：

1. 変更追跡テーブルを作成
2. 学生テーブルにトリガーを設定
3. 増分更新プロシージャを作成
4. テストデータで動作確認

問題37-3：時系列マテリアライズドビュー

日別の新規登録学生数と累計学生数を記録するマテリアライズドビュー`mv_daily_student_stats`を作成してください：

- 日付、新規登録数、累計学生数を記録
- 過去30日分のデータを生成する更新プロシージャ
- グラフ表示用のデータ形式で出力

問題37-4：複合指標マテリアライズドビュー

講座の総合評価指標を計算するマテリアライズドビュー`mv_course_quality_index`を作成してください：

- 平均成績、出席率、受講者数、教師評価を組み合わせた品質指数
- 指数計算式：`(平均成績 * 0.4 + 出席率 * 0.3 + 受講者充足率 * 0.2 + 教師評価 * 0.1)`
- ランキング表示機能も含める

問題37-5：自動更新スケジューラー

以下の機能を持つマテリアライズドビュー自動更新システムを実装してください：

1. 更新スケジュール管理テーブル
2. エラーハンドリング機能付き自動実行プロシージャ
3. 実行ログとパフォーマンス監視
4. 複数のマテリアライズドビューの管理

問題37-6：パフォーマンス最適化

大量データ用のマテリアライズドビューシステムを設計してください：

1. パーティション化戦略の実装
2. インデックス最適化
3. 並行更新対応
4. メモリ使用量とストレージの最適化
5. パフォーマンステストとベンチマーク

解答

解答37-1

```
-- 教師の作業負荷マテリアライズドビュー
CREATE TABLE mv_teacher_workload (
```

```
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64),
    courses_count INT DEFAULT 0,
    total_students INT DEFAULT 0,
    total_grades INT DEFAULT 0,
    average_grade DECIMAL(5,2) DEFAULT 0.00,
    workload_score DECIMAL(5,2) DEFAULT 0.00,  -- 総合作業負荷スコア
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_workload_score (workload_score),
    INDEX idx_courses_count (courses_count)
);

-- 更新プロシージャ
DELIMITER //
CREATE PROCEDURE refresh_teacher_workload()
BEGIN
    TRUNCATE TABLE mv_teacher_workload;

    INSERT INTO mv_teacher_workload
        (teacher_id, teacher_name, courses_count, total_students, total_grades,
average_grade, workload_score)
    SELECT
        t.teacher_id,
        t.teacher_name,
        COUNT(DISTINCT c.course_id) as courses_count,
        COUNT(DISTINCT sc.student_id) as total_students,
        COUNT(g.grade_id) as total_grades,
        ROUND(AVG(g.score), 2) as average_grade,
        -- 作業負荷スコア：講座数×10 + 学生数×0.5 + 成績数×0.1
        ROUND(COUNT(DISTINCT c.course_id) * 10 + COUNT(DISTINCT sc.student_id) *
0.5 + COUNT(g.grade_id) * 0.1, 2) as workload_score
    FROM teachers t
    LEFT JOIN courses c ON t.teacher_id = c.teacher_id
    LEFT JOIN student_courses sc ON c.course_id = sc.course_id
    LEFT JOIN grades g ON c.course_id = g.course_id
    GROUP BY t.teacher_id, t.teacher_name;

    -- ログ記録
    INSERT INTO mv_refresh_log (mv_name, refresh_type, record_count)
    SELECT 'mv_teacher_workload', 'COMPLETE', COUNT(*) FROM mv_teacher_workload;
END //
DELIMITER ;

-- 初期データ投入
CALL refresh_teacher_workload();

-- 結果確認
SELECT * FROM mv_teacher_workload ORDER BY workload_score DESC;
```

解答37-2

```
-- 学生マテリアライズドビュー
CREATE TABLE mv_student_list (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100),
    enrollment_count INT DEFAULT 0,
    grade_count INT DEFAULT 0,
    is_active BOOLEAN DEFAULT TRUE,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 学生変更追跡用トリガー
DELIMITER //
CREATE TRIGGER tr_students_insert_mv
AFTER INSERT ON students
FOR EACH ROW
BEGIN
    INSERT INTO mv_change_log (table_name, operation_type, record_id)
    VALUES ('students', 'INSERT', NEW.student_id);
END //

CREATE TRIGGER tr_students_update_mv
AFTER UPDATE ON students
FOR EACH ROW
BEGIN
    INSERT INTO mv_change_log (table_name, operation_type, record_id)
    VALUES ('students', 'UPDATE', NEW.student_id);
END //

CREATE TRIGGER tr_students_delete_mv
AFTER DELETE ON students
FOR EACH ROW
BEGIN
    INSERT INTO mv_change_log (table_name, operation_type, record_id)
    VALUES ('students', 'DELETE', OLD.student_id);
END //
DELIMITER ;

-- 増分更新プロシージャ
DELIMITER //
CREATE PROCEDURE incremental_refresh_student_list()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE affected_student_id BIGINT;
    DECLARE operation_type VARCHAR(10);

    DECLARE change_cursor CURSOR FOR
        SELECT DISTINCT record_id, operation_type
        FROM mv_change_log
        WHERE table_name = 'students' AND processed = FALSE;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN change_cursor;
```

```
    change_loop: LOOP
        FETCH change_cursor INTO affected_student_id, operation_type;
        IF done THEN
            LEAVE change_loop;
        END IF;

        IF operation_type = 'DELETE' THEN
            DELETE FROM mv_student_list WHERE student_id = affected_student_id;
        ELSE
            INSERT INTO mv_student_list
                (student_id, student_name, enrollment_count, grade_count)
            SELECT
                s.student_id,
                s.student_name,
                COUNT(DISTINCT sc.course_id) as enrollment_count,
                COUNT(g.grade_id) as grade_count
            FROM students s
            LEFT JOIN student_courses sc ON s.student_id = sc.student_id
            LEFT JOIN grades g ON s.student_id = g.student_id
            WHERE s.student_id = affected_student_id
            GROUP BY s.student_id, s.student_name
            ON DUPLICATE KEY UPDATE
                student_name = VALUES(student_name),
                enrollment_count = VALUES(enrollment_count),
                grade_count = VALUES(grade_count),
                last_updated = CURRENT_TIMESTAMP;
        END IF;
    END LOOP;

    CLOSE change_cursor;

    -- 処理済みマーク
    UPDATE mv_change_log
    SET processed = TRUE
    WHERE table_name = 'students' AND processed = FALSE;
END //
DELIMITER ;

-- 初期データ投入
INSERT INTO mv_student_list (student_id, student_name, enrollment_count,
grade_count)
SELECT
    s.student_id,
    s.student_name,
    COUNT(DISTINCT sc.course_id) as enrollment_count,
    COUNT(g.grade_id) as grade_count
FROM students s
LEFT JOIN student_courses sc ON s.student_id = sc.student_id
LEFT JOIN grades g ON s.student_id = g.student_id
GROUP BY s.student_id, s.student_name;

-- テスト：学生データ更新
UPDATE students SET student_name = '田中太郎（更新）' WHERE student_id = 301;
```

```
-- 増分更新実行
CALL incremental_refresh_student_list();

-- 結果確認
SELECT * FROM mv_student_list WHERE student_id = 301;
```

解答37-3

```
-- 日別学生統計マテリアライズドビュー
CREATE TABLE mv_daily_student_stats (
    stat_date DATE PRIMARY KEY,
    new_registrations INT DEFAULT 0,
    cumulative_students INT DEFAULT 0,
    registration_rate DECIMAL(5,2) DEFAULT 0.00,  -- 前日比増加率
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_stat_date (stat_date)
);

-- 過去30日分データ生成プロシージャ
DELIMITER //
CREATE PROCEDURE refresh_daily_student_stats_30days()
BEGIN
    DECLARE current_date DATE;
    DECLARE end_date DATE DEFAULT CURRENT_DATE;
    DECLARE start_date DATE DEFAULT DATE_SUB(CURRENT_DATE, INTERVAL 30 DAY);
    DECLARE new_reg_count INT;
    DECLARE cumulative_count INT;
    DECLARE prev_cumulative INT DEFAULT 0;
    DECLARE rate_calc DECIMAL(5,2);

    -- 既存の30日分データを削除
    DELETE FROM mv_daily_student_stats
    WHERE stat_date >= start_date;

    SET current_date = start_date;

    WHILE current_date <= end_date DO
        -- その日の新規登録数を計算（admission_dateを使用）
        SELECT COUNT(*) INTO new_reg_count
        FROM students
        WHERE DATE(created_at) = current_date;

        -- 累計学生数を計算
        SELECT COUNT(*) INTO cumulative_count
        FROM students
        WHERE DATE(created_at) <= current_date;

        -- 前日比増加率を計算
        IF prev_cumulative > 0 THEN
```

```
            SET rate_calc = ROUND(((cumulative_count - prev_cumulative) /
prev_cumulative) * 100, 2);
        ELSE
            SET rate_calc = 0.00;
        END IF;

        -- データ挿入
        INSERT INTO mv_daily_student_stats
            (stat_date, new_registrations, cumulative_students, registration_rate)
        VALUES
            (current_date, new_reg_count, cumulative_count, rate_calc);

        SET prev_cumulative = cumulative_count;
        SET current_date = DATE_ADD(current_date, INTERVAL 1 DAY);
    END WHILE;

    -- ログ記録
    INSERT INTO mv_refresh_log (mv_name, refresh_type, record_count)
    VALUES ('mv_daily_student_stats', 'COMPLETE', 30);
END //
DELIMITER ;

-- 更新実行
CALL refresh_daily_student_stats_30days();

-- グラフ表示用データ出力
SELECT
    stat_date,
    new_registrations,
    cumulative_students,
    registration_rate,
    -- 7日移動平均
    ROUND(AVG(new_registrations) OVER (
        ORDER BY stat_date
        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ), 1) as moving_avg_7days
FROM mv_daily_student_stats
ORDER BY stat_date;
```

解答37-4

```
-- 講座品質指数マテリアライズドビュー
CREATE TABLE mv_course_quality_index (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128),
    teacher_name VARCHAR(64),
    average_score DECIMAL(5,2) DEFAULT 0.00,
    attendance_rate DECIMAL(5,2) DEFAULT 0.00,
    enrollment_rate DECIMAL(5,2) DEFAULT 0.00,   -- 受講者充足率
    teacher_rating DECIMAL(3,2) DEFAULT 3.00,    -- 教師評価（1-5）
    quality_index DECIMAL(5,2) DEFAULT 0.00,     -- 総合品質指数
```

```
    quality_rank INT DEFAULT 0,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_quality_index (quality_index),
    INDEX idx_quality_rank (quality_rank)
);

-- 品質指数計算プロシージャ
DELIMITER //
CREATE PROCEDURE refresh_course_quality_index()
BEGIN
    TRUNCATE TABLE mv_course_quality_index;

    INSERT INTO mv_course_quality_index
        (course_id, course_name, teacher_name, average_score, attendance_rate,
         enrollment_rate, teacher_rating, quality_index)
    SELECT
        c.course_id,
        c.course_name,
        t.teacher_name,
        COALESCE(ROUND(AVG(g.score), 2), 0) as average_score,
        COALESCE(ROUND(AVG(CASE WHEN a.status = 'present' THEN 100.0 ELSE 0 END),
2), 0) as attendance_rate,
        ROUND((COUNT(DISTINCT sc.student_id) / 30.0) * 100, 2) as enrollment_rate,
-- 定員30名と仮定
        4.0 as teacher_rating,  -- 固定値（実際は評価テーブルから取得）
        -- 品質指数計算
        ROUND(
            COALESCE(AVG(g.score), 0) * 0.4 +
            COALESCE(AVG(CASE WHEN a.status = 'present' THEN 100.0 ELSE 0 END), 0)
* 0.3 +
            LEAST((COUNT(DISTINCT sc.student_id) / 30.0) * 100, 100) * 0.2 +
            4.0 * 20 * 0.1,  -- 教師評価を100点満点に換算
            2
        ) as quality_index
    FROM courses c
    JOIN teachers t ON c.teacher_id = t.teacher_id
    LEFT JOIN student_courses sc ON c.course_id = sc.course_id
    LEFT JOIN grades g ON 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
    GROUP BY c.course_id, c.course_name, t.teacher_name;

    -- ランキング設定
    SET @rank = 0;
    UPDATE mv_course_quality_index cqi1
    JOIN (
        SELECT course_id, @rank := @rank + 1 as new_rank
        FROM mv_course_quality_index
        ORDER BY quality_index DESC
    ) cqi2 ON cqi1.course_id = cqi2.course_id
    SET cqi1.quality_rank = cqi2.new_rank;

    -- ログ記録
```

```
    INSERT INTO mv_refresh_log (mv_name, refresh_type, record_count)
    SELECT 'mv_course_quality_index', 'COMPLETE', COUNT(*) FROM
mv_course_quality_index;
END //
DELIMITER ;

-- 更新実行
CALL refresh_course_quality_index();

-- ランキング表示
SELECT
    quality_rank,
    course_name,
    teacher_name,
    quality_index,
    average_score,
    attendance_rate,
    enrollment_rate
FROM mv_course_quality_index
ORDER BY quality_rank
LIMIT 10;
```

