

SQL基礎 学習資料













データベース操作の基本から実践応用まで



2025年8月18日

≡ 目次

SQL基礎学習 - 12セクション構成

- 1  SQL基礎概念とデータベース入門
- 2  データベース設計の基本
- 3  テーブル作成とデータ型
- 4  SELECT文の基本操作
- 5  WHERE句とフィルタリング
- 6  ソートとグループ化
- 7  結合（JOIN）操作
- 8  サブクエリとビュー
- 9  データ操作（INSERT、UPDATE、DELETE）
- 10  インデックスとパフォーマンス最適化
- 11  よくある質問（FAQ）
- 12  参考リンク・資料

SQL基礎概念とデータベース入門

SQL (Structured Query Language) は、リレーショナルデータベースを操作するための標準言語です。データの検索、挿入、更新、削除といった基本操作から高度な分析まで、幅広い用途で使用されています。





SQLの特徴

- ✓ 宣言的な言語（何を取得するかを指定）
- ✓ ANSI/ISO標準に準拠したポータブル性
- ✓ 複雑なデータ関係を効率的に処理
- ✓ 多くのRDBMS（Oracle、MySQL、PostgreSQL等）で共通

学習目標

SQLの基本構文を理解し、データベースからの効率的なデータ取得・操作ができるようになること

RDBMSの種類




-  **PostgreSQL** - オープンソース、高機能
-  **MySQL** - 高速、Webアプリケーション向け
-  **Oracle** - エンタープライズ向け
-  **SQLite** - 軽量、組み込み用途

データベース活用場面



- Webアプリケーションのユーザー管理
- ECサイトの商品・注文管理
- 業務システムの売上分析
- ログデータの集計・レポート作成

データベース設計の基本

データベースの構成要素

-  **テーブル（表）**
データを行（レコード）と列（フィールド）で管理する構造
-  **レコード（行）**
一つのエンティティ（例：1人のユーザー情報）
-  **フィールド（列）**
データの属性（例：名前、年齢、メールアドレス）

キーの概念

-  **主キー（Primary Key）** - レコードを一意に識別
-  **外部キー（Foreign Key）** - 他テーブルとの関連付け

正規化の基本

-  **第1正規形** - 繰り返し項目の排除
-  **第2正規形** - 部分関数従属の排除
-  **第3正規形** - 推移関数従属の排除

ER図の基本

- ・エンティティ（四角形）・関係（ひし形）
- ・属性（楕円形）・カーディナリティ

設計のメリット

- データ重複排除・更新異常防止
- 保守性向上・データ整合性確保

テーブル作成とデータ型

CREATE TABLE文の基本構文

```
CREATE TABLE テーブル名 (  
  列名1 データ型 制約,  
  列名2 データ型 制約,  
  PRIMARY KEY (列名)  
);
```

主要なデータ型

- # **INTEGER** - 整数値 (例: 年齢、ID)
- A **VARCHAR(n)** - 可変長文字列 (例: 名前、メール)
- 📅 **DATE** - 日付 (例: 生年月日、登録日)
- 🕒 **TIMESTAMP** - 日時 (例: 更新日時)
- 💰 **DECIMAL(m,n)** - 小数値 (例: 価格、評価)

主要な制約

- 1 **NOT NULL**
必須項目 (空値不可)
- 2 **UNIQUE**
一意性保証 (重複不可)
- 3 **PRIMARY KEY**
主キー (NOT NULL + UNIQUE)
- 4 **CHECK**
値の範囲制限

インデックスの基本

- 検索速度の向上
- 主キーには自動作成
- 頻繁に検索する列に設定
- 更新時のオーバーヘッド

SELECT文の基本操作

SELECT文の基本構文

```
SELECT 列名1, 列名2, ...  
FROM テーブル名;
```

基本的な使用例

- * 全列取得：SELECT * FROM users;
- 特定列取得：SELECT name, email FROM users;
- 別名使用：SELECT name AS 氏名 FROM users;
- 重複排除：SELECT DISTINCT department FROM employees;

