19章 サブクエリを学ぼう

19章 サブクエリを学ぼう

より高度なデータ検索を実現するサブクエリの使い方を学びます。

本章の目標

- サブクエリとは何かを知ること
- サブクエリの基本的な使い方を理解すること
- サブクエリを活用した実践的なクエリを書けるようになること

19章 サブクエリとは

サブクエリとは、SQL文の中に入れ子（ネスト）にして記述される別のSELECT文のことです。

- メインのクエリの中に、別のクエリを組み込む
- 複雑な条件を表現できる
- JOINでは実現しにくい検索も可能になる
- 別名「入れ子クエリ」「ネストクエリ」とも呼ばれる

19章 サブクエリとは

サブクエリは**括弧 `()` で囲んで記述**します。

```
SELECT カラム名
FROM テーブル名
WHERE カラム名 = (サブクエリ);
```

サブクエリ部分が先に実行され、その結果をメインクエリで使用します。

基本的な構文

```
SELECT カラム名
FROM テーブル名
WHERE カラム名 演算子 (
    SELECT カラム名
    FROM テーブル名
    WHERE 条件
);
```

- 外側のクエリ：**メインクエリ（外部クエリ）**
- 内側のクエリ：**サブクエリ（内部クエリ）**

19章 サブクエリの実行順序

サブクエリは**内側から外側に向かって実行**されます。

1. **サブクエリ**が先に実行される
2. サブクエリの結果を使って**メインクエリ**が実行される

この順序を理解することが、サブクエリを使いこなす鍵です。

19章 サブクエリの実例

例えば「山田太郎さんと同じ年齢のユーザーを取得したい」場合を考えます。

通常の方法（2段階）

```sql
-- 1. 山田太郎さんの年齢を調べる
SELECT age FROM users WHERE name = '山田太郎';  -- 結果: 25

-- 2. 年齢が25のユーザーを取得
SELECT * FROM users WHERE age = 25;
```

サブクエリを使う方法（1回で完結）

```sql
SELECT * FROM users
WHERE age = (
    SELECT age FROM users WHERE name = '山田太郎'
);
```

19章 WHERE句でのサブクエリ

WHERE句でサブクエリを使う場合、サブクエリの結果が**1つの値を返す**必要があります。

```
SELECT name, price
FROM products
WHERE price > (
    SELECT AVG(price)
    FROM products
);
```

このクエリは「平均価格より高い商品」を取得します。

19章 IN演算子とサブクエリ

サブクエリが**複数の値を返す**場合は、`IN`演算子を使います。

```sql
SELECT name, department_id
FROM employees
WHERE department_id IN (
    SELECT id
    FROM departments
    WHERE location = '東京'
);
```

このクエリは「東京にある部署に所属する社員」を取得します。

19章 NOT IN演算子とサブクエリ

`NOT IN`を使うと、サブクエリの結果に**含まれない**データを取得できます。

```sql
SELECT name, price
FROM products
WHERE category_id NOT IN (
    SELECT id
    FROM categories
    WHERE name = '廃盤'
);
```

このクエリは「廃盤カテゴリに属さない商品」を取得します。

19章 EXISTS演算子とサブクエリ

EXISTS演算子は、サブクエリの結果が存在するかどうかを判定します。

```
SELECT name
FROM customers
WHERE EXISTS (
    SELECT 1
    FROM orders
    WHERE orders.customer_id = customers.id
);
```

このクエリは「注文履歴がある顧客」を取得します。

19章 NOT EXISTS演算子

`NOT EXISTS`を使うと、サブクエリの結果が**存在しない**場合にTRUEになります。

```sql
SELECT name
FROM customers
WHERE NOT EXISTS (
    SELECT 1
    FROM orders
    WHERE orders.customer_id = customers.id
);
```