解答37-5

```
-- エラーログテーブル
CREATE TABLE mv_error_log (
    error_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    mv_name VARCHAR(100),
    error_message TEXT,
    error_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    execution_context JSON
);

-- 改良された自動実行プロシージャ
DELIMITER //
CREATE PROCEDURE execute_scheduled_refreshes_with_error_handling()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE mv_name_var VARCHAR(100);
    DECLARE refresh_type_var VARCHAR(20);
    DECLARE refresh_freq_var VARCHAR(20);
    DECLARE start_time TIMESTAMP;
    DECLARE execution_time INT;
    DECLARE error_occurred BOOLEAN DEFAULT FALSE;
    DECLARE error_message TEXT DEFAULT '';

    DECLARE schedule_cursor CURSOR FOR
        SELECT mv_name, refresh_type, refresh_frequency
        FROM mv_refresh_schedule
        WHERE is_active = TRUE
        AND next_refresh <= NOW();
```

```
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
        GET DIAGNOSTICS CONDITION 1
            error_message = MESSAGE_TEXT;
        SET error_occurred = TRUE;
    END;

    OPEN schedule_cursor;

    refresh_loop: LOOP
        FETCH schedule_cursor INTO mv_name_var, refresh_type_var,
refresh_freq_var;
        IF done THEN
            LEAVE refresh_loop;
        END IF;

        SET start_time = NOW();
        SET error_occurred = FALSE;
        SET error_message = '';

        -- エラーハンドリング付きで更新実行
        BEGIN
            DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
            BEGIN
                GET DIAGNOSTICS CONDITION 1
                    error_message = MESSAGE_TEXT;
                SET error_occurred = TRUE;
            END;

            CASE mv_name_var
                WHEN 'mv_student_grade_summary' THEN
                    IF refresh_type_var = 'COMPLETE' THEN
                        CALL refresh_student_grade_summary();
                    ELSE
                        CALL incremental_refresh_student_grades();
                    END IF;

                WHEN 'mv_teacher_workload' THEN
                    CALL refresh_teacher_workload();

                WHEN 'mv_course_quality_index' THEN
                    CALL refresh_course_quality_index();
            END CASE;
        END;

        SET execution_time = TIMESTAMPDIFF(SECOND, start_time, NOW());

        IF error_occurred THEN
            -- エラーログ記録
            INSERT INTO mv_error_log (mv_name, error_message, execution_context)
            VALUES (mv_name_var, error_message, JSON_OBJECT(
                'refresh_type', refresh_type_var,
```

```
                'execution_time', execution_time,
                'start_time', start_time
            ));

            -- スケジュールは次回に延期
            UPDATE mv_refresh_schedule
            SET next_refresh = DATE_ADD(NOW(), INTERVAL 1 HOUR)
            WHERE mv_name = mv_name_var;
        ELSE
            -- 正常終了時のスケジュール更新
            UPDATE mv_refresh_schedule
            SET
                last_refresh = start_time,
                next_refresh = CASE
                    WHEN refresh_freq_var = 'HOURLY' THEN DATE_ADD(start_time,
INTERVAL 1 HOUR)
                    WHEN refresh_freq_var = 'DAILY' THEN DATE_ADD(start_time,
INTERVAL 1 DAY)
                    WHEN refresh_freq_var = 'WEEKLY' THEN DATE_ADD(start_time,
INTERVAL 1 WEEK)
                    WHEN refresh_freq_var = 'MONTHLY' THEN DATE_ADD(start_time,
INTERVAL 1 MONTH)
                END
            WHERE mv_name = mv_name_var;

            -- 成功ログ更新
            UPDATE mv_refresh_log
            SET execution_time_ms = execution_time * 1000
            WHERE mv_name = mv_name_var
            AND refresh_time >= start_time
            ORDER BY refresh_time DESC
            LIMIT 1;
        END IF;

    END LOOP;

    CLOSE schedule_cursor;
END //
DELIMITER ;

-- 監視クエリ
SELECT
    'スケジュール状況' as report_type,
    mv_name,
    refresh_frequency,
    last_refresh,
    next_refresh,
    CASE WHEN next_refresh <= NOW() THEN '実行待ち' ELSE '待機中' END as status
FROM mv_refresh_schedule
UNION ALL
SELECT
    'エラー状況',
    mv_name,
    COUNT(*),
```

```
    MAX(error_time),
    NULL,
    CONCAT(COUNT(*), '件のエラー')
FROM mv_error_log
WHERE error_time >= DATE_SUB(NOW(), INTERVAL 24 HOUR)
GROUP BY mv_name;
```

解答37-6

```
-- パーティション化マテリアライズドビュー（年月別）
CREATE TABLE mv_large_performance_data (
    data_date DATE NOT NULL,
    student_id BIGINT NOT NULL,
    performance_metrics JSON,
    aggregated_score DECIMAL(8,2),
    ranking_position INT,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    PRIMARY KEY (data_date, student_id),
    INDEX idx_aggregated_score (aggregated_score),
    INDEX idx_ranking (ranking_position)
) PARTITION BY RANGE (YEAR(data_date) * 100 + MONTH(data_date)) (
    PARTITION p202401 VALUES LESS THAN (202402),
    PARTITION p202402 VALUES LESS THAN (202403),
    PARTITION p202403 VALUES LESS THAN (202404),
    PARTITION p202404 VALUES LESS THAN (202405),
    PARTITION p202405 VALUES LESS THAN (202406),
    PARTITION p202406 VALUES LESS THAN (202407),
    PARTITION p202407 VALUES LESS THAN (202408),
    PARTITION p202408 VALUES LESS THAN (202409),
    PARTITION p202409 VALUES LESS THAN (202410),
    PARTITION p202410 VALUES LESS THAN (202411),
    PARTITION p202411 VALUES LESS THAN (202412),
    PARTITION p202412 VALUES LESS THAN (202501),
    PARTITION p202501 VALUES LESS THAN (202502),
    PARTITION p202502 VALUES LESS THAN (202503),
    PARTITION p202503 VALUES LESS THAN (202504),
    PARTITION p202504 VALUES LESS THAN (202505),
    PARTITION p202505 VALUES LESS THAN (202506),
    PARTITION p202506 VALUES LESS THAN (MAXVALUE)
);

-- 並行更新対応プロシージャ
DELIMITER //
CREATE PROCEDURE refresh_large_performance_data_concurrent(
    IN target_date DATE,
    IN worker_id INT,
    IN total_workers INT
)
BEGIN
    DECLARE batch_size INT DEFAULT 1000;
```

```
    DECLARE offset_val INT DEFAULT 0;
    DECLARE processed_count INT DEFAULT 0;

    -- 該当日付のパーティションデータを削除
    DELETE FROM mv_large_performance_data
    WHERE data_date = target_date
    AND MOD(student_id, total_workers) = worker_id - 1;

    -- バッチ処理でデータ挿入
    batch_loop: LOOP
        SET @sql = CONCAT(
            'INSERT INTO mv_large_performance_data ',
            '(data_date, student_id, performance_metrics, aggregated_score,
ranking_position) ',
            'SELECT ?, s.student_id, ',
            'JSON_OBJECT(',
                '"avg_grade", COALESCE(AVG(g.score), 0), ',
                '"attendance_rate", COALESCE(AVG(CASE WHEN a.status = "present"
THEN 100.0 ELSE 0 END), 0), ',
                '"assignment_completion", COALESCE(COUNT(g.grade_id) / 10.0 * 100,
0)',
            '), ',
            'ROUND(COALESCE(AVG(g.score), 0) * 0.6 + ',
            'COALESCE(AVG(CASE WHEN a.status = "present" THEN 100.0 ELSE 0 END),
0) * 0.4, 2), ',
            '0 ',
            'FROM students s ',
            'LEFT JOIN grades g ON s.student_id = g.student_id ',
            'LEFT JOIN attendance a ON s.student_id = a.student_id ',
            'WHERE MOD(s.student_id, ?) = ? ',
            'GROUP BY s.student_id ',
            'LIMIT ? OFFSET ?'
        );

        PREPARE stmt FROM @sql;
        EXECUTE stmt USING target_date, total_workers, worker_id - 1, batch_size,
offset_val;
        DEALLOCATE PREPARE stmt;

        -- 処理件数確認
        SET processed_count = ROW_COUNT();
        IF processed_count < batch_size THEN
            LEAVE batch_loop;
        END IF;

        SET offset_val = offset_val + batch_size;
    END LOOP;

    -- ランキング更新（該当ワーカーの担当分のみ）
    SET @rank = 0;
    UPDATE mv_large_performance_data lpd1
    JOIN (
        SELECT
            student_id,
```

```
            @rank := @rank + 1 as new_rank
        FROM mv_large_performance_data
        WHERE data_date = target_date
        AND MOD(student_id, total_workers) = worker_id - 1
        ORDER BY aggregated_score DESC
    ) lpd2 ON lpd1.student_id = lpd2.student_id
    SET lpd1.ranking_position = lpd2.new_rank
    WHERE lpd1.data_date = target_date;

END //
DELIMITER ;

-- パフォーマンステスト
DELIMITER //
CREATE PROCEDURE benchmark_materialized_view_performance()
BEGIN
    DECLARE start_time TIMESTAMP;
    DECLARE end_time TIMESTAMP;
    DECLARE mv_time INT;
    DECLARE regular_time INT;

    -- マテリアライズドビューのクエリ
    SET start_time = NOW(6);
    SELECT COUNT(*), AVG(aggregated_score), MAX(aggregated_score)
    FROM mv_large_performance_data
    WHERE data_date = CURRENT_DATE;
    SET end_time = NOW(6);
    SET mv_time = TIMESTAMPDIFF(MICROSECOND, start_time, end_time);

    -- 通常クエリ
    SET start_time = NOW(6);
    SELECT COUNT(*), AVG(score_calc), MAX(score_calc)
    FROM (
        SELECT s.student_id,
               ROUND(COALESCE(AVG(g.score), 0) * 0.6 +
                     COALESCE(AVG(CASE WHEN a.status = 'present' THEN 100.0 ELSE 0
END), 0) * 0.4, 2) as score_calc
        FROM students s
        LEFT JOIN grades g ON s.student_id = g.student_id
        LEFT JOIN attendance a ON s.student_id = a.student_id
        GROUP BY s.student_id
    ) calculated;
    SET end_time = NOW(6);
    SET regular_time = TIMESTAMPDIFF(MICROSECOND, start_time, end_time);

    -- ベンチマーク結果
    SELECT
        mv_time as materialized_view_microseconds,
        regular_time as regular_query_microseconds,
        ROUND(regular_time / mv_time, 2) as performance_improvement,
        ROUND((regular_time - mv_time) / 1000, 2) as time_saved_ms;
END //
DELIMITER ;
```

```
-- ベンチマーク実行
CALL benchmark_materialized_view_performance();

-- ストレージ使用量分析
SELECT
    PARTITION_NAME,
    TABLE_ROWS,
    ROUND(DATA_LENGTH / 1024 / 1024, 2) as data_mb,
    ROUND(INDEX_LENGTH / 1024 / 1024, 2) as index_mb
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = DATABASE()
AND TABLE_NAME = 'mv_large_performance_data'
AND PARTITION_NAME IS NOT NULL
ORDER BY PARTITION_NAME;
```

まとめ

この章では、MySQLでのマテリアライズドビューの実装について詳しく学びました：

1. **マテリアライズドビューの基本概念**：
   - 通常のビューとの違い
   - パフォーマンス向上の仕組み
   - MySQLでの代替実装方法
2. **基本的な実装パターン**：
   - 集計テーブルの作成
   - 更新プロシージャの設計
   - 初期データの投入
3. **増分更新システム**：
   - 変更追跡メカニズム
   - トリガーによる自動検知
   - 効率的な部分更新
4. **複雑な集計ビューの構築**：
   - 月別統計の管理
   - パフォーマンス分析指標
   - JSON形式での柔軟なデータ格納
5. **自動更新システム**：
   - スケジュール管理
   - エラーハンドリング
   - 実行ログと監視
6. **パフォーマンス最適化**：
   - パーティション化戦略

    - インデックス設計
    - 並行処理対応

7. **実践的な運用例**：

    - ダッシュボード用ビュー
    - データ品質監査
    - ベンチマークと最適化

8. **大規模データ対応**：

    - 分散更新処理
    - メモリ使用量最適化
    - ストレージ効率化

マテリアライズドビューは、適切に設計・運用することで劇的なパフォーマンス向上をもたらします。ただし、データの整合性とリフレッシュコストのバランスを慎重に考慮する必要があります。

次の章では、「スキーマ設計：正規化と非正規化」について学び、効率的なデータベース設計の原則を理解していきます。

---

38. スキーマ設計：正規化と非正規化

はじめに

前章では、マテリアライズドビューによるパフォーマンス向上について学習しました。この章では、データベース設計の基礎となる「正規化」と、実際の運用で必要となる「非正規化」について詳しく学習します。

正規化は、データベース設計における最も重要な概念の一つで、データの冗長性を排除し、更新異常を防ぐための理論的基盤です。一方、非正規化は、パフォーマンス要件を満たすために意図的に冗長性を導入する実践的な手法です。

スキーマ設計が重要となる場面の例：

- 「新しい学校管理システムのデータベースを設計したい」
- 「既存のシステムでデータの不整合が頻繁に発生している」
- 「複雑なクエリの実行時間を短縮したい」
- 「データの冗長性を排除してストレージを効率化したい」
- 「データの整合性を保ちつつ高速なアクセスを実現したい」
- 「システムの拡張性を考慮した設計にしたい」
- 「レポート機能の高速化が必要」

この章では、理論的な正規化の原則から、実践的な非正規化の手法まで、バランスの取れたスキーマ設計について学んでいきます。

正規化とは

正規化は、データベースの設計において、データの冗長性を排除し、データの整合性を保つためのプロセスです。エドガー・F・コッド博士によって提唱された理論で、複数の正規形が定義されています。

> **用語解説：**
>
> - **正規化（Normalization）**：データの冗長性を排除し、整合性を保つためのデータベース設計手法です。
> - **正規形（Normal Form）**：正規化の各段階を表す形式です（1NF、2NF、3NF、BCNF等）。
> - **非正規化（Denormalization）**：パフォーマンス向上のために意図的に冗長性を導入することです。
> - **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。
> - **関数従属（Functional Dependency）**：あるカラムの値が決まると別のカラムの値が一意に決まる関係です。
> - **部分関数従属（Partial Dependency）**：複合主キーの一部のカラムに従属する関係です。
> - **推移関数従属（Transitive Dependency）**：A→B→Cという間接的な従属関係です。
> - **更新異常（Update Anomaly）**：データ更新時に発生する不整合や問題です。
> - **挿入異常（Insert Anomaly）**：新しいデータを挿入する際の問題です。
> - **削除異常（Delete Anomaly）**：データ削除時に必要な情報まで失われる問題です。

正規化の目的と利点

正規化の目的

1. **データの冗長性排除**：同じ情報の重複を避ける
2. **更新異常の防止**：データ更新時の不整合を防ぐ
3. **ストレージ効率化**：無駄な容量使用を削減
4. **データ整合性の向上**：一貫性のあるデータ管理