SQLの実行順序

FROM → WHERE → SELECT → ORDER BY の順で処理される

FROM句の役割

- データを取得するテーブルを指定
- 複数テーブルの結合も可能
- テーブルに別名（エイリアス）付与可能

実用的なポイント

- SELECT * は開発時のみ使用
- 本番環境では必要列のみ指定
- AS句で分かりやすい列名を設定
- DISTINCT は重複が多い場合に使用

パフォーマンス考慮

- 不要な列は取得しない
- 大量データではLIMITを活用
- インデックスが効くクエリを心がける

WHERE句とフィルタリング

WHERE句の基本構文

```
SELECT 列名 FROM テーブル名  
WHERE 条件式;
```

比較演算子

=, <> - 等しい、等しくない

>, >=, <, <= - 大小比較

🔍 LIKE - パターンマッチング (% , _)

☰ IN - 複数値との比較

↔ BETWEEN - 範囲指定

? IS NULL - NULL値の判定

論理演算子

+ AND

すべての条件が真

? OR

いずれかの条件が真

! NOT

条件の否定

実用例

```
WHERE age >= 20 AND age <= 65
```

```
WHERE name LIKE '田中%'
```

```
WHERE status IN ('active', 'pending')
```

注意点

- NULL値は=では比較できない
- 文字列は大文字小文字を区別
- 日付は文字列として指定

ソートとグループ化

ORDER BY句（ソート）

```
SELECT * FROM テーブル名  
ORDER BY 列名 [ASC|DESC];
```



ASC（昇順）

小さい値から大きい値へ（デフォルト）



DESC（降順）

大きい値から小さい値へ


GROUP BY句（グループ化）

```
SELECT 列名, 集約関数  
FROM テーブル名  
GROUP BY 列名;
```

ポイント

GROUP BYで指定した列と集約関数のみをSELECTで指定可能

集約関数

 **COUNT(*)** - レコード数をカウント

+ **SUM(列名)** - 数値の合計

 **AVG(列名)** - 数値の平均

↑ **MAX(列名)** - 最大値

↓ **MIN(列名)** - 最小値

HAVING句

```
GROUP BY 列名  
HAVING 集約関数の条件;
```

グループ化後の結果をフィルタリング

実用例

- 部署別の人数集計
- 月別の売上合計
- 年齢帯別の平均給与
- 商品カテゴリ別の最高価格

結合（JOIN）操作

JOIN操作の基本構文

```
SELECT 列名  
FROM テーブル1  
[JOIN種類] テーブル2  
ON 結合条件;
```

JOIN操作の種類

INNER JOIN（内部結合）

両テーブルに一致するレコードのみ

LEFT JOIN（左外部結合）

左テーブル全体 + 右テーブル一致分

RIGHT JOIN（右外部結合）

右テーブル全体 + 左テーブル一致分

FULL OUTER JOIN

両テーブルの全レコードを取得

実用例

顧客と注文の結合

```
SELECT c.name, o.order_date  
FROM customers c  
INNER JOIN orders o  
ON c.id = o.customer_id;
```

商品とカテゴリの結合

```
SELECT p.name, cat.category_name  
FROM products p  
LEFT JOIN categories cat  
ON p.category_id = cat.id;
```

ポイント

- 結合キーにインデックスを設定
- テーブルエイリアス（別名）を活用
- 必要な列のみをSELECTで指定
- WHERE句で結果を絞り込み

自己結合

同一テーブル内での階層構造や関連データの取得に使用

サブクエリとビュー

サブクエリの基本

```
SELECT * FROM products
WHERE price > (
  SELECT AVG(price) FROM products
);
```

WHERE句でのサブクエリ

条件判定に別クエリの結果を使用

FROM句でのサブクエリ

サブクエリを仮想テーブルとして使用




関連サブクエリ

外部クエリの列を参照するサブクエリ

ビューの作成

```
CREATE VIEW view_name AS
SELECT 列名 FROM テーブル名
WHERE 条件;
```

ビューの活用

-  複雑なクエリを簡単なテーブルとして扱える
-  データアクセスのセキュリティ向上
-  再利用可能なロジックの定義

実用例

```
-- 顧客情報ビュー
CREATE VIEW customer_summary AS
SELECT c.name, COUNT(o.id) as order_count
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, c.name;
```

パフォーマンス考慮

- サブクエリは適切に使用
- EXISTS、NOT EXISTSの活用
- ビューはパフォーマンスに影響あり
- インデックスとの組み合わせを考慮

データ操作 (INSERT、UPDATE、DELETE)