このクエリは「注文履歴がない顧客」を取得します。

19章 FROM句でのサブクエリ

FROM句にサブクエリを記述することもできます。この場合、サブクエリの結果を**仮想テーブル**として扱います。

```
SELECT dept_name, avg_salary
FROM (
    SELECT department_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
) AS dept_avg
JOIN departments ON dept_avg.department_id = departments.id;
```

FROM句のサブクエリには**必ずエイリアス（別名）**を付ける必要があります。

19章 SELECT句でのサブクエリ

SELECT句にもサブクエリを記述できます。

```
SELECT
    name,
    price,
    (SELECT AVG(price) FROM products) AS avg_price,
    price - (SELECT AVG(price) FROM products) AS price_diff
FROM products;
```

各商品の価格と平均価格との差を一覧で取得できます。

相関サブクエリとは、メインクエリのテーブルを参照するサブクエリです。

```sql
SELECT name, salary
FROM employees e1
WHERE salary > (
    SELECT AVG(salary)
    FROM employees e2
    WHERE e2.department_id = e1.department_id
);
```

このクエリは「各部署の平均給与より高い給与をもらっている社員」を取得します。

19章 相関サブクエリの特徴

相関サブクエリは、メインクエリの**各行ごとに実行**されます。

- 通常のサブクエリ：1回だけ実行される
- 相関サブクエリ：メインクエリの行数分実行される

そのため、データ量が多い場合は**パフォーマンスに注意**が必要です。

19章 サブクエリとJOINの使い分け

同じ結果を得られる場合でも、サブクエリとJOINのどちらを使うか選択できます。

サブクエリが適している場合

- 条件判定に別テーブルのデータを使う
- EXISTS/NOT EXISTSで存在確認をする
- 集計結果を条件に使う

JOINが適している場合

- 複数テーブルのカラムを同時に取得する
- 大量データでパフォーマンスが重要

19章 サブクエリの実践例①

「最も高い商品と同じ価格の商品を取得」

```
SELECT name, price
FROM products
WHERE price = (
    SELECT MAX(price)
    FROM products
);
```

19章 サブクエリの実践例②

「注文されたことのある商品のみを取得」

```sql
SELECT name, price
FROM products
WHERE id IN (
    SELECT DISTINCT product_id
    FROM order_details
);
```

`DISTINCT`を使って重複を除いています。

19章 サブクエリの実践例③

「各カテゴリで最も高い商品を取得」

```sql
SELECT name, price, category_id
FROM products p1
WHERE price = (
    SELECT MAX(price)
    FROM products p2
    WHERE p2.category_id = p1.category_id
);
```

相関サブクエリを使った例です。

19章 サブクエリ使用時の注意点

1. **サブクエリは括弧で囲む**必要がある
2. **FROM句のサブクエリには必ずエイリアスを付ける**
3. WHERE句で使う場合、**返す値の数に注意**（単一値なら `=`、複数値なら `IN`）
4. **相関サブクエリはパフォーマンスに影響**する可能性がある
5. サブクエリが複雑になりすぎたら、**JOINでの書き換えを検討**する

19章 WITH句（CTE）とは

WITH句は、クエリの前に一時的な結果セット（共通テーブル式：CTE）を定義する構文です。

- **CTE**（Common Table Expression：共通テーブル式）とも呼ばれる
- サブクエリを**名前付きで定義**できる
- 複雑なクエリを**読みやすく整理**できる
- 同じサブクエリを**複数回参照**できる

19章 WITH句の基本構文

基本的な構文

```
WITH CTE名 AS (
    SELECT カラム名
    FROM テーブル名
    WHERE 条件
)
SELECT カラム名
FROM CTE名;
```

WITH句で定義したCTEは、そのクエリ内でテーブルのように使用できます。

19章 WITH句の実例

サブクエリを使った書き方とWITH句を使った書き方を比較します。

サブクエリを使った書き方

```
SELECT name, price FROM products
WHERE price > (
    SELECT AVG(price) FROM products
);
```

WITH句を使った書き方

```
WITH avg_price AS (
    SELECT AVG(price) AS price FROM products
)
SELECT name, price FROM products, avg_price
WHERE products.price > avg_price.price;
```