正規化の利点と課題

利点	課題
データの整合性向上	クエリの複雑化
ストレージ使用量削減	結合処理による性能低下
更新処理の効率化	設計の複雑化
保守性の向上	理解の難しさ

非正規化データの問題例

まず、正規化されていないデータの問題を確認しましょう。

```
-- 非正規化された学生成績テーブル（問題のある設計）
CREATE TABLE bad_student_grades (
    record_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT,
    student_name VARCHAR(100),
    student_email VARCHAR(255),
    course_id VARCHAR(16),
    course_name VARCHAR(128),
    teacher_id BIGINT,
```

```
    teacher_name VARCHAR(64),
    teacher_email VARCHAR(255),
    grade_type VARCHAR(32),
    score DECIMAL(5,2),
    max_score DECIMAL(5,2),
    submission_date DATE
);

-- 問題のあるサンプルデータ
INSERT INTO bad_student_grades VALUES
(1, 301, '黒沢春馬', 'kurosawa@school.com', '1', 'ITのための基礎知識', 101, '寺内鞍',
'terauchi@school.com', '中間テスト', 85.0, 100.0, '2025-05-15'),
(2, 301, '黒沢春馬', 'kurosawa@school.com', '1', 'ITのための基礎知識', 101, '寺内鞍',
'terauchi@school.com', 'レポート1', 90.0, 100.0, '2025-05-20'),
(3, 301, '黒沢春馬', 'kurosawa@school.com', '2', 'UNIX入門', 102, '佐野真',
'sano@school.com', '中間テスト', 78.0, 100.0, '2025-05-18'),
(4, 302, '新垣愛留', 'aragaki@school.com', '1', 'ITのための基礎知識', 101, '寺内鞍',
'terauchi@school.com', '中間テスト', 92.0, 100.0, '2025-05-15');

-- この設計の問題点を確認
SELECT * FROM bad_student_grades;
```

非正規化データの問題点

```
-- 問題1：冗長性（同じ情報の重複）
SELECT
    COUNT(*) as total_records,
    COUNT(DISTINCT student_name) as unique_students,
    COUNT(DISTINCT course_name) as unique_courses,
    COUNT(DISTINCT teacher_name) as unique_teachers
FROM bad_student_grades;

-- 問題2：更新異常（学生名を変更する場合）
UPDATE bad_student_grades
SET student_name = '黒沢春馬（更新）'
WHERE student_id = 301 AND record_id = 1;  -- 一部のレコードのみ更新

-- 不整合の発生確認
SELECT student_id, student_name, COUNT(*) as count
FROM bad_student_grades
WHERE student_id = 301
GROUP BY student_id, student_name;

-- 問題3：挿入異常（学生情報だけを登録したい場合）
-- 成績データなしに学生情報だけを入れることができない

-- 問題4：削除異常（最後の成績レコードを削除すると学生情報も失われる）
```

第1正規形（1NF）

第1正規形は、すべてのカラムが原子値（分割できない値）を持ち、繰り返しグループがない状態です。

1NFの条件

1. **原子値**：各カラムには分割できない単一の値のみ
2. **繰り返しグループなし**：同じ種類のデータを複数のカラムに分けない
3. **一意な行**：各行が一意に識別できる

1NF違反の例と修正

```
-- 1NF違反の例（繰り返しグループと複合値）
CREATE TABLE unnormalized_courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128),
    teacher_names TEXT,  -- 複数の教師名をカンマ区切りで格納（1NF違反）
    student_list TEXT,   -- 複数の学生IDをカンマ区切りで格納（1NF違反）
    schedule_info VARCHAR(500)  -- 複合情報（曜日、時限、教室を一つのカラムに）
);

-- 1NF違反データの例
INSERT INTO unnormalized_courses VALUES
('1', 'ITのための基礎知識', '寺内鞍,佐野真', '301,302,303,304', '月曜日1限:101A,水曜日2
限:102B');

-- 1NFに修正したテーブル設計
CREATE TABLE courses_1nf (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL
);

CREATE TABLE course_teachers_1nf (
    course_id VARCHAR(16),
    teacher_id BIGINT,
    PRIMARY KEY (course_id, teacher_id)
);

CREATE TABLE course_students_1nf (
    course_id VARCHAR(16),
    student_id BIGINT,
    enrollment_date DATE,
    PRIMARY KEY (course_id, student_id)
);

CREATE TABLE course_schedules_1nf (
    schedule_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_id VARCHAR(16),
    day_of_week TINYINT,  -- 1=月曜日
    period TINYINT,       -- 時限
    classroom_id VARCHAR(16)
);

-- 1NFに準拠したデータ挿入
```

```
INSERT INTO courses_1nf VALUES ('1', 'ITのための基礎知識');

INSERT INTO course_teachers_1nf VALUES
('1', 101), ('1', 102);

INSERT INTO course_students_1nf VALUES
('1', 301, '2025-04-01'),
('1', 302, '2025-04-01'),
('1', 303, '2025-04-01');

INSERT INTO course_schedules_1nf VALUES
(1, '1', 1, 1, '101A'),
(2, '1', 3, 2, '102B');
```

第2正規形（2NF）

第2正規形は、1NFであり、かつ非キー属性が主キー全体に完全関数従属している状態です。

2NFの条件

1. **1NFである**
2. **部分関数従属がない**：複合主キーの一部分だけに依存するカラムがない

2NF違反の例と修正

```
-- 2NF違反の例（部分関数従属がある）
CREATE TABLE student_course_grades_not_2nf (
    student_id BIGINT,
    course_id VARCHAR(16),
    grade_type VARCHAR(32),

    -- 主キー（複合キー）
    PRIMARY KEY (student_id, course_id, grade_type),

    -- 以下のカラムに部分関数従属がある
    student_name VARCHAR(100),    -- student_idのみに依存（部分関数従属）
    student_email VARCHAR(255),   -- student_idのみに依存（部分関数従属）
    course_name VARCHAR(128),     -- course_idのみに依存（部分関数従属）
    teacher_name VARCHAR(64),     -- course_idのみに依存（部分関数従属）

    -- 以下は主キー全体に依存（完全関数従属）
    score DECIMAL(5,2),
    max_score DECIMAL(5,2),
    submission_date DATE
);

-- 部分関数従属の確認
INSERT INTO student_course_grades_not_2nf VALUES
(301, '1', '中間テスト', '黒沢春馬', 'kurosawa@school.com', 'ITのための基礎知識', '寺内
鞍', 85.0, 100.0, '2025-05-15'),
(301, '1', 'レポート1', '黒沢春馬', 'kurosawa@school.com', 'ITのための基礎知識', '寺内
```

```
鞍', 90.0, 100.0, '2025-05-20'),
(301, '2', '中間テスト', '黒沢春馬', 'kurosawa@school.com', 'UNIX入門', '佐野真',
78.0, 100.0, '2025-05-18');

-- 問題の確認：冗長性と更新異常
SELECT student_id, student_name, course_id, course_name, COUNT(*) as duplicates
FROM student_course_grades_not_2nf
GROUP BY student_id, student_name, course_id, course_name;

-- 2NFに修正したテーブル設計
CREATE TABLE students_2nf (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(255) UNIQUE
);

CREATE TABLE courses_2nf (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL
);

CREATE TABLE teachers_2nf (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64) NOT NULL
);

CREATE TABLE grades_2nf (
    student_id BIGINT,
    course_id VARCHAR(16),
    grade_type VARCHAR(32),
    score DECIMAL(5,2),
    max_score DECIMAL(5,2),
    submission_date DATE,

    PRIMARY KEY (student_id, course_id, grade_type),
    FOREIGN KEY (student_id) REFERENCES students_2nf(student_id),
    FOREIGN KEY (course_id) REFERENCES courses_2nf(course_id)
);

-- 2NFに準拠したデータ挿入
INSERT INTO teachers_2nf VALUES (101, '寺内鞍'), (102, '佐野真');
INSERT INTO students_2nf VALUES (301, '黒沢春馬', 'kurosawa@school.com');
INSERT INTO courses_2nf VALUES ('1', 'ITのための基礎知識', 101), ('2', 'UNIX入門',
102);

INSERT INTO grades_2nf VALUES
(301, '1', '中間テスト', 85.0, 100.0, '2025-05-15'),
(301, '1', 'レポート1', 90.0, 100.0, '2025-05-20'),
(301, '2', '中間テスト', 78.0, 100.0, '2025-05-18');
```

第3正規形（3NF）

第3正規形は、2NFであり、かつ推移関数従属がない状態です。

3NFの条件

1. **2NFである**
2. **推移関数従属がない**：非キー属性が他の非キー属性を通じて主キーに依存しない

3NF違反の例と修正

```
-- 3NF違反の例（推移関数従属がある）
CREATE TABLE courses_with_department_not_3nf (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT NOT NULL,
    teacher_name VARCHAR(64),        -- teacher_idに依存（推移関数従属）
    department_id INT,               -- teacher_idに依存（推移関数従属）
    department_name VARCHAR(100),    -- department_idに依存（推移関数従属）
    department_building VARCHAR(50)  -- department_idに依存（推移関数従属）
);

-- 推移関数従属のあるデータ
INSERT INTO courses_with_department_not_3nf VALUES
('1', 'ITのための基礎知識', 101, '寺内鞍', 1, '情報工学科', 'A棟'),
('2', 'UNIX入門', 102, '佐野真', 1, '情報工学科', 'A棟'),
('3', 'データベース基礎', 101, '寺内鞍', 1, '情報工学科', 'A棟');

-- 推移関数従属の確認
-- course_id → teacher_id → teacher_name (推移関数従属)
-- course_id → teacher_id → department_id → department_name (推移関数従属)
SELECT
    teacher_id, teacher_name, department_id, department_name,
    COUNT(DISTINCT course_id) as course_count
FROM courses_with_department_not_3nf
GROUP BY teacher_id, teacher_name, department_id, department_name;

-- 3NFに修正したテーブル設計
CREATE TABLE departments_3nf (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL,
    department_building VARCHAR(50)
);

CREATE TABLE teachers_3nf (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64) NOT NULL,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments_3nf(department_id)
);

CREATE TABLE courses_3nf (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
```

```
    teacher_id BIGINT NOT NULL,
    FOREIGN KEY (teacher_id) REFERENCES teachers_3nf(teacher_id)
);

-- 3NFに準拠したデータ挿入
INSERT INTO departments_3nf VALUES (1, '情報工学科', 'A棟'), (2, '数学科', 'B棟');

INSERT INTO teachers_3nf VALUES
(101, '寺内鞍', 1),
(102, '佐野真', 1),
(103, '田中教授', 2);

INSERT INTO courses_3nf VALUES
('1', 'ITのための基礎知識', 101),
('2', 'UNIX入門', 102),
('3', 'データベース基礎', 101);

-- 正規化後のクエリ例（結合が必要）
SELECT
    c.course_id,
    c.course_name,
    t.teacher_name,
    d.department_name,
    d.department_building
FROM courses_3nf c
JOIN teachers_3nf t ON c.teacher_id = t.teacher_id
JOIN departments_3nf d ON t.department_id = d.department_id;
```

正規化の実践例：学校データベースの段階的設計

ステップ1：要件分析

```
-- 学校データベースで管理したい情報
-- 1. 学生情報（ID、名前、メール、学年、専攻）
-- 2. 教師情報（ID、名前、所属学科、研究分野）
-- 3. 講座情報（ID、名前、単位数、難易度）
-- 4. 受講情報（学生と講座の関係、受講日）
-- 5. 成績情報（学生、講座、評価タイプ、点数）
-- 6. 教室情報（ID、名前、収容人数、設備）
-- 7. 授業スケジュール（講座、教室、日時）
```

ステップ2：1NFの適用

```
-- 原子値の確保、繰り返しグループの排除
CREATE TABLE entities_1nf (
    -- 学生エンティティ
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(255) UNIQUE,
```

```
    student_grade INT,
    student_major VARCHAR(100),

    -- 教師エンティティ
    teacher_id BIGINT,
    teacher_name VARCHAR(64),
    teacher_department VARCHAR(100),
    teacher_specialty VARCHAR(200),

    -- 講座エンティティ
    course_id VARCHAR(16),
    course_name VARCHAR(128),
    course_credits INT,
    course_difficulty ENUM('beginner', 'intermediate', 'advanced'),

    -- 受講関係
    enrollment_date DATE,

    -- 成績情報
    grade_type VARCHAR(32),
    score DECIMAL(5,2),
    max_score DECIMAL(5,2),

    -- 教室情報
    classroom_id VARCHAR(16),
    classroom_name VARCHAR(64),
    classroom_capacity INT,

    -- スケジュール情報
    schedule_date DATE,
    schedule_period INT
);

-- この設計は1NFだが、まだ最適ではない
```

ステップ3：2NFの適用

```
-- 部分関数従属を排除した設計

-- 学生テーブル
CREATE TABLE students_normalized (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(255) UNIQUE,
    student_grade INT CHECK (student_grade >= 1 AND student_grade <= 6),
    student_major VARCHAR(100)
);

-- 教師テーブル
CREATE TABLE teachers_normalized (
    teacher_id BIGINT PRIMARY KEY,
```

```
    teacher_name VARCHAR(64) NOT NULL,
    teacher_department VARCHAR(100),
    teacher_specialty VARCHAR(200)
);

-- 講座テーブル
CREATE TABLE courses_normalized (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    course_credits INT DEFAULT 2,
    course_difficulty ENUM('beginner', 'intermediate', 'advanced') DEFAULT
'beginner',
    teacher_id BIGINT,
    FOREIGN KEY (teacher_id) REFERENCES teachers_normalized(teacher_id)
);

-- 教室テーブル
CREATE TABLE classrooms_normalized (
    classroom_id VARCHAR(16) PRIMARY KEY,
    classroom_name VARCHAR(64) NOT NULL,
    classroom_capacity INT,
    classroom_equipment TEXT
);

-- 受講テーブル（多対多関係の解決）
CREATE TABLE enrollments_normalized (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    status ENUM('enrolled', 'completed', 'dropped') DEFAULT 'enrolled',
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students_normalized(student_id),
    FOREIGN KEY (course_id) REFERENCES courses_normalized(course_id)
);

-- 成績テーブル
CREATE TABLE grades_normalized (
    student_id BIGINT,
    course_id VARCHAR(16),
    grade_type VARCHAR(32),
    score DECIMAL(5,2),
    max_score DECIMAL(5,2) DEFAULT 100.0,
    submission_date DATE,
    PRIMARY KEY (student_id, course_id, grade_type),
    FOREIGN KEY (student_id) REFERENCES students_normalized(student_id),
    FOREIGN KEY (course_id) REFERENCES courses_normalized(course_id)
);

-- スケジュールテーブル
CREATE TABLE schedules_normalized (
    schedule_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_id VARCHAR(16),
    classroom_id VARCHAR(16),
    schedule_date DATE,
```

```
    schedule_period INT,
    FOREIGN KEY (course_id) REFERENCES courses_normalized(course_id),
    FOREIGN KEY (classroom_id) REFERENCES classrooms_normalized(classroom_id),
    UNIQUE KEY unique_schedule (classroom_id, schedule_date, schedule_period)
);
```