INSERT文 (データ挿入)

```
INSERT INTO テーブル名 (列名1, 列名2, ...)  
VALUES (値1, 値2, ...);
```

UPDATE文 (データ更新)

```
UPDATE テーブル名  
SET 列名1 = 値1, 列名2 = 値2  
WHERE 条件;
```

DELETE文 (データ削除)

```
DELETE FROM テーブル名  
WHERE 条件;
```

注意点

UPDATE、DELETE文ではWHERE句を必ず指定すること。未指定では全レコードが対象になる

トランザクション制御

- ▶ **BEGIN/START TRANSACTION** - トランザクション開始
- ✓ **COMMIT** - 変更を確定
- ↺ **ROLLBACK** - 変更を取り消し

トランザクション例

```
BEGIN;  
UPDATE accounts SET balance = balance - 1000  
WHERE account_id = 1;  
UPDATE accounts SET balance = balance + 1000  
WHERE account_id = 2;  
COMMIT;
```

パフォーマンスのポイント

- 一括操作ではバッチ処理を活用
- インデックスがある列で条件指定
- 大量データの削除は分割実行

インデックスとパフォーマンス最適化

インデックスの仕組み

-  **検索速度の向上**
データへのアクセス時間を大幅短縮
-  **ソート演算の最適化**
ORDER BY句の処理が高速化
-  **JOIN演算の効率化**
結合キーにインデックスがある場合





EXPLAINの活用

```
EXPLAIN SELECT * FROM users  
WHERE age > 25;
```

```
EXPLAIN ANALYZE SELECT ...
```

クエリの実行計画やコストを確認

クエリ最適化の基本原則

-  SELECT * を避け、必要な列のみ取得
-  WHERE句で早めにデータを絞り込み
-  インデックスが効く条件を優先
-  サブクエリよりJOINを優先使用

パフォーマンスチューニング

- 頻繁に検索する列にインデックス作成
- 文字列の部分一致検索は前方一致を使用
- 結合キーには必ずインデックスを設定
- 統計情報を定期的に更新
- LIMIT句で取得件数を制限

インデックスの注意点

- 更新時のオーバーヘッド
- ストレージ容量の増加
- 不要インデックスの定期削除

よくある質問（FAQ）

❓ NULL値の扱い方は？

NULLは「値が不明」を意味し、=では比較できません。IS NULL、IS NOT NULLを使用してください。

🔠 文字化けが起こる場合は？

データベース、テーブル、接続の文字コード設定を確認し、UTF-8に統一することを推奨します。

🔍 大文字小文字の区別について

RDBMSによって異なります。MySQLはデフォルトで区別しませんが、PostgreSQLは区別します。

📅 日付データの扱い方は？

'YYYY-MM-DD'形式で文字列として指定するか、DATE()関数やDATETIME関数を使用します。

⚠ エラーが出た時の対処法は？

エラーメッセージをよく読み、構文エラー、テーブル名、列名のスペルミスを確認してください。

🕒 クエリが遅い場合は？

EXPLAINで実行計画を確認し、インデックスの追加やWHERE句の最適化を検討してください。

🔒 セキュリティ対策は？

SQLインジェクション対策として、プリペアドステートメントやパラメータ化クエリを使用してください。

参考リンク・資料

📖 公式ドキュメント

- 🔗 PostgreSQL公式ドキュメント
- 🔗 MySQL公式ドキュメント
- 🔗 SQLiteドキュメント
- 🔗 Oracle Databaseドキュメント

🏠 オンライン学習リソース

- 🌐 W3Schools SQL Tutorial
- 🌐 SQLBolt (インタラクティブ学習)
- 🎧 Khan AcademyのSQLコース

pgAdmin

PostgreSQL管理ツール

MySQL Workbench

MySQL開発環境

DBeaver

マルチDBクライアント

HeidiSQL

軽量SQLクライアント

🛠 次のステップ

- ストアドプロシージャの学習
- トリガーの作成と活用
- パフォーマンスチューニング実践
- NoSQLデータベースの学習

💡 学習のコツ

- 🔧 実際に手を動かして練習する
- 🔧 小さなデータセットから始める
- 🔧 エラーメッセージをしっかりと読む
- 🔧 EXPLAINでクエリの動作を理解

お疲れさまでした！

SQL基礎を習得できました。実際のプロジェクトで活用し、継続的な学習でスキルを伸ばしていきましょう！