19章 WITH句のメリット

1. **可読性の向上**
    - 複雑なサブクエリに名前を付けて整理できる
    - クエリの意図が分かりやすくなる
2. **再利用性**
    - 同じサブクエリを複数回参照できる
    - コードの重複を避けられる
3. **保守性の向上**
    - 修正箇所が1か所で済む
    - デバッグしやすい

19章 WITH句の実践例①

「カテゴリごとの平均価格を計算し、それより高い商品を取得」

```sql
WITH category_avg AS (
    SELECT
        category_id,
        AVG(price) AS avg_price
    FROM products
    GROUP BY category_id
)
SELECT
    p.name,
    p.price,
    p.category_id,
    ca.avg_price
FROM products p
JOIN category_avg ca ON p.category_id = ca.category_id
WHERE p.price > ca.avg_price;
```

19章 WITH句の実践例②

「売上TOP10の商品の詳細を取得」

```sql
WITH top_products AS (
    SELECT
        product_id,
        SUM(quantity) AS total_sales
    FROM order_details
    GROUP BY product_id
    ORDER BY total_sales DESC
    LIMIT 10
)
SELECT
    p.name,
    p.price,
    tp.total_sales
FROM products p
JOIN top_products tp ON p.id = tp.product_id
ORDER BY tp.total_sales DESC;
```

19章 複数のCTEを定義する

WITH句では、カンマ区切りで複数のCTEを定義できます。

```
WITH
sales_summary AS (
    SELECT product_id, SUM(quantity) AS total_qty
    FROM order_details
    GROUP BY product_id
),
high_price AS (
    SELECT id, name, price
    FROM products
    WHERE price > 1000
)
SELECT
    hp.name,hp.price,ss.total_qty
FROM high_price hp
JOIN sales_summary ss ON hp.id = ss.product_id;
```

19章 CTEの連鎖

後のCTEで前のCTEを参照することもできます。

```
WITH
category_totals AS (
    SELECT category_id, COUNT(*) AS product_count
    FROM products
    GROUP BY category_id
),
large_categories AS (
    SELECT category_id
    FROM category_totals
    WHERE product_count > 10
)
SELECT p.name, p.category_id
FROM products p
JOIN large_categories lc ON p.category_id = lc.category_id;
```

19章 WITH句とサブクエリの使い分け

WITH句が適している場合

- サブクエリが複雑で長い
- 同じサブクエリを複数回使う
- クエリの可読性を重視する
- 段階的にデータを加工する

サブクエリが適している場合

- シンプルで短いクエリ
- 1回だけ使用する
- パフォーマンスが最優先

19章 WITH句使用時の注意点

1. **CTEは一時的**
    - そのクエリ内でのみ有効
    - クエリ実行後は消える
2. **インデックスが使えない場合がある**
    - CTEは一時的な結果セットのため
    - パフォーマンスに注意

19章 まとめ（1/3）

サブクエリについて

- **サブクエリ**とは、SQL文の中に入れ子にして記述される別のSELECT文
- サブクエリは**括弧で囲んで記述**する
- **内側から外側**に向かって実行される
- WHERE句、FROM句、SELECT句など、さまざまな場所で使用できる

19章 まとめ（2/3）

WITH句（CTE）について

- **WITH句**は、クエリの前に一時的な結果セットを定義する構文
- サブクエリを**名前付きで定義**できる
- 複雑なクエリを**読みやすく整理**できる
- **複数のCTE**をカンマ区切りで定義可能

19章 まとめ（3/3）

サブクエリとWITH句の比較：

項目	サブクエリ	WITH句（CTE）
可読性	複雑になると読みにくい	名前を付けて整理できる
再利用	できない（毎回記述）	同じクエリ内で再利用可能
保守性	修正箇所が複数になる	修正箇所が1か所で済む
適用場面	シンプルなクエリ	複雑なクエリ、段階的処理