ステップ4：3NFの適用

```
-- 推移関数従属を排除した最終設計

-- 学科テーブルの分離
CREATE TABLE departments_final (
    department_id INT AUTO_INCREMENT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL UNIQUE,
    department_building VARCHAR(50),
    department_head_teacher_id BIGINT
);

-- 専攻テーブルの分離
CREATE TABLE majors_final (
    major_id INT AUTO_INCREMENT PRIMARY KEY,
    major_name VARCHAR(100) NOT NULL UNIQUE,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments_final(department_id)
);

-- 教師テーブル（3NF準拠）
CREATE TABLE teachers_final (
    teacher_id BIGINT PRIMARY KEY,
    teacher_name VARCHAR(64) NOT NULL,
    teacher_email VARCHAR(255) UNIQUE,
    department_id INT,
    hire_date DATE,
    FOREIGN KEY (department_id) REFERENCES departments_final(department_id)
);

-- 学生テーブル（3NF準拠）
CREATE TABLE students_final (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(255) UNIQUE,
    student_grade INT CHECK (student_grade >= 1 AND student_grade <= 6),
    major_id INT,
    admission_date DATE,
    FOREIGN KEY (major_id) REFERENCES majors_final(major_id)
);

-- 講座テーブル（3NF準拠）
CREATE TABLE courses_final (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
```

```
    course_description TEXT,
    course_credits INT DEFAULT 2,
    course_difficulty ENUM('beginner', 'intermediate', 'advanced') DEFAULT
'beginner',
    teacher_id BIGINT NOT NULL,
    department_id INT,
    FOREIGN KEY (teacher_id) REFERENCES teachers_final(teacher_id),
    FOREIGN KEY (department_id) REFERENCES departments_final(department_id)
);

-- サンプルデータの挿入
INSERT INTO departments_final (department_name, department_building) VALUES
('情報工学科', 'A棟'),
('数学科', 'B棟'),
('物理学科', 'C棟');

INSERT INTO majors_final (major_name, department_id) VALUES
('コンピュータサイエンス', 1),
('情報システム', 1),
('応用数学', 2),
('理論物理', 3);

INSERT INTO teachers_final (teacher_id, teacher_name, teacher_email,
department_id, hire_date) VALUES
(101, '寺内鞍', 'terauchi@school.com', 1, '2020-04-01'),
(102, '佐野真', 'sano@school.com', 1, '2019-04-01'),
(103, '田中教授', 'tanaka@school.com', 2, '2018-04-01');

INSERT INTO students_final (student_id, student_name, student_email,
student_grade, major_id, admission_date) VALUES
(301, '黒沢春馬', 'kurosawa@school.com', 2, 1, '2024-04-01'),
(302, '新垣愛留', 'aragaki@school.com', 2, 2, '2024-04-01'),
(303, '柴崎春花', 'shibasaki@school.com', 1, 1, '2025-04-01');

INSERT INTO courses_final (course_id, course_name, course_description,
course_credits, teacher_id, department_id) VALUES
('CS101', 'ITのための基礎知識', 'コンピュータとITの基礎概念を学習', 2, 101, 1),
('CS102', 'UNIX入門', 'UNIXシステムの基本操作と概念', 2, 102, 1),
('MATH201', '微分積分学', '1変数および多変数の微分積分', 4, 103, 2);

-- 正規化後の複雑なクエリ例
SELECT
    s.student_name,
    s.student_grade,
    m.major_name,
    d.department_name,
    c.course_name,
    c.course_credits,
    t.teacher_name
FROM students_final s
JOIN majors_final m ON s.major_id = m.major_id
JOIN departments_final d ON m.department_id = d.department_id
JOIN enrollments_normalized e ON s.student_id = e.student_id
JOIN courses_final c ON e.course_id = c.course_id
```

```
JOIN teachers_final t ON c.teacher_id = t.teacher_id
WHERE s.student_grade = 2
ORDER BY s.student_name, c.course_name;
```

非正規化とは

非正規化は、パフォーマンス要件を満たすために、意図的にデータの冗長性を導入する設計手法です。

非正規化が必要な場面

1. **頻繁な結合クエリの最適化**
2. **リアルタイム性が要求される処理**
3. **大量データの集計処理**
4. **レポート機能の高速化**
5. **読み取り専用システムの最適化**

非正規化の手法

1. 計算済み値の格納

```
-- 学生の成績統計を事前計算して格納
ALTER TABLE students_final
ADD COLUMN total_credits INT DEFAULT 0,
ADD COLUMN gpa DECIMAL(3,2) DEFAULT 0.00,
ADD COLUMN total_courses INT DEFAULT 0;

-- 統計情報更新プロシージャ
DELIMITER //
CREATE PROCEDURE update_student_statistics(IN p_student_id BIGINT)
BEGIN
    DECLARE total_credits_val INT DEFAULT 0;
    DECLARE gpa_val DECIMAL(3,2) DEFAULT 0.00;
    DECLARE total_courses_val INT DEFAULT 0;

    -- 総単位数の計算
    SELECT COALESCE(SUM(c.course_credits), 0)
    INTO total_credits_val
    FROM enrollments_normalized e
    JOIN courses_final c ON e.course_id = c.course_id
    WHERE e.student_id = p_student_id AND e.status = 'completed';

    -- GPA計算（簡易版：平均点を4.0スケールに変換）
    SELECT COALESCE(AVG(g.score) / 25.0, 0.00)  -- 100点満点を4.0スケールに変換
    INTO gpa_val
    FROM grades_normalized g
    WHERE g.student_id = p_student_id;

    -- 総受講講座数
    SELECT COUNT(*)
    INTO total_courses_val
```

```
    FROM enrollments_normalized
    WHERE student_id = p_student_id;

    -- 学生テーブルを更新
    UPDATE students_final
    SET
        total_credits = total_credits_val,
        gpa = LEAST(gpa_val, 4.00),  -- 上限4.0
        total_courses = total_courses_val
    WHERE student_id = p_student_id;
END //
DELIMITER ;

-- 統計情報の更新
CALL update_student_statistics(301);
CALL update_student_statistics(302);
CALL update_student_statistics(303);

-- 高速な学生一覧取得（結合不要）
SELECT
    student_name,
    student_grade,
    total_credits,
    gpa,
    total_courses
FROM students_final
WHERE gpa >= 3.5
ORDER BY gpa DESC;
```

2. 頻繁にアクセスされる結合結果の事前計算

```
-- 講座詳細情報の非正規化テーブル
CREATE TABLE course_details_denormalized (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128),
    course_description TEXT,
    course_credits INT,
    course_difficulty ENUM('beginner', 'intermediate', 'advanced'),
    teacher_id BIGINT,
    teacher_name VARCHAR(64),
    teacher_email VARCHAR(255),
    department_id INT,
    department_name VARCHAR(100),
    department_building VARCHAR(50),
    enrolled_students_count INT DEFAULT 0,
    average_grade DECIMAL(5,2) DEFAULT 0.00,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- 非正規化テーブルの初期化プロシージャ
DELIMITER //
```

```
CREATE PROCEDURE initialize_course_details_denormalized()
BEGIN
    TRUNCATE TABLE course_details_denormalized;

    INSERT INTO course_details_denormalized
        (course_id, course_name, course_description, course_credits,
course_difficulty,
         teacher_id, teacher_name, teacher_email, department_id, department_name,
department_building,
         enrolled_students_count, average_grade)
    SELECT
        c.course_id,
        c.course_name,
        c.course_description,
        c.course_credits,
        c.course_difficulty,
        c.teacher_id,
        t.teacher_name,
        t.teacher_email,
        d.department_id,
        d.department_name,
        d.department_building,
        COALESCE(student_count.count, 0) as enrolled_students_count,
        COALESCE(avg_grades.avg_score, 0.00) as average_grade
    FROM courses_final c
    JOIN teachers_final t ON c.teacher_id = t.teacher_id
    JOIN departments_final d ON c.department_id = d.department_id
    LEFT JOIN (
        SELECT course_id, COUNT(*) as count
        FROM enrollments_normalized
        WHERE status = 'enrolled'
        GROUP BY course_id
    ) student_count ON c.course_id = student_count.course_id
    LEFT JOIN (
        SELECT course_id, AVG(score) as avg_score
        FROM grades_normalized
        GROUP BY course_id
    ) avg_grades ON c.course_id = avg_grades.course_id;
END //
DELIMITER ;

-- 初期化実行
CALL initialize_course_details_denormalized();

-- 高速な講座検索（結合不要）
SELECT
    course_name,
    teacher_name,
    department_name,
    enrolled_students_count,
    average_grade
FROM course_details_denormalized
WHERE department_name = '情報工学科'
```

```
  AND enrolled_students_count > 0
ORDER BY average_grade DESC;
```

3. 履歴データの効率化

```
-- 月次成績サマリテーブル（レポート用）
CREATE TABLE monthly_grade_summary (
    summary_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    year_month VARCHAR(7),  -- YYYY-MM
    student_id BIGINT,
    student_name VARCHAR(100),  -- 非正規化：名前を重複格納
    course_id VARCHAR(16),
    course_name VARCHAR(128),   -- 非正規化：講座名を重複格納
    grade_count INT DEFAULT 0,
    average_score DECIMAL(5,2) DEFAULT 0.00,
    highest_score DECIMAL(5,2) DEFAULT 0.00,
    lowest_score DECIMAL(5,2) DEFAULT 0.00,

    UNIQUE KEY unique_monthly_summary (year_month, student_id, course_id),
    INDEX idx_year_month (year_month),
    INDEX idx_student_performance (student_id, average_score)
);

-- 月次サマリ生成プロシージャ
DELIMITER //
CREATE PROCEDURE generate_monthly_grade_summary(IN target_year_month VARCHAR(7))
BEGIN
    -- 該当月のデータを削除
    DELETE FROM monthly_grade_summary WHERE year_month = target_year_month;

    -- 新しいサマリを生成
    INSERT INTO monthly_grade_summary
        (year_month, student_id, student_name, course_id, course_name,
         grade_count, average_score, highest_score, lowest_score)
    SELECT
        target_year_month,
        s.student_id,
        s.student_name,  -- 非正規化
        c.course_id,
        c.course_name,   -- 非正規化
        COUNT(g.score) as grade_count,
        ROUND(AVG(g.score), 2) as average_score,
        MAX(g.score) as highest_score,
        MIN(g.score) as lowest_score
    FROM students_final s
    JOIN grades_normalized g ON s.student_id = g.student_id
    JOIN courses_final c ON g.course_id = c.course_id
    WHERE DATE_FORMAT(g.submission_date, '%Y-%m') = target_year_month
    GROUP BY s.student_id, s.student_name, c.course_id, c.course_name
    HAVING grade_count > 0;
END //
```

```
DELIMITER ;

-- 月次サマリ生成
CALL generate_monthly_grade_summary('2025-05');

-- 高速な月次レポート生成
SELECT
    student_name,
    course_name,
    grade_count,
    average_score,
    CASE
        WHEN average_score >= 90 THEN 'A'
        WHEN average_score >= 80 THEN 'B'
        WHEN average_score >= 70 THEN 'C'
        WHEN average_score >= 60 THEN 'D'
        ELSE 'F'
    END as letter_grade
FROM monthly_grade_summary
WHERE year_month = '2025-05'
ORDER BY student_name, average_score DESC;
```

正規化vs非正規化の判断基準

判断マトリックス

要因	正規化を選ぶ	非正規化を選ぶ
データ整合性	重要	許容範囲内
更新頻度	高い	低い
読み取り頻度	普通	非常に高い
クエリの複雑さ	許容範囲	簡略化が必要
ストレージコスト	重要	二次的
保守性	重要	パフォーマンス優先
レスポンス時間	許容範囲	厳しい要件

実践的な設計指針

```
-- ハイブリッド設計の例

-- 1. メインデータは正規化を維持
CREATE TABLE core_students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(255) UNIQUE,
    major_id INT,
```

```
    admission_date DATE,
    FOREIGN KEY (major_id) REFERENCES majors_final(major_id)
);

-- 2. 頻繁にアクセスされる集計データは非正規化
CREATE TABLE student_performance_cache (
    student_id BIGINT PRIMARY KEY,
    current_gpa DECIMAL(3,2),
    total_credits INT,
    courses_completed INT,
    last_activity_date DATE,
    performance_rank INT,
    cache_updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    FOREIGN KEY (student_id) REFERENCES core_students(student_id) ON DELETE
CASCADE
);

-- 3. リアルタイム更新の仕組み
DELIMITER //
CREATE TRIGGER update_performance_cache_on_grade
AFTER INSERT ON grades_normalized
FOR EACH ROW
BEGIN
    -- 成績が追加されたら性能キャッシュを更新
    CALL refresh_student_performance_cache(NEW.student_id);
END //

CREATE TRIGGER update_performance_cache_on_grade_update
AFTER UPDATE ON grades_normalized
FOR EACH ROW
BEGIN
    -- 成績が更新されたら性能キャッシュを更新
    CALL refresh_student_performance_cache(NEW.student_id);
    IF OLD.student_id != NEW.student_id THEN
        CALL refresh_student_performance_cache(OLD.student_id);
    END IF;
END //

CREATE PROCEDURE refresh_student_performance_cache(IN p_student_id BIGINT)
BEGIN
    DECLARE new_gpa DECIMAL(3,2);
    DECLARE new_credits INT;
    DECLARE new_courses INT;
    DECLARE new_activity DATE;

    -- GPA計算
    SELECT ROUND(AVG(score) / 25.0, 2) INTO new_gpa
    FROM grades_normalized
    WHERE student_id = p_student_id;

    -- 単位数計算
    SELECT COALESCE(SUM(c.course_credits), 0) INTO new_credits
```

```
    FROM enrollments_normalized e
    JOIN courses_final c ON e.course_id = c.course_id
    WHERE e.student_id = p_student_id AND e.status = 'completed';

    -- 完了講座数
    SELECT COUNT(*) INTO new_courses
    FROM enrollments_normalized
    WHERE student_id = p_student_id AND status = 'completed';

    -- 最終活動日
    SELECT MAX(submission_date) INTO new_activity
    FROM grades_normalized
    WHERE student_id = p_student_id;

    -- キャッシュ更新
    INSERT INTO student_performance_cache
        (student_id, current_gpa, total_credits, courses_completed,
last_activity_date)
    VALUES
        (p_student_id, COALESCE(new_gpa, 0.00), COALESCE(new_credits, 0),
         COALESCE(new_courses, 0), new_activity)
    ON DUPLICATE KEY UPDATE
        current_gpa = COALESCE(new_gpa, 0.00),
        total_credits = COALESCE(new_credits, 0),
        courses_completed = COALESCE(new_courses, 0),
        last_activity_date = new_activity,
        cache_updated_at = CURRENT_TIMESTAMP;
END //
DELIMITER ;

-- 高速な学生パフォーマンス取得
SELECT
    s.student_name,
    s.student_email,
    p.current_gpa,
    p.total_credits,
    p.courses_completed,
    p.last_activity_date,
    DATEDIFF(CURRENT_DATE, p.last_activity_date) as days_since_last_activity
FROM core_students s
JOIN student_performance_cache p ON s.student_id = p.student_id
WHERE p.current_gpa >= 3.0
ORDER BY p.current_gpa DESC, p.total_credits DESC;
```

パフォーマンス比較分析

正規化vs非正規化のベンチマーク

```
-- ベンチマーク用データ生成
DELIMITER //
CREATE PROCEDURE generate_benchmark_data()
```

```
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE j INT DEFAULT 1;

    -- 大量の学生データ生成
    WHILE i <= 1000 DO
        INSERT INTO core_students (student_id, student_name, student_email,
major_id, admission_date)
        VALUES (i, CONCAT('学生', i), CONCAT('student', i, '@school.com'),
                (i % 4) + 1, DATE_SUB(CURRENT_DATE, INTERVAL FLOOR(RAND() * 1460)
DAY));
        SET i = i + 1;
    END WHILE;

    -- 大量の成績データ生成
    SET i = 1;
    WHILE i <= 1000 DO
        SET j = 1;
        WHILE j <= 10 DO  -- 各学生10件の成績
            INSERT INTO grades_normalized (student_id, course_id, grade_type,
score, submission_date)
            VALUES (i, CONCAT('CS10', (j % 3) + 1),
                   CASE j % 4
                       WHEN 0 THEN '中間テスト'
                       WHEN 1 THEN 'レポート1'
                       WHEN 2 THEN '課題1'
                       ELSE '期末テスト'
                   END,
                   60 + RAND() * 40,  -- 60-100点のランダム
                   DATE_SUB(CURRENT_DATE, INTERVAL FLOOR(RAND() * 30) DAY));
            SET j = j + 1;
        END WHILE;
        SET i = i + 1;
    END WHILE;
END //
DELIMITER ;

-- パフォーマンステストプロシージャ
CREATE PROCEDURE benchmark_query_performance()
BEGIN
    DECLARE start_time TIMESTAMP(6);
    DECLARE end_time TIMESTAMP(6);
    DECLARE normalized_time INT;
    DECLARE denormalized_time INT;

    -- 正規化テーブルでの複雑なクエリ
    SET start_time = NOW(6);

    SELECT
        s.student_name,
        AVG(g.score) as avg_score,
        COUNT(g.score) as grade_count,
        MAX(g.submission_date) as last_submission
    FROM core_students s
```

```
    JOIN grades_normalized g ON s.student_id = g.student_id
    WHERE s.major_id IN (1, 2)
    GROUP BY s.student_id, s.student_name
    HAVING avg_score >= 75
    ORDER BY avg_score DESC
    LIMIT 20;

    SET end_time = NOW(6);
    SET normalized_time = TIMESTAMPDIFF(MICROSECOND, start_time, end_time);

    -- 非正規化キャッシュでのクエリ
    SET start_time = NOW(6);

    SELECT
        s.student_name,
        p.current_gpa * 25 as avg_score,  -- 4.0スケールを100点に戻す
        p.courses_completed as grade_count,
        p.last_activity_date as last_submission
    FROM core_students s
    JOIN student_performance_cache p ON s.student_id = p.student_id
    WHERE s.major_id IN (1, 2)
      AND p.current_gpa >= 3.0
    ORDER BY p.current_gpa DESC
    LIMIT 20;

    SET end_time = NOW(6);
    SET denormalized_time = TIMESTAMPDIFF(MICROSECOND, start_time, end_time);

    -- 結果出力
    SELECT
        normalized_time as normalized_microseconds,
        denormalized_time as denormalized_microseconds,
        ROUND(normalized_time / denormalized_time, 2) as performance_ratio,
        normalized_time - denormalized_time as time_saved_microseconds;
END //
DELIMITER ;

-- ベンチマーク実行
-- CALL generate_benchmark_data();  -- 大量データ生成（注意：時間がかかります）
-- CALL benchmark_query_performance();
```

練習問題

問題38-1：正規化の段階的適用

以下の非正規化テーブルを1NF、2NF、3NFの順で正規化してください：

```
CREATE TABLE library_records_unnormalized (
    record_id INT PRIMARY KEY,
    book_isbn VARCHAR(20),
    book_title VARCHAR(200),
```

```
    book_authors TEXT,  -- カンマ区切りの複数著者
    publisher_name VARCHAR(100),
    publisher_address VARCHAR(200),
    publisher_phone VARCHAR(20),
    student_id BIGINT,
    student_name VARCHAR(100),
    student_email VARCHAR(255),
    student_department VARCHAR(100),
    loan_date DATE,
    return_date DATE,
    fine_amount DECIMAL(8,2)
);
```

問題38-2：関数従属の分析

以下のテーブルで関数従属関係を特定し、どの正規形に違反しているか分析してください：

```
CREATE TABLE course_enrollment_data (
    student_id BIGINT,
    course_id VARCHAR(16),
    semester VARCHAR(10),
    student_name VARCHAR(100),
    course_name VARCHAR(128),
    instructor_id BIGINT,
    instructor_name VARCHAR(64),
    department_id INT,
    department_name VARCHAR(100),
    grade CHAR(2),
    credit_hours INT,

    PRIMARY KEY (student_id, course_id, semester)
);
```

問題38-3：非正規化の設計

高頻度でアクセスされる以下のクエリを最適化するために、適切な非正規化テーブルを設計してください：

```
-- 頻繁に実行されるクエリ
SELECT
    s.student_name,
    COUNT(DISTINCT e.course_id) as total_courses,
    AVG(g.score) as average_grade,
    SUM(c.course_credits) as total_credits,
    d.department_name
FROM students s
JOIN enrollments e ON s.student_id = e.student_id
JOIN courses c ON e.course_id = c.course_id
JOIN grades g ON s.student_id = g.student_id AND c.course_id = g.course_id
JOIN departments d ON s.department_id = d.department_id
```

```
WHERE e.status = 'completed'
GROUP BY s.student_id, s.student_name, d.department_name;
```

問題38-4：ハイブリッド設計

読み取り頻度が非常に高いレポートシステム用に、正規化されたメインテーブルと非正規化されたレポート用テーブルの両方を含むハイブリッド設計を作成してください。以下の要件を満たすこと：

1. メインデータは3NFを維持
2. レポート用の非正規化テーブルを作成
3. メインデータ更新時の自動同期機能
4. 月次・年次レポート高速生成

問題38-5：パフォーマンス最適化

以下の要件でテーブル設計を最適化してください：

- 学生の成績検索が1秒以内で完了すること
- 講座別統計の生成が3秒以内で完了すること
- データの整合性は保持すること
- ストレージ使用量は最小限に抑えること 正規化と非正規化の組み合わせによる最適解を提案してください。

問題38-6：複雑なスキーマ設計

オンライン学習プラットフォームのデータベーススキーマを設計してください：

- 学生、講師、コース、レッスン、課題、成績管理
- 動画視聴履歴、進捗管理
- 支払い情報、サブスクリプション管理
- リアルタイムの学習分析とレポート機能 正規化と非正規化を適切に組み合わせた包括的な設計を提示してください。

解答

解答38-1

```
-- Step 1: 第1正規形（1NF）への変換
-- 問題：book_authors が複数値を持っている

-- 1NF準拠テーブル
CREATE TABLE library_records_1nf (
    record_id INT PRIMARY KEY,
    book_isbn VARCHAR(20),
    book_title VARCHAR(200),
    book_author VARCHAR(100),  -- 単一著者
    publisher_name VARCHAR(100),
    publisher_address VARCHAR(200),
    publisher_phone VARCHAR(20),
```

```
    student_id BIGINT,
    student_name VARCHAR(100),
    student_email VARCHAR(255),
    student_department VARCHAR(100),
    loan_date DATE,
    return_date DATE,
    fine_amount DECIMAL(8,2)
);

-- 複数著者は別テーブルで管理
CREATE TABLE book_authors_1nf (
    book_isbn VARCHAR(20),
    author_name VARCHAR(100),
    author_order INT,
    PRIMARY KEY (book_isbn, author_name)
);

-- Step 2: 第2正規形（2NF）への変換
-- 問題：複合主キーに対する部分関数従属

-- 図書テーブル
CREATE TABLE books_2nf (
    book_isbn VARCHAR(20) PRIMARY KEY,
    book_title VARCHAR(200) NOT NULL,
    publisher_name VARCHAR(100),
    publisher_address VARCHAR(200),
    publisher_phone VARCHAR(20)
);

-- 学生テーブル
CREATE TABLE students_2nf (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(255) UNIQUE,
    student_department VARCHAR(100)
);

-- 貸出記録テーブル
CREATE TABLE loan_records_2nf (
    record_id INT AUTO_INCREMENT PRIMARY KEY,
    book_isbn VARCHAR(20),
    student_id BIGINT,
    loan_date DATE NOT NULL,
    return_date DATE,
    fine_amount DECIMAL(8,2) DEFAULT 0.00,

    FOREIGN KEY (book_isbn) REFERENCES books_2nf(book_isbn),
    FOREIGN KEY (student_id) REFERENCES students_2nf(student_id)
);

-- 著者テーブル
CREATE TABLE book_authors_2nf (
    book_isbn VARCHAR(20),
    author_name VARCHAR(100),
```

```
    author_order INT DEFAULT 1,
    PRIMARY KEY (book_isbn, author_name),
    FOREIGN KEY (book_isbn) REFERENCES books_2nf(book_isbn)
);

-- Step 3: 第3正規形（3NF）への変換
-- 問題：publisher情報の推移関数従属

-- 出版社テーブル
CREATE TABLE publishers_3nf (
    publisher_id INT AUTO_INCREMENT PRIMARY KEY,
    publisher_name VARCHAR(100) NOT NULL UNIQUE,
    publisher_address VARCHAR(200),
    publisher_phone VARCHAR(20)
);

-- 学科テーブル
CREATE TABLE departments_3nf (
    department_id INT AUTO_INCREMENT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL UNIQUE
);

-- 図書テーブル（3NF）
CREATE TABLE books_3nf (
    book_isbn VARCHAR(20) PRIMARY KEY,
    book_title VARCHAR(200) NOT NULL,
    publisher_id INT,
    publication_date DATE,
    FOREIGN KEY (publisher_id) REFERENCES publishers_3nf(publisher_id)
);

-- 学生テーブル（3NF）
CREATE TABLE students_3nf (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(255) UNIQUE,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments_3nf(department_id)
);

-- 貸出記録テーブル（3NF）
CREATE TABLE loan_records_3nf (
    record_id INT AUTO_INCREMENT PRIMARY KEY,
    book_isbn VARCHAR(20),
    student_id BIGINT,
    loan_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,
    fine_amount DECIMAL(8,2) DEFAULT 0.00,

    FOREIGN KEY (book_isbn) REFERENCES books_3nf(book_isbn),
    FOREIGN KEY (student_id) REFERENCES students_3nf(student_id),
    CHECK (due_date >= loan_date),
    CHECK (return_date IS NULL OR return_date >= loan_date)
```

```
);

-- 著者テーブル（3NF）
CREATE TABLE book_authors_3nf (
    book_isbn VARCHAR(20),
    author_name VARCHAR(100),
    author_order INT DEFAULT 1,
    PRIMARY KEY (book_isbn, author_name),
    FOREIGN KEY (book_isbn) REFERENCES books_3nf(book_isbn)
);
```

解答38-2

```
-- 関数従属の分析

/*
主キー: (student_id, course_id, semester)

関数従属関係:
1. student_id → student_name (部分関数従属 - 2NF違反)
2. course_id → course_name (部分関数従属 - 2NF違反)
3. course_id → instructor_id (部分関数従属 - 2NF違反)
4. course_id → credit_hours (部分関数従属 - 2NF違反)
5. instructor_id → instructor_name (推移関数従属 - 3NF違反)
6. instructor_id → department_id (推移関数従属 - 3NF違反)
7. department_id → department_name (推移関数従属 - 3NF違反)

結論: このテーブルは1NFだが、2NFと3NFの両方に違反している
*/

-- 正規化された設計

-- 学科テーブル
CREATE TABLE departments_analysis (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL UNIQUE
);

-- 教師テーブル
CREATE TABLE instructors_analysis (
    instructor_id BIGINT PRIMARY KEY,
    instructor_name VARCHAR(64) NOT NULL,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments_analysis(department_id)
);

-- 学生テーブル
CREATE TABLE students_analysis (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL
);
```

```
-- 講座テーブル
CREATE TABLE courses_analysis (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    instructor_id BIGINT,
    credit_hours INT,
    FOREIGN KEY (instructor_id) REFERENCES instructors_analysis(instructor_id)
);

-- 受講・成績テーブル
CREATE TABLE enrollments_analysis (
    student_id BIGINT,
    course_id VARCHAR(16),
    semester VARCHAR(10),
    grade CHAR(2),

    PRIMARY KEY (student_id, course_id, semester),
    FOREIGN KEY (student_id) REFERENCES students_analysis(student_id),
    FOREIGN KEY (course_id) REFERENCES courses_analysis(course_id)
);
```

解答38-3

```
-- 非正規化テーブルの設計
CREATE TABLE student_academic_summary (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,  -- 非正規化
    department_name VARCHAR(100),         -- 非正規化
    total_courses INT DEFAULT 0,
    total_credits INT DEFAULT 0,
    completed_credits INT DEFAULT 0,
    average_grade DECIMAL(5,2) DEFAULT 0.00,
    gpa DECIMAL(3,2) DEFAULT 0.00,
    highest_grade DECIMAL(5,2) DEFAULT 0.00,
    lowest_grade DECIMAL(5,2) DEFAULT 0.00,
    last_enrollment_date DATE,
    academic_standing ENUM('Good', 'Warning', 'Probation') DEFAULT 'Good',
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_department (department_name),
    INDEX idx_gpa (gpa),
    INDEX idx_academic_standing (academic_standing)
);

-- サマリテーブル更新プロシージャ
DELIMITER //
CREATE PROCEDURE refresh_student_academic_summary(IN p_student_id BIGINT)
BEGIN
    DECLARE v_student_name VARCHAR(100);
    DECLARE v_department_name VARCHAR(100);
```

```
    DECLARE v_total_courses INT DEFAULT 0;
    DECLARE v_total_credits INT DEFAULT 0;
    DECLARE v_completed_credits INT DEFAULT 0;
    DECLARE v_average_grade DECIMAL(5,2) DEFAULT 0.00;
    DECLARE v_gpa DECIMAL(3,2) DEFAULT 0.00;
    DECLARE v_highest_grade DECIMAL(5,2) DEFAULT 0.00;
    DECLARE v_lowest_grade DECIMAL(5,2) DEFAULT 0.00;
    DECLARE v_last_enrollment DATE;
    DECLARE v_academic_standing ENUM('Good', 'Warning', 'Probation') DEFAULT
'Good';

    -- 基本情報取得
    SELECT s.student_name, d.department_name
    INTO v_student_name, v_department_name
    FROM students s
    JOIN departments d ON s.department_id = d.department_id
    WHERE s.student_id = p_student_id;

    -- 学習統計計算
    SELECT
        COUNT(DISTINCT e.course_id),
        COALESCE(SUM(c.course_credits), 0),
        COALESCE(SUM(CASE WHEN e.status = 'completed' THEN c.course_credits ELSE 0
END), 0),
        COALESCE(AVG(g.score), 0.00),
        COALESCE(AVG(g.score) / 25.0, 0.00),  -- GPA計算
        COALESCE(MAX(g.score), 0.00),
        COALESCE(MIN(g.score), 0.00),
        MAX(e.enrollment_date)
    INTO
        v_total_courses, v_total_credits, v_completed_credits,
        v_average_grade, v_gpa, v_highest_grade, v_lowest_grade, v_last_enrollment
    FROM enrollments e
    JOIN courses c ON e.course_id = c.course_id
    LEFT JOIN grades g ON e.student_id = g.student_id AND e.course_id =
g.course_id
    WHERE e.student_id = p_student_id;

    -- 学習状況判定
    SET v_academic_standing = CASE
        WHEN v_gpa >= 3.0 THEN 'Good'
        WHEN v_gpa >= 2.0 THEN 'Warning'
        ELSE 'Probation'
    END;

    -- サマリテーブル更新
    INSERT INTO student_academic_summary
        (student_id, student_name, department_name, total_courses, total_credits,
         completed_credits, average_grade, gpa, highest_grade, lowest_grade,
         last_enrollment_date, academic_standing)
    VALUES
        (p_student_id, v_student_name, v_department_name, v_total_courses,
v_total_credits,
         v_completed_credits, v_average_grade, v_gpa, v_highest_grade,
```

```
    v_lowest_grade,
             v_last_enrollment, v_academic_standing)
        ON DUPLICATE KEY UPDATE
            student_name = v_student_name,
            department_name = v_department_name,
            total_courses = v_total_courses,
            total_credits = v_total_credits,
            completed_credits = v_completed_credits,
            average_grade = v_average_grade,
            gpa = v_gpa,
            highest_grade = v_highest_grade,
            lowest_grade = v_lowest_grade,
            last_enrollment_date = v_last_enrollment,
            academic_standing = v_academic_standing,
            last_updated = CURRENT_TIMESTAMP;
    END //
    DELIMITER ;

    -- 高速クエリ例
    SELECT
        student_name,
        department_name,
        total_courses,
        completed_credits,
        gpa,
        academic_standing
    FROM student_academic_summary
    WHERE department_name = '情報工学科'
      AND academic_standing = 'Good'
    ORDER BY gpa DESC, completed_credits DESC;
```

解答38-4

```
-- ハイブリッド設計：正規化メインテーブル + 非正規化レポートテーブル

-- 1. 正規化されたメインテーブル（3NF準拠）
CREATE TABLE main_students (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    major_id INT,
    admission_date DATE,
    FOREIGN KEY (major_id) REFERENCES majors(major_id)
);

CREATE TABLE main_courses (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128) NOT NULL,
    teacher_id BIGINT,
    credits INT,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);
```

```
CREATE TABLE main_enrollments (
    student_id BIGINT,
    course_id VARCHAR(16),
    enrollment_date DATE,
    status ENUM('enrolled', 'completed', 'dropped'),
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES main_students(student_id),
    FOREIGN KEY (course_id) REFERENCES main_courses(course_id)
);

CREATE TABLE main_grades (
    student_id BIGINT,
    course_id VARCHAR(16),
    grade_type VARCHAR(32),
    score DECIMAL(5,2),
    submission_date DATE,
    PRIMARY KEY (student_id, course_id, grade_type),
    FOREIGN KEY (student_id, course_id) REFERENCES main_enrollments(student_id,
course_id)
);

-- 2. 非正規化レポートテーブル
CREATE TABLE report_monthly_summary (
    summary_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    report_month DATE,  -- 月初日
    student_id BIGINT,
    student_name VARCHAR(100),      -- 非正規化
    major_name VARCHAR(100),        -- 非正規化
    courses_enrolled INT DEFAULT 0,
    courses_completed INT DEFAULT 0,
    total_assignments INT DEFAULT 0,
    assignments_submitted INT DEFAULT 0,
    average_score DECIMAL(5,2) DEFAULT 0.00,
    credits_earned INT DEFAULT 0,
    attendance_rate DECIMAL(5,2) DEFAULT 0.00,

    UNIQUE KEY unique_monthly_student (report_month, student_id),
    INDEX idx_report_month (report_month),
    INDEX idx_student_performance (student_id, average_score)
);

CREATE TABLE report_annual_summary (
    summary_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    academic_year INT,
    student_id BIGINT,
    student_name VARCHAR(100),      -- 非正規化
    major_name VARCHAR(100),        -- 非正規化
    year_level INT,
    total_courses INT DEFAULT 0,
    completed_courses INT DEFAULT 0,
    cumulative_gpa DECIMAL(3,2) DEFAULT 0.00,
    total_credits INT DEFAULT 0,
    academic_status VARCHAR(50),
```

```
    honors_level VARCHAR(50),

    UNIQUE KEY unique_annual_student (academic_year, student_id),
    INDEX idx_academic_year (academic_year),
    INDEX idx_gpa (cumulative_gpa)
);

-- 3. 自動同期システム
DELIMITER //

-- 月次サマリ更新プロシージャ
CREATE PROCEDURE sync_monthly_summary(IN target_month DATE)
BEGIN
    DECLARE first_day DATE;
    DECLARE last_day DATE;

    SET first_day = DATE_FORMAT(target_month, '%Y-%m-01');
    SET last_day = LAST_DAY(first_day);

    -- 既存データ削除
    DELETE FROM report_monthly_summary WHERE report_month = first_day;

    -- 新データ生成
    INSERT INTO report_monthly_summary
        (report_month, student_id, student_name, major_name, courses_enrolled,
         courses_completed, total_assignments, assignments_submitted,
average_score, credits_earned)
    SELECT
        first_day,
        s.student_id,
        s.student_name,
        m.major_name,
        COUNT(DISTINCT e.course_id) as courses_enrolled,
        COUNT(DISTINCT CASE WHEN e.status = 'completed' THEN e.course_id END) as
courses_completed,
        COUNT(g.score) as total_assignments,
        COUNT(CASE WHEN g.score IS NOT NULL THEN 1 END) as assignments_submitted,
        COALESCE(AVG(g.score), 0.00) as average_score,
        COALESCE(SUM(CASE WHEN e.status = 'completed' THEN c.credits ELSE 0 END),
0) as credits_earned
    FROM main_students s
    JOIN majors m ON s.major_id = m.major_id
    LEFT JOIN main_enrollments e ON s.student_id = e.student_id
        AND e.enrollment_date BETWEEN first_day AND last_day
    LEFT JOIN main_courses c ON e.course_id = c.course_id
    LEFT JOIN main_grades g ON s.student_id = g.student_id
        AND g.submission_date BETWEEN first_day AND last_day
    GROUP BY s.student_id, s.student_name, m.major_name;
END //

-- 年次サマリ更新プロシージャ
CREATE PROCEDURE sync_annual_summary(IN target_year INT)
BEGIN
    DELETE FROM report_annual_summary WHERE academic_year = target_year;
```

```
    INSERT INTO report_annual_summary
        (academic_year, student_id, student_name, major_name, year_level,
         total_courses, completed_courses, cumulative_gpa, total_credits,
academic_status)
    SELECT
        target_year,
        s.student_id,
        s.student_name,
        m.major_name,
        target_year - YEAR(s.admission_date) + 1 as year_level,
        COUNT(DISTINCT e.course_id) as total_courses,
        COUNT(DISTINCT CASE WHEN e.status = 'completed' THEN e.course_id END) as
completed_courses,
        COALESCE(AVG(g.score) / 25.0, 0.00) as cumulative_gpa,
        COALESCE(SUM(CASE WHEN e.status = 'completed' THEN c.credits ELSE 0 END),
0) as total_credits,
        CASE
            WHEN AVG(g.score) >= 90 THEN 'Excellent'
            WHEN AVG(g.score) >= 80 THEN 'Good'
            WHEN AVG(g.score) >= 70 THEN 'Satisfactory'
            ELSE 'Needs Improvement'
        END as academic_status
    FROM main_students s
    JOIN majors m ON s.major_id = m.major_id
    LEFT JOIN main_enrollments e ON s.student_id = e.student_id
    LEFT JOIN main_courses c ON e.course_id = c.course_id
    LEFT JOIN main_grades g ON s.student_id = g.student_id
    WHERE YEAR(s.admission_date) <= target_year
    GROUP BY s.student_id, s.student_name, m.major_name, year_level;
END //

-- 自動同期トリガー
CREATE TRIGGER sync_on_grade_insert
AFTER INSERT ON main_grades
FOR EACH ROW
BEGIN
    CALL sync_monthly_summary(NEW.submission_date);
END //

CREATE TRIGGER sync_on_enrollment_update
AFTER UPDATE ON main_enrollments
FOR EACH ROW
BEGIN
    IF OLD.status != NEW.status THEN
        CALL sync_monthly_summary(CURRENT_DATE);
    END IF;
END //

DELIMITER ;

-- 4. 高速レポート生成
-- 月次レポート
SELECT
```

```
    student_name,
    major_name,
    courses_enrolled,
    courses_completed,
    average_score,
    credits_earned
FROM report_monthly_summary
WHERE report_month = '2025-05-01'
ORDER BY average_score DESC;

-- 年次レポート
SELECT
    major_name,
    COUNT(*) as student_count,
    AVG(cumulative_gpa) as avg_gpa,
    SUM(total_credits) as total_credits,
    academic_status,
    COUNT(*) as status_count
FROM report_annual_summary
WHERE academic_year = 2025
GROUP BY major_name, academic_status
ORDER BY major_name, avg_gpa DESC;
```

解答38-5

```
-- パフォーマンス最適化設計

-- 1. 正規化コアテーブル（データ整合性重視）
CREATE TABLE core_students_optimized (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE,
    major_id INT,
    admission_date DATE,
    is_active BOOLEAN DEFAULT TRUE,

    INDEX idx_major_active (major_id, is_active),
    INDEX idx_admission_date (admission_date)
);

CREATE TABLE core_grades_optimized (
    grade_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_id VARCHAR(16) NOT NULL,
    grade_type VARCHAR(32) NOT NULL,
    score DECIMAL(5,2),
    max_score DECIMAL(5,2) DEFAULT 100,
    submission_date DATE,

    FOREIGN KEY (student_id) REFERENCES core_students_optimized(student_id),
    INDEX idx_student_submission (student_id, submission_date),
```

```
    INDEX idx_course_type (course_id, grade_type),
    INDEX idx_score_range (score)
);

-- 2. 高速検索用非正規化テーブル
CREATE TABLE fast_student_grades (
    student_id BIGINT PRIMARY KEY,
    student_name VARCHAR(100),           -- 非正規化
    major_name VARCHAR(100),             -- 非正規化
    total_grades INT DEFAULT 0,
    current_semester_avg DECIMAL(5,2) DEFAULT 0.00,
    cumulative_avg DECIMAL(5,2) DEFAULT 0.00,
    highest_score DECIMAL(5,2) DEFAULT 0.00,
    lowest_score DECIMAL(5,2) DEFAULT 0.00,
    last_grade_date DATE,
    grade_trend ENUM('improving', 'stable', 'declining') DEFAULT 'stable',
    performance_rank INT DEFAULT 0,
    cache_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    INDEX idx_cumulative_avg (cumulative_avg),
    INDEX idx_performance_rank (performance_rank),
    INDEX idx_major_performance (major_name, cumulative_avg)
);

-- 3. 講座統計専用テーブル
CREATE TABLE fast_course_statistics (
    course_id VARCHAR(16) PRIMARY KEY,
    course_name VARCHAR(128),            -- 非正規化
    teacher_name VARCHAR(64),            -- 非正規化
    enrolled_count INT DEFAULT 0,
    grades_count INT DEFAULT 0,
    average_score DECIMAL(5,2) DEFAULT 0.00,
    pass_rate DECIMAL(5,2) DEFAULT 0.00,
    difficulty_index DECIMAL(5,2) DEFAULT 0.00,
    completion_rate DECIMAL(5,2) DEFAULT 0.00,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_average_score (average_score),
    INDEX idx_pass_rate (pass_rate),
    INDEX idx_teacher_course (teacher_name, average_score)
);

-- 4. 高性能更新システム
DELIMITER //

-- 学生成績キャッシュ更新（最適化版）
CREATE PROCEDURE fast_update_student_cache(IN p_student_id BIGINT)
BEGIN
    DECLARE v_student_name VARCHAR(100);
    DECLARE v_major_name VARCHAR(100);
    DECLARE v_total_grades INT;
    DECLARE v_current_avg DECIMAL(5,2);
    DECLARE v_cumulative_avg DECIMAL(5,2);
```

```
    DECLARE v_highest DECIMAL(5,2);
    DECLARE v_lowest DECIMAL(5,2);
    DECLARE v_last_date DATE;
    DECLARE v_trend ENUM('improving', 'stable', 'declining');

    -- 効率的な一回のクエリで全統計を取得
    SELECT
        s.student_name,
        m.major_name,
        COUNT(g.grade_id),
        COALESCE(AVG(CASE WHEN g.submission_date >= DATE_SUB(CURRENT_DATE,
INTERVAL 3 MONTH)
                          THEN g.score END), 0),
        COALESCE(AVG(g.score), 0),
        COALESCE(MAX(g.score), 0),
        COALESCE(MIN(g.score), 0),
        MAX(g.submission_date)
    INTO v_student_name, v_major_name, v_total_grades, v_current_avg,
         v_cumulative_avg, v_highest, v_lowest, v_last_date
    FROM core_students_optimized s
    LEFT JOIN majors m ON s.major_id = m.major_id
    LEFT JOIN core_grades_optimized g ON s.student_id = g.student_id
    WHERE s.student_id = p_student_id
    GROUP BY s.student_id, s.student_name, m.major_name;

    -- トレンド計算
    SET v_trend = CASE
        WHEN v_current_avg > v_cumulative_avg + 5 THEN 'improving'
        WHEN v_current_avg < v_cumulative_avg - 5 THEN 'declining'
        ELSE 'stable'
    END;

    -- キャッシュ更新
    INSERT INTO fast_student_grades
        (student_id, student_name, major_name, total_grades, current_semester_avg,
         cumulative_avg, highest_score, lowest_score, last_grade_date,
grade_trend)
    VALUES
        (p_student_id, v_student_name, v_major_name, v_total_grades,
v_current_avg,
         v_cumulative_avg, v_highest, v_lowest, v_last_date, v_trend)
    ON DUPLICATE KEY UPDATE
        student_name = v_student_name,
        major_name = v_major_name,
        total_grades = v_total_grades,
        current_semester_avg = v_current_avg,
        cumulative_avg = v_cumulative_avg,
        highest_score = v_highest,
        lowest_score = v_lowest,
        last_grade_date = v_last_date,
        grade_trend = v_trend;
END //

-- 講座統計更新（最適化版）
```

```
CREATE PROCEDURE fast_update_course_statistics(IN p_course_id VARCHAR(16))
BEGIN
    DECLARE v_course_name VARCHAR(128);
    DECLARE v_teacher_name VARCHAR(64);
    DECLARE v_enrolled_count INT;
    DECLARE v_grades_count INT;
    DECLARE v_average_score DECIMAL(5,2);
    DECLARE v_pass_rate DECIMAL(5,2);
    DECLARE v_completion_rate DECIMAL(5,2);

    SELECT
        c.course_name,
        t.teacher_name,
        enrollment_stats.enrolled_count,
        grade_stats.grades_count,
        grade_stats.average_score,
        grade_stats.pass_rate,
        enrollment_stats.completion_rate
    INTO v_course_name, v_teacher_name, v_enrolled_count, v_grades_count,
         v_average_score, v_pass_rate, v_completion_rate
    FROM courses c
    JOIN teachers t ON c.teacher_id = t.teacher_id
    LEFT JOIN (
        SELECT
            e.course_id,
            COUNT(*) as enrolled_count,
            COUNT(CASE WHEN e.status = 'completed' THEN 1 END) * 100.0 / COUNT(*)
as completion_rate
        FROM enrollments e
        WHERE e.course_id = p_course_id
        GROUP BY e.course_id
    ) enrollment_stats ON c.course_id = enrollment_stats.course_id
    LEFT JOIN (
        SELECT
            g.course_id,
            COUNT(*) as grades_count,
            AVG(g.score) as average_score,
            AVG(CASE WHEN g.score >= 60 THEN 100.0 ELSE 0 END) as pass_rate
        FROM core_grades_optimized g
        WHERE g.course_id = p_course_id
        GROUP BY g.course_id
    ) grade_stats ON c.course_id = grade_stats.course_id
    WHERE c.course_id = p_course_id;

    INSERT INTO fast_course_statistics
        (course_id, course_name, teacher_name, enrolled_count, grades_count,
         average_score, pass_rate, completion_rate)
    VALUES
        (p_course_id, v_course_name, v_teacher_name, COALESCE(v_enrolled_count,
0),
         COALESCE(v_grades_count, 0), COALESCE(v_average_score, 0),
         COALESCE(v_pass_rate, 0), COALESCE(v_completion_rate, 0))
    ON DUPLICATE KEY UPDATE
        course_name = v_course_name,
```

```
        teacher_name = v_teacher_name,
        enrolled_count = COALESCE(v_enrolled_count, 0),
        grades_count = COALESCE(v_grades_count, 0),
        average_score = COALESCE(v_average_score, 0),
        pass_rate = COALESCE(v_pass_rate, 0),
        completion_rate = COALESCE(v_completion_rate, 0);
END //

-- 自動更新トリガー（最適化版）
CREATE TRIGGER optimized_grade_update
AFTER INSERT ON core_grades_optimized
FOR EACH ROW
BEGIN
    -- 非同期的な更新のため、バッチ処理フラグを設定
    INSERT IGNORE INTO update_queue (table_name, record_id, update_type)
    VALUES ('student_cache', NEW.student_id, 'grade_change');

    INSERT IGNORE INTO update_queue (table_name, record_id, update_type)
    VALUES ('course_stats', NEW.course_id, 'grade_change');
END //

DELIMITER ;

-- 5. 高速検索クエリ例
-- 学生成績検索（1秒以内）
SELECT
    student_name,
    major_name,
    cumulative_avg,
    grade_trend,
    performance_rank
FROM fast_student_grades
WHERE major_name = '情報工学科'
  AND cumulative_avg >= 80
ORDER BY performance_rank
LIMIT 50;

-- 講座統計生成（3秒以内）
SELECT
    course_name,
    teacher_name,
    enrolled_count,
    average_score,
    pass_rate,
    difficulty_index
FROM fast_course_statistics
WHERE average_score >= 70
ORDER BY pass_rate DESC, average_score DESC;
```

解答38-6

```sql
-- オンライン学習プラットフォームの包括的スキーマ設計

-- 1. 正規化されたコアエンティティ（3NF準拠）

-- ユーザー管理
CREATE TABLE users (
    user_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    user_type ENUM('student', 'instructor', 'admin') NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE
);

CREATE TABLE user_profiles (
    user_id BIGINT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    avatar_url VARCHAR(500),
    bio TEXT,
    timezone VARCHAR(50) DEFAULT 'UTC',
    language VARCHAR(10) DEFAULT 'en',
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

-- カテゴリ・コース管理
CREATE TABLE categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(100) NOT NULL,
    parent_category_id INT,
    description TEXT,
    FOREIGN KEY (parent_category_id) REFERENCES categories(category_id)
);

CREATE TABLE courses (
    course_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_title VARCHAR(200) NOT NULL,
    course_description TEXT,
    instructor_id BIGINT NOT NULL,
    category_id INT,
    difficulty_level ENUM('beginner', 'intermediate', 'advanced') DEFAULT
'beginner',
    estimated_duration_hours INT,
    price DECIMAL(10,2),
    currency VARCHAR(3) DEFAULT 'USD',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    is_published BOOLEAN DEFAULT FALSE,

    FOREIGN KEY (instructor_id) REFERENCES users(user_id),
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
```

```
);

-- レッスン・コンテンツ管理
CREATE TABLE lessons (
    lesson_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_id BIGINT NOT NULL,
    lesson_title VARCHAR(200) NOT NULL,
    lesson_description TEXT,
    lesson_order INT NOT NULL,
    content_type ENUM('video', 'text', 'quiz', 'assignment') NOT NULL,
    content_url VARCHAR(500),
    duration_minutes INT,
    is_preview BOOLEAN DEFAULT FALSE,

    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE,
    INDEX idx_course_order (course_id, lesson_order)
);

CREATE TABLE lesson_content (
    content_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    lesson_id BIGINT NOT NULL,
    content_type ENUM('video', 'text', 'pdf', 'code', 'quiz') NOT NULL,
    content_data JSON,  -- 柔軟なコンテンツ格納
    file_url VARCHAR(500),
    file_size BIGINT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (lesson_id) REFERENCES lessons(lesson_id) ON DELETE CASCADE
);

-- 課題・評価管理
CREATE TABLE assignments (
    assignment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    course_id BIGINT NOT NULL,
    lesson_id BIGINT,
    assignment_title VARCHAR(200) NOT NULL,
    description TEXT,
    assignment_type ENUM('quiz', 'project', 'essay', 'code') NOT NULL,
    max_score DECIMAL(5,2) DEFAULT 100,
    time_limit_minutes INT,
    due_date TIMESTAMP,
    instructions JSON,

    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE,
    FOREIGN KEY (lesson_id) REFERENCES lessons(lesson_id) ON DELETE SET NULL
);

-- 受講・進捗管理
CREATE TABLE enrollments (
    enrollment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    course_id BIGINT NOT NULL,
    enrolled_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP NULL,
```

```
    progress_percentage DECIMAL(5,2) DEFAULT 0.00,
    last_accessed TIMESTAMP,
    status ENUM('active', 'completed', 'paused', 'cancelled') DEFAULT 'active',

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE,
    UNIQUE KEY unique_enrollment (user_id, course_id)
);

CREATE TABLE lesson_progress (
    progress_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    lesson_id BIGINT NOT NULL,
    watched_duration_seconds INT DEFAULT 0,
    completion_percentage DECIMAL(5,2) DEFAULT 0.00,
    is_completed BOOLEAN DEFAULT FALSE,
    first_accessed TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_accessed TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (lesson_id) REFERENCES lessons(lesson_id) ON DELETE CASCADE,
    UNIQUE KEY unique_lesson_progress (user_id, lesson_id)
);

-- 支払い・サブスクリプション管理
CREATE TABLE subscription_plans (
    plan_id INT AUTO_INCREMENT PRIMARY KEY,
    plan_name VARCHAR(100) NOT NULL,
    plan_type ENUM('monthly', 'yearly', 'lifetime') NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    currency VARCHAR(3) DEFAULT 'USD',
    features JSON,
    is_active BOOLEAN DEFAULT TRUE
);

CREATE TABLE payments (
    payment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    course_id BIGINT,
    plan_id INT,
    amount DECIMAL(10,2) NOT NULL,
    currency VARCHAR(3) DEFAULT 'USD',
    payment_method ENUM('credit_card', 'paypal', 'bank_transfer') NOT NULL,
    payment_status ENUM('pending', 'completed', 'failed', 'refunded') DEFAULT
'pending',
    transaction_id VARCHAR(100) UNIQUE,
    processed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE SET NULL,
    FOREIGN KEY (plan_id) REFERENCES subscription_plans(plan_id) ON DELETE SET
NULL,
    INDEX idx_user_payment (user_id, processed_at),
    INDEX idx_status (payment_status)
```

```
);

CREATE TABLE user_subscriptions (
    subscription_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    plan_id INT NOT NULL,
    payment_id BIGINT,
    start_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    end_date TIMESTAMP NOT NULL,
    status ENUM('active', 'expired', 'cancelled') DEFAULT 'active',
    auto_renewal BOOLEAN DEFAULT TRUE,

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (plan_id) REFERENCES subscription_plans(plan_id),
    FOREIGN KEY (payment_id) REFERENCES payments(payment_id),
    INDEX idx_user_subscription (user_id, status),
    INDEX idx_expiry (end_date, status)
);

-- 2. 非正規化されたパフォーマンステーブル

-- 学習分析用高速テーブル
CREATE TABLE analytics_user_summary (
    user_id BIGINT PRIMARY KEY,
    username VARCHAR(50),                    -- 非正規化
    full_name VARCHAR(101),                  -- 非正規化（first_name + last_name）
    user_type ENUM('student', 'instructor', 'admin'),
    total_courses_enrolled INT DEFAULT 0,
    total_courses_completed INT DEFAULT 0,
    total_lessons_watched INT DEFAULT 0,
    total_watch_time_hours DECIMAL(8,2) DEFAULT 0.00,
    average_completion_rate DECIMAL(5,2) DEFAULT 0.00,
    last_learning_activity TIMESTAMP,
    learning_streak_days INT DEFAULT 0,
    skill_level ENUM('beginner', 'intermediate', 'advanced') DEFAULT 'beginner',
    preferred_category_id INT,
    preferred_category_name VARCHAR(100),    -- 非正規化
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_completion_rate (average_completion_rate),
    INDEX idx_skill_level (skill_level),
    INDEX idx_category_preference (preferred_category_id)
);

-- コース統計用高速テーブル
CREATE TABLE analytics_course_summary (
    course_id BIGINT PRIMARY KEY,
    course_title VARCHAR(200),               -- 非正規化
    instructor_name VARCHAR(101),            -- 非正規化
    category_name VARCHAR(100),              -- 非正規化
    total_enrollments INT DEFAULT 0,
    active_enrollments INT DEFAULT 0,
    completion_count INT DEFAULT 0,
    completion_rate DECIMAL(5,2) DEFAULT 0.00,
```

```
    average_rating DECIMAL(3,2) DEFAULT 0.00,
    total_revenue DECIMAL(12,2) DEFAULT 0.00,
    avg_watch_time_per_lesson DECIMAL(8,2) DEFAULT 0.00,
    student_engagement_score DECIMAL(5,2) DEFAULT 0.00,
    last_enrollment_date TIMESTAMP,
    performance_trend ENUM('improving', 'stable', 'declining') DEFAULT 'stable',
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_completion_rate (completion_rate),
    INDEX idx_rating (average_rating),
    INDEX idx_revenue (total_revenue)
);

-- リアルタイム学習活動ログ
CREATE TABLE analytics_learning_sessions (
    session_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    course_id BIGINT NOT NULL,
    lesson_id BIGINT,
    session_start TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    session_end TIMESTAMP,
    duration_seconds INT,
    device_type ENUM('desktop', 'tablet', 'mobile') DEFAULT 'desktop',
    browser VARCHAR(50),
    ip_address VARCHAR(45),
    location_country VARCHAR(50),
    actions_taken JSON,                    -- クリック、一時停止、巻き戻し等

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    INDEX idx_user_session (user_id, session_start),
    INDEX idx_course_session (course_id, session_start),
    PARTITION BY RANGE (UNIX_TIMESTAMP(session_start)) (
        PARTITION p_2025_01 VALUES LESS THAN (UNIX_TIMESTAMP('2025-02-01')),
        PARTITION p_2025_02 VALUES LESS THAN (UNIX_TIMESTAMP('2025-03-01')),
        PARTITION p_2025_03 VALUES LESS THAN (UNIX_TIMESTAMP('2025-04-01')),
        PARTITION p_2025_04 VALUES LESS THAN (UNIX_TIMESTAMP('2025-05-01')),
        PARTITION p_2025_05 VALUES LESS THAN (UNIX_TIMESTAMP('2025-06-01')),
        PARTITION p_future VALUES LESS THAN MAXVALUE
    )
);

-- 3. 自動更新・同期システム
DELIMITER //

-- ユーザー学習サマリ更新
CREATE PROCEDURE update_user_analytics(IN p_user_id BIGINT)
BEGIN
    DECLARE v_username VARCHAR(50);
    DECLARE v_full_name VARCHAR(101);
    DECLARE v_user_type ENUM('student', 'instructor', 'admin');
    DECLARE v_total_enrolled INT DEFAULT 0;
    DECLARE v_total_completed INT DEFAULT 0;
    DECLARE v_total_lessons INT DEFAULT 0;
    DECLARE v_total_watch_time DECIMAL(8,2) DEFAULT 0.00;
```

```
    DECLARE v_avg_completion DECIMAL(5,2) DEFAULT 0.00;
    DECLARE v_last_activity TIMESTAMP;
    DECLARE v_preferred_category_id INT;
    DECLARE v_preferred_category_name VARCHAR(100);

    -- 基本情報取得
    SELECT
        u.username,
        CONCAT(p.first_name, ' ', p.last_name),
        u.user_type
    INTO v_username, v_full_name, v_user_type
    FROM users u
    LEFT JOIN user_profiles p ON u.user_id = p.user_id
    WHERE u.user_id = p_user_id;

    -- 学習統計計算
    SELECT
        COUNT(DISTINCT e.course_id),
        COUNT(DISTINCT CASE WHEN e.status = 'completed' THEN e.course_id END),
        COUNT(DISTINCT lp.lesson_id),
        COALESCE(SUM(lp.watched_duration_seconds) / 3600.0, 0),
        COALESCE(AVG(e.progress_percentage), 0),
        MAX(e.last_accessed)
    INTO
        v_total_enrolled, v_total_completed, v_total_lessons,
        v_total_watch_time, v_avg_completion, v_last_activity
    FROM enrollments e
    LEFT JOIN lesson_progress lp ON e.user_id = lp.user_id
    WHERE e.user_id = p_user_id;

    -- 好みのカテゴリ分析
    SELECT
        c.category_id,
        cat.category_name
    INTO v_preferred_category_id, v_preferred_category_name
    FROM enrollments e
    JOIN courses c ON e.course_id = c.course_id
    JOIN categories cat ON c.category_id = cat.category_id
    WHERE e.user_id = p_user_id
    GROUP BY c.category_id, cat.category_name
    ORDER BY COUNT(*) DESC
    LIMIT 1;

    -- サマリテーブル更新
    INSERT INTO analytics_user_summary
        (user_id, username, full_name, user_type, total_courses_enrolled,
         total_courses_completed, total_lessons_watched, total_watch_time_hours,
         average_completion_rate, last_learning_activity, preferred_category_id,
         preferred_category_name)
    VALUES
        (p_user_id, v_username, v_full_name, v_user_type, v_total_enrolled,
         v_total_completed, v_total_lessons, v_total_watch_time,
         v_avg_completion, v_last_activity, v_preferred_category_id,
         v_preferred_category_name)
```

```
    ON DUPLICATE KEY UPDATE
        username = v_username,
        full_name = v_full_name,
        total_courses_enrolled = v_total_enrolled,
        total_courses_completed = v_total_completed,
        total_lessons_watched = v_total_lessons,
        total_watch_time_hours = v_total_watch_time,
        average_completion_rate = v_avg_completion,
        last_learning_activity = v_last_activity,
        preferred_category_id = v_preferred_category_id,
        preferred_category_name = v_preferred_category_name;
END //

-- コース統計更新
CREATE PROCEDURE update_course_analytics(IN p_course_id BIGINT)
BEGIN
    DECLARE v_course_title VARCHAR(200);
    DECLARE v_instructor_name VARCHAR(101);
    DECLARE v_category_name VARCHAR(100);
    DECLARE v_total_enrollments INT;
    DECLARE v_active_enrollments INT;
    DECLARE v_completion_count INT;
    DECLARE v_completion_rate DECIMAL(5,2);
    DECLARE v_total_revenue DECIMAL(12,2);
    DECLARE v_avg_watch_time DECIMAL(8,2);

    -- コース基本情報
    SELECT
        c.course_title,
        CONCAT(p.first_name, ' ', p.last_name),
        cat.category_name
    INTO v_course_title, v_instructor_name, v_category_name
    FROM courses c
    JOIN users u ON c.instructor_id = u.user_id
    LEFT JOIN user_profiles p ON u.user_id = p.user_id
    LEFT JOIN categories cat ON c.category_id = cat.category_id
    WHERE c.course_id = p_course_id;

    -- 受講統計
    SELECT
        COUNT(*),
        COUNT(CASE WHEN status = 'active' THEN 1 END),
        COUNT(CASE WHEN status = 'completed' THEN 1 END),
        CASE WHEN COUNT(*) > 0
             THEN COUNT(CASE WHEN status = 'completed' THEN 1 END) * 100.0 /
COUNT(*)
             ELSE 0 END
    INTO v_total_enrollments, v_active_enrollments, v_completion_count,
v_completion_rate
    FROM enrollments
    WHERE course_id = p_course_id;

    -- 収益計算
    SELECT COALESCE(SUM(amount), 0)
```

```
    INTO v_total_revenue
    FROM payments
    WHERE course_id = p_course_id AND payment_status = 'completed';

    -- 平均視聴時間
    SELECT COALESCE(AVG(lp.watched_duration_seconds / 60.0), 0)
    INTO v_avg_watch_time
    FROM lesson_progress lp
    JOIN lessons l ON lp.lesson_id = l.lesson_id
    WHERE l.course_id = p_course_id;

    -- 統計テーブル更新
    INSERT INTO analytics_course_summary
        (course_id, course_title, instructor_name, category_name,
         total_enrollments, active_enrollments, completion_count, completion_rate,
         total_revenue, avg_watch_time_per_lesson)
    VALUES
        (p_course_id, v_course_title, v_instructor_name, v_category_name,
         v_total_enrollments, v_active_enrollments, v_completion_count,
v_completion_rate,
         v_total_revenue, v_avg_watch_time)
    ON DUPLICATE KEY UPDATE
        course_title = v_course_title,
        instructor_name = v_instructor_name,
        category_name = v_category_name,
        total_enrollments = v_total_enrollments,
        active_enrollments = v_active_enrollments,
        completion_count = v_completion_count,
        completion_rate = v_completion_rate,
        total_revenue = v_total_revenue,
        avg_watch_time_per_lesson = v_avg_watch_time;
END //

-- リアルタイム更新トリガー
CREATE TRIGGER update_analytics_on_enrollment
AFTER INSERT ON enrollments
FOR EACH ROW
BEGIN
    CALL update_user_analytics(NEW.user_id);
    CALL update_course_analytics(NEW.course_id);
END //

CREATE TRIGGER update_analytics_on_progress
AFTER UPDATE ON lesson_progress
FOR EACH ROW
BEGIN
    IF NEW.completion_percentage != OLD.completion_percentage THEN
        CALL update_user_analytics(NEW.user_id);
    END IF;
END //

DELIMITER ;

-- 4. 高速レポート・分析クエリ例
```

```sql
-- ダッシュボード用KPI取得
SELECT
    'user_metrics' as metric_type,
    COUNT(DISTINCT user_id) as total_users,
    AVG(total_courses_enrolled) as avg_courses_per_user,
    AVG(average_completion_rate) as platform_completion_rate,
    SUM(total_watch_time_hours) as total_platform_hours
FROM analytics_user_summary
WHERE user_type = 'student'
UNION ALL
SELECT
    'course_metrics',
    COUNT(DISTINCT course_id),
    AVG(total_enrollments),
    AVG(completion_rate),
    SUM(total_revenue)
FROM analytics_course_summary;

-- 人気コースランキング
SELECT
    course_title,
    instructor_name,
    category_name,
    total_enrollments,
    completion_rate,
    average_rating,
    total_revenue
FROM analytics_course_summary
WHERE total_enrollments > 50
ORDER BY completion_rate DESC, total_enrollments DESC
LIMIT 20;

-- 学習者エンゲージメント分析
SELECT
    preferred_category_name,
    skill_level,
    COUNT(*) as user_count,
    AVG(average_completion_rate) as avg_completion,
    AVG(total_watch_time_hours) as avg_watch_time,
    AVG(learning_streak_days) as avg_streak
FROM analytics_user_summary
WHERE user_type = 'student'
  AND last_learning_activity >= DATE_SUB(NOW(), INTERVAL 30 DAY)
GROUP BY preferred_category_name, skill_level
ORDER BY user_count DESC;

-- 収益分析
SELECT
    DATE_FORMAT(processed_at, '%Y-%m') as month,
    COUNT(*) as transaction_count,
    SUM(amount) as total_revenue,
    AVG(amount) as avg_transaction_value,
    COUNT(DISTINCT user_id) as paying_users
```

```
FROM payments
WHERE payment_status = 'completed'
  AND processed_at >= DATE_SUB(NOW(), INTERVAL 12 MONTH)
GROUP BY DATE_FORMAT(processed_at, '%Y-%m')
ORDER BY month DESC;

-- リアルタイム学習活動
SELECT
    DATE_FORMAT(session_start, '%Y-%m-%d %H:00:00') as hour_bucket,
    COUNT(DISTINCT user_id) as active_users,
    COUNT(*) as total_sessions,
    AVG(duration_seconds / 60) as avg_session_minutes,
    device_type,
    COUNT(*) as device_sessions
FROM analytics_learning_sessions
WHERE session_start >= DATE_SUB(NOW(), INTERVAL 24 HOUR)
GROUP BY hour_bucket, device_type
ORDER BY hour_bucket DESC, device_sessions DESC;
```

まとめ

この章では、スキーマ設計における正規化と非正規化について詳しく学びました：

1. **正規化の理論と実践**：
   - 第1正規形（1NF）：原子値と繰り返しグループの排除
   - 第2正規形（2NF）：部分関数従属の排除
   - 第3正規形（3NF）：推移関数従属の排除
   - 段階的な正規化プロセス
2. **正規化の利点と課題**：
   - データ整合性の向上
   - 冗長性の排除
   - クエリの複雑化
   - パフォーマンスへの影響
3. **非正規化の戦略**：
   - 計算済み値の事前格納
   - 頻繁な結合結果の物理化
   - レポート専用テーブルの設計
   - キャッシュテーブルの活用
4. **ハイブリッド設計**：
   - 正規化コアテーブルの維持
   - 非正規化パフォーマンステーブルの追加
   - 自動同期システムの構築
   - 一貫性保証の仕組み

5. **設計判断基準**：
    - データ整合性要件
    - パフォーマンス要件
    - 更新頻度と読み取り頻度
    - 保守性とスケーラビリティ
6. **実践的な設計例**：
    - 学校管理システム
    - 図書館システム
    - オンライン学習プラットフォーム
    - 包括的なエンタープライズシステム

正規化と非正規化は対立する概念ではなく、システムの要件に応じて適切に組み合わせることが重要です。データの整合性を保ちつつ、必要なパフォーマンスを実現する**バランスの取れた設計**が、実用的なデータベースシステムの鍵となります。

これで第6章「データ定義言語（DDL）と管理」が完了しました。CREATE TABLE、ALTER TABLE、DROP/TRUNCATE、制約、AUTO_INCREMENT、マテリアライズドビュー、スキーマ設計まで、データベース構造の設計・管理に必要な包括的な知識を習得できました